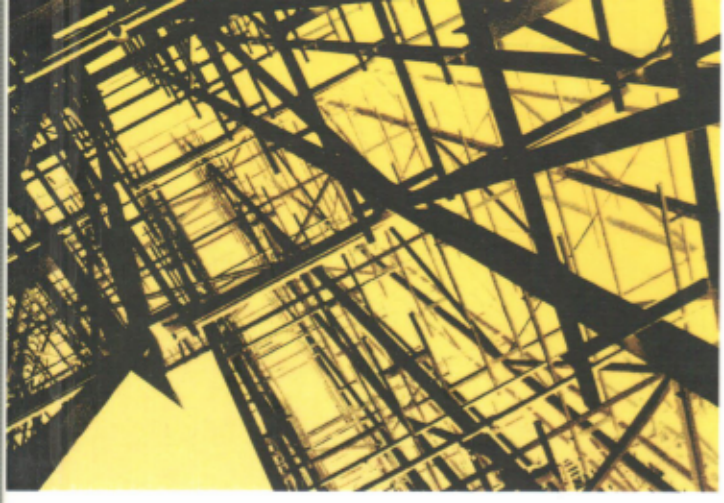


softwarearen ingeniaritza

[II. ATALA: garapen monolitikotik hiru mailako arkitekturara bezero/zerbitzariak bisitatuz]

Jose Ramon **Zubizarreta Aizpuru**

Udako Euskal **Unibertsitatea**

SOFTWAREAREN INGENIARITZA

[II. ATALA: Garapen monolitikotik hiru mailako arkitekturara bezero/zerbitzariak bisitatuz]

Jose Ramon Zubizarreta Aizpuru

Udako **Euskal Unibertsitatea**
Bilbo, 2008

EUSKO JAURLARITZA
GOBIERNO VASCO

HEZKUNTZA, UNIBERTSITATE
ETA IKERKETA SAILA
DEPARTAMENTO DE EDUCACIÓN
UNIVERSIDADES E INVESTIGACIÓN

«Liburu hau Hezkuntza, Unibertsitate eta
Ikerketa Sailaren laguntzaz argitaratu da»

© Udako Euskal Unibertsitatea
© Jose Ramon Zubizarreta Aizpuru

ISBN (bilduma osorako): 978-84-8438-090-0
ISBN (II. atalerako): 978-84-8438-165-5
Lege-gordailua: BI-1037-08

Inprimategia: CUATROAS, Bilbo
Azalaren diseinua: Iñigo Ordozgoiti
Hizkuntza-zuzenketen arduraduna: Ander Altuna Gabiola

Banatzaileak: UEU. Erribera 14, 1. D BILBO telf. 946790546 Faxa. 944793039
Helbide elektronikoa: argitalpenak@ueu.org
www.ueu.org

Elkar Banaketa: Igerabide, 88 DONOSTIA

Galarazita dago liburu honen kopia egitea, osoa nahiz zatikakoa, edozein modutara delarik ere, edizio honen Copyright-jabeen baimenik gabe.

Aurkibidea

IRAKURTZAILEARI 9

II. ATALA: Softwarearen garapenaren zenbait arlo 13
1. LEHEN HURBILKETA: GARAPEN MONOLITIKOA (AWT-SWT/Java/JDBC) 15
1.1. Erabiltzailearen interfaze grafikoaren eraikuntza: AWT-SWING. . . 15
1.1.1. Sarrera 15
1.1.2. Interfaze grafikoaren osagaiak (Component) 21
1.1.3. Interfaze grafikoen gertaeren kudeaketa 29
1.1.4. Diseinu-administratzaileak 31
1.2. Informazioaren iraunkortasunaren kudeaketa: JBDC 39
1.2.1. Sarrera 39
1.2.2. Kontrolatzaile motak. 40
1.2.3. Prozesu orokorraren ezaugarri nagusiak 42
1.2.4. JDBC 2.0 APIaren ezaugarri nagusiak 51
1.3. Laburpena 56
1.4. Oinarrizko bibliografia 57
1.5. Web gune interesgarriak 57

2. BIGARREN HURBILKETA: OINARRIZKO BEZERO/ZERBITZARI ARKITEKTURA: AWT-SWING/RMI/JDBC. 59
2.1. RMI (Remote Method Invocation) 59
2.2. Laburpena 75
2.3. Oinarrizko bibliografia 76
2.4. Web gune interesgarriak 76

3. HIRUGARREN HURBILKETA: HIRU MAILAKO ARKITEKTURA. . . 77
3.1. Applet/Html. 77
3.2. JSP/Servlet. 82
3.2.1. Servletak. 82
3.2.1.1. Sarrera. 82
3.2.1.2. Servleten bizi-zikloa. 83
3.2.1.3. Servleten barne-portaeraren azterketa. 84

3.2.2. JSP (Java Server Pages) orriak 108
3.2.2.1. Sarrera 108
3.2.2.2. JSP orriaren izaera. 109
3.3. J2EE teknologia. 128
3.3.1. Sarrera 128
3.3.2. EJB osagaiak garatzen J2EE teknologia erabilita. 130
3.3.3. J2EE teknologiaren bean motak.. 147
3.3.4. EJB egoeradun sesio-bean baten garapena. 163
3.3.5. Entitate-beanak. 168
3.3.5.1. Beanak kudeatutako iraunkortasuna.. 168
3.3.5.2. Edukiontziak kudeatutako iraunkortasuna. 180
3.3.5.3. Aplikazio oso baten garapena EJB beanak erabilita. 195
3.3.6. J2EE diseinu-patroiak 198
3.3.6.1. Aurkezpen maila: View Helper 199
3.3.6.2. Aurkezpen maila: Front Controller 202
3.3.6.3. Domeinu eta integrazio mailak: Session Façade. . . 209
3.3.6.4. Domeinu eta integrazio mailak: Business Delegate 212
3.3.6.5. Domeinu eta integrazio mailak: DAO (Data Access Object). 215
3.3.7. J2EE diseinu-patroien ama: Eredu-Bista-Kontrol eta Struts . . 219
3.4. Spring ingurune aurreratua 227
3.4.1. Sarrera 227
3.4.2. Spring ingurune aurreratuaren arkitektura orokorra 230
3.4.3. Mendekotasun-injekzioa 237
3.5. Laburpena 250
3.6. Oinarrizko bibliografia 251
3.7. Web gune interesgarriak 251

4. APLIKAZIO INFORMATIKOEN DISEINU MOTAK ETA IRAUNKORTASUNAREN TRATAERA AURRERATUAK 253
4.1. Sarrera 253
4.2. Ikuspegi prozedural aurreratua: Transaction Script eta iBATIS 255
4.3. Iraunkortasunaren trataera aurreratua: Hibernate 282
4.4. Domeinuaren logikaren trataera POJO-Façade bidez 301
4.5. Aplikazioaren osagaien inplementazioa EJB 3.0 erabilita 313
4.6. Objektuetara zuzendutako iraunkortasuna: db4o 332
4.7. Laburpena 352
4.8. Oinarrizko bibliografia 353
4.9. Web gune interesgarriak 353

5. LAUGARREN HURBILKETA: ZERBITZUETARA ZUZENDUTAKO ARKITEKTURA (WEB ZERBITZUAK) 355
5.1. JAXP (Java API for XML Processing) 358
5.2. JAXB (Java Architecture for XML Binding) 364
5.3. SAAJ (SOAP with Attachments API for Java) 379
5.4. JAX-RPC (Java API for XML-based Remote Procedure Calls) 390
5.5. JAXR (Java API for XML Registries) 403
5.6. Adibide konplexu baten diseinua eta garapena web zerbitzuak erabilita 407
5.7. Laburpena 416
5.8. Oinarrizko bibliografia 416
5.9. Web gune interesgarriak 417

6. RUBY ON RAILS 419
6.1. Sarrera 419
6.2. Osagaien iraunkortasuna Ruby on Rails erabilita 422
6.3. Laburpena 433
6.4. Oinarrizko bibliografia 433
6.5. Web gune interesgarriak 434

7. ERANSKINAK: ARIKETAK ETA ORDENAGAILUAREKIN EGITEKO LABORATEGIAK 435
7.1. Java: ariketa prozeduralak 435
7.2. Java: RMI ariketak 446
7.3. Ariketak: hiru mailako arkitektura 461
7.4. Ariketak: JGL (Java Generic Library) 464
7.5. Ariketak: EJB (Enterprise JavaBeans) 469
7.6. Ariketak: diseinu-patroiak 471
7.7. Ariketak: UML (Unified Modeling Language) 477
7.8. Ordenagailuarekin egiteko laborategiak 495

8. BIBLIOGRAFIA 519

Irakurtzaileari

Zure hasierako honako hau, irakurle, nire urteetako jardueraren emaitza duzu. Bidaia, luzea eta neketsua izan da, agian luzeegia[1] dinoste supazterrekoek. Lehenengo liburuan esan nizun Axularren hitzekin[2], bi partetan banatuko nituela azaldu beharrekoak. Hark, lehendabizikoarekin, bekataria bekatutik aterarazi eta konbertsiora eraman nahi zuen. Bigarrenarekin, ordea, konbertitu den gizona, kristau bizitzan barna aitzinatu, gidatu eta bultzarazi nahi zuen, hau da, Villasanteren hitzetan, kristau bizitzaren alderdi arrai eta positiboak agertu.

Jakin badakigu, orduko garaiak ez zirela egokienak eztitasunetan ibiltzeko eta agian horregatik, agindutako bigarren parterik gabe gelditu gara. Edo, hurrean, Axular bera, beldurraren beldurrez beldurtuta, ez zen ausartu hain txukun eta dotore gelditu zitzaiona hedatzen. Nik nire inperfekzioekin aspaldi ikasi nuenez bizitzen, agindutako bigarren atalarekin natorkizu.

Bigarren liburu honetan, hasiera-hasieratik, arazo larri batekin aurkitu nintzen: nola eman batasuna aurkeztu eta aztertu nahi ziren hainbat teknologia eta diseinu motari.

Argi edukita teknologiak ezin zuela hautaketa baldintzatu, aplikazioaren arkitekturan ipini zen arreta batasun hori bilatzeko, eta, hiru mailako arkitektura aukeratu zen liburuaren norabidea antolatzeko. Arkitektura aukeratu ondoren, nola egituratu liburuaren atalburuak? Informatikan hain ezaguna den goranzko (bottom-up) estrategia erabilita.

Hala, lehen atalburuan, lehen hurbilketa egiten da aipatutako hiru mailako arkitekturaren garapen monolitikoaren osagai nagusiak aurkeztuta. Lehen liburuan bezala, honako honetan ere, Javaren teknologia erabili da arlo bakoitzaren inplementazioa azaltzeko, iturburu-kode mailan.

Ez dakit asmatu dudan, baina, aurkezpenean, saiatu naiz, teknologia, azaldu nahi zen kontzeptuaren menpe jartzen, teknologia aldakorra baita beti eta zer esanik ez informatikari lotua dagoena. Ez hala, kontzeptua, patroia, arkitektura, horiek nekez aldatzen dira.

---

1. Puede que no valga la pena escribir, tan lejos de la poesía. Por la boca vive el pez. Fito Cabrales.

2. Eta hala deliberatu nuen, buru-eragotzkarri bezala, liburutto baten, bi partetan partiturik, gero hunen gaiñean egitera.

Bigarren atalburuan, geruza berria jartzen diogu lehenengoan aurkeztutakoari, bezeroak eta zerbitzariak urruneko ostalarietan jarrita. Hori lortzeko, Javaren RMI teknologia aurkeztuko da urruneko deien nondik norakoa antolatzeko.

Hirugarren atalburuan, liburuaren muinean murgilduko gara hiru mailako arkitekturaren ezaugarriak aztertuta. Lehendabizi, aurkezpen maila egituratzeko, Javaren teknologiak eskaintzen duen oinarrizko azpiegitura azalduko da appletak eta JSP orriak aztertuta. Javaren teknologian, bezeroaren eta zerbitzariaren arteko oinarrizko komunikazioa servletak erabilita gauzatzen denez, teknologia horren barne-portaeraren bizi-zikloa azalduko da jarraian.

Klaseak erabili ordez, osagaiak erabiltzen badira aplikazioaren diseinua inplementatzeko, J2EE teknologiaren baliabideak ondo baino hobeto datozkigu arkitekturaren geruza bakoitza inplementatzeko. Gainera, geruzen arteko eta geruza barneko komunikazioa ondo egituratu nahi bada, ebazpen generikoak eraikita, behin eta berriz errepikatzen diren arazoei, laguntza paregabea eskaintzen diote J2EE diseinu-patroiek aplikazioaren diseinatzaileari eta programatzaileari.

Amaitzeko hirugarren atalburu hau, Spring ingurune aurreratua aztertuko da. Edukiontzi leunaren ezaugarriak txertatu nahi badira aplikazioaren arkitekturan, Spring teknologiak hainbat baliabide eskaintzen ditu helburu hori gauzatzeko. Hala, aplikazioaren ezaugarri asko, edukiontziaren tiraniatik zertxobait urruntzen dira eta aplikazioa malguagoa bihurtzen da osagaiak probatzeko eta zerbitzuz hornitzeko.

Aplikazioaren analistak, hiru mailako arkitektura erantsi nahi dionean eraiki nahi den aplikazio informatikoari, erabaki garrantzitsuak hartu behar ditu diseinua eta inplementazioa bideratzeko eta egokitzeko lanean ari den erakunde informatikoaren lan egiteko erara eta aplikazioak eskatzen duen konplexutasun mailara. Laugarren atalburuan, korapilo horiek askatzeko behar diren teknologiak eta diseinu motak aurkeztuko dira.

Aplikazioaren konplexutasun maila eta erakundearen eskarmentua txikiak badira, ikuspegi prozedurala erabilita antola daiteke domeinuaren logika, Transaction Script diseinu-patroiak erabilita. Iraunkortasunaren maila antolatzeko, erakundearen eskarmentuak eta teknologia mailak sano baldintzatuko dute hautaketa. Hala nola, txikiak badira, JDBC erabili beharko du zuzenean eta gordinki, atzipenen antolaketa guztia programatzailearen eskuetan geldituz.

JDBC teknologia ondo ezagutzen duenak jakin badaki ezen iraunkortasunaren kudeaketaren hainbat eragiketa behin eta berriz errepikatzen direla balio erantsirik gehitu gabe aplikazioaren iturburu-kodeari. Hala, aplikazioaren iturburu-kodea, *zikinduta* gelditzen da, programatzaileak kode arrotza txertatu behar baitu aplikazioaren exekuzio-fluxuan, iraunkortasunaren nondik norako guztiak zehetuta adierazteko.

Hibernate eta iBATIS tresnek, besteak beste, nabari hobetu dute iraunkortasunaren trataera, automatizatuz iraunkortasunaren lan mekanikoa eta objektu-taulen arteko mapaketa. Tresna horiek erabilita, domeinuaren logikan kokatzen du programatzaileak arreta nagusia.

Begi-bistakoa da, aplikazioaren konplexutasun maila igo ahala, hiru mailako arkitekturaren geruzen arteko komunikazioa korapilatu egiten dela. Horrelako egoeretan, ezin ordainduzko laguntza ematen du Façade patroi ezagunak domeinuaren logika kapsulatzeko. Hala, aurkezpen mailak behar dituen zerbitzuak, zuzenean hornitzen dizkio patroia inplementatzen duen osagaiak, berak kapsulatzen baitu domeinuaren logika guztia.

Aplikazioaren osagaien inplementazioa POJO[3] erako objektuak erabilita egiten bada, hainbat zerbitzu garrantzitsu, adibidez, transakzioak kudeatzeko zerbitzuak edota segurtasunaren kudeaketa, programatzailearen ardurapean gelditzen dira. Hala, aplikazioa malguagoa bihurtzen da eta edukiontzi leunak erabilita kudea daitezke osagaiak.

Ingurune horren ordez, EJB 3.0 espezifikazioa inplementatzen duen teknologia erabiltzen bada osagaiak eraiki eta kudeatzeko, zerbitzuen kudeaketa edukiontziak berak gauzatuko du automatikoki, ingurune trinkoagoan txertatuz osagaiak bere alde on eta txar guztiekin.

Orain arte azaldu ditugun egoera guztietan, osagaien iraunkortasuna, objektu-taulen arteko mapaketa gauzatuta egiten zen. Hala, iraunkortasunaren trataerak datu-bihurketa ugari ditu biltegiratutako informazioa osagai erara antolatzeko. Azken urteetan, teknologia berriak garatu dira datu-bihurketa hori ekiditeko objektuetara zuzendutako iraunkortasuna erabilita. Laugarren atalburuaren amaieran, db4o objektuetara zuzendutako datu-basearen ezaugarriak aurkeztuko dira iraunkortasun mota horren nondik norakoak azaleratzeko.

Sistema informatiko baten osagaiak harremanetan jarri nahi direnean beste sistema informatiko baten osagaiekin, ez da arazorik egongo, baldin bi sistemek erabiltzen dituzten plataformak eta teknologiak bateragarriak badira; zoritxarrez, sarritan, informatikaren ekosistema poliedrikoak bateragarritasun hori hautsi egiten du, orduan zer egin? Zerbitzuetara zuzendutako arkitektura erabili arazoa konpontzeko. Bosgarren atalburuan, arkitektura horren ezaugarriak eta arkitektura hori inplementatzeko Javaren teknologiak eskaintzen dituen baliabideak azalduko dira. Teknologiak XML dokumentuak eta generizitatea erabiltzen ditu iturburu nagusi gisa, sistemen arteko bateragarritasuna lortzeko.

Zerbitzuen antolaketa generikoa espezifikazio estandarrak erabilita egiten da. Hala, zerbitzu-bezeroak edota zerbitzu-hornitzaileak badaki zeintzuk diren

3. http://en.wikipedia.org/wiki/Plain_Old_Java_Object

aplikatzen diren kudeaketa-arauak zerbitzuak sortzeko, biltegiratzeko eta eskuratzeko.

Hiru mailako arkitekturak inplementatu nahi direnean aplikazio informatikoetan, bi konplexutasun-geruza tratatu behar ditu informatikariak: mekanizatu nahi den domeinuaren konplexutasuna eta erabiliko den teknologiaren konplexutasuna. Konplexutasun bien maila handia denean, teknologia konplexuak erabiltzen dira arkitektura inplementatzeko, hala eskatzen baitu domeinuaren konplexutasunak. Harritzekoa da, baina, domeinuaren konplexutasun hori txikia denean ere, eskakizun teknologikoak maila altuan kokatzea. Gainera maila horretan, gauzatuko diren eragiketak, behin eta berriz errepikatzen badira aplikazio bakoitzarentzat, arazoa larria bihurtzen da. Testuinguru horretan, Ruby on Rails eta antzeko tresna aurreratuek beren onera ekarriko dituzte teknologia-arazoak, sinplifikatuz eta automatizatuz diseinuaren eta inplementazioaren lan mekaniko astuna. Seigarren atalburuan, adibide bat diseinatu eta inplementatuko da aipatutako testuinguruan. Rails tresnaren baliabideak erabilita, urratsez urrats, eraikiko da aplikazioa.

Lehenengo liburuan, ibilaldi luze honetan lagundu didaten eta euskarri izan ditudan hurbilekoak gogoratu, agurtu eta eskertu nituen. Jakina, beti ahazten da norbait, bigarren honek balio beza eskerrak eta barkamena eskatzeko ahaztutakoei. Besteak beste, UEUko Nekaneri eta Anderri argitaratze unean irudiak eta testua txukun antolatzeagatik eta orrazteagatik. Orrazketarekin jarraituz, eskerrak baita EHUko Iñigori eta Amaiari. Ez ahaztu baina, irakurle, ile-apaindegiko arduradun nagusia[4] ni izan naizela.

Amaitzeko, egindakoa hitz gutxitan laburbildu beharko banu, Fito Cabrales zakur zaharrak dioenarekin gelditu ko nintzateke: etorkizunean edota hor nonbait aurkitu nahi banauzu irakurle, amaieratik hurbil izango nauzu dena amaieratik hurbil hasten delako.

Donostian, 2007ko azaroa

4. If I have seen further it is by standing on the shoulders of giants.

I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.

Isaac Newton

II Atala

Garapen monolitikotik hiru mailako arkitekturara bezero/zerbitzariak bisitatuz

1. Lehen hurbilketa: garapen monolitikoa (AWT-SWT/Java/JDBC)

1.1. ERABILTZAILEAREN INTERFAZE GRAFIKOAREN ERAIKUNTZA: AWT-SWING

1.1.1. Sarrera

Aplikazio informatikoen sarrera-irteera estandarrak teklatua eta pantaila izan ohi dira eta hala dirudi jarraituko dutela izaten datozen urteetan. Informatikaren hastapenean eta ondorengo urteetan, aplikazio bat egikaritzen zenean, programak, pantailan ateratzen zituen mezuak eta emaitzak, karaktere sorta batean lerroz lerro sailkatuta.

Erabiltzaileak gainera zerbait sartu behar zuenean programan karaktere-lerro bat tekleatzen zuen, eta `return` edo `intro` tekla zapalduta egikaritzen jarraitzen zuen programak. Windows-en[5] arrakastak iraultza ekarri zuen informatikara, erabiltzailea ez zen eroso sentitzen aurreko belaunaldiko interfazeekin eta beste zerbait eskatzen zuen. Sistema eragilea erosoa, atsegina eta ikonoz betea agertzen zitzaion egunero ordenagailua pizten zuenean eta, aldiz, programa bat egikaritzen zuenean, dena bihurtzen zen beltza, aspergarria, sekuentziala eta determinista. Hortik aurrera, norbaitek aplikazio bat, software-tresna aurreratu bat, programazio-lengoaia bat edo sistema eragile bat diseinatu nahi bazuen, itxura grafikoa oso kontuan hartzera behartuta zegoen.

Egun, zaila da ikustea aplikazio komertzial bat interfaze grafikorik gabe. Botoiak, taulak, grafikoak, menuak, markoak eta panelak barra-barra agertzen dira programa egikaritu ahala. Ondo diseinatuta badaude elementu grafiko horiek guztiak erabiltzailearen mesederako dira, erroreak ekiditen baitira eta sarrera-irteerak erraztu.

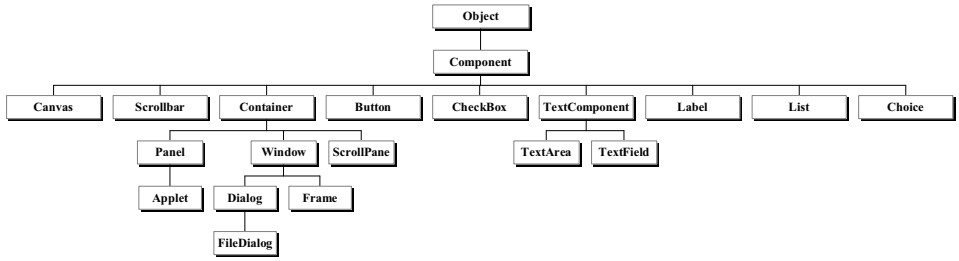
---

5. Benetako meritua Apple Computers eta MacIntosh ordenagailuena litzateke. Idazten duenak oraindik gogoan ditu, Microtxofen, barka Microsoften lehen ahalegin antzuak Windows sistema eragilea MacIntoshizatzeko; emaitzak ikusita ezin du ezkutatu irribarre maltzurra. Ez alferrik, garai hartan behin eta berriz errepikatzen zen Windows-95 Apple-85 zela besterik ez.

Softwarea eraikitzeko joera berri honek termodinamikaren lege zahar hura dakarkigu gogora: erabiltzailearen erosotasuna informatikariaren kaltean egiten dela. Informatikariak jakin badaki, aplikazio baten sarrera-irteera programatu behar duenean askoz ere erosoagoa dela `print` edo `read` arruntak programatzea, interfaze grafiko sofistikatuak diseinatzea baino. Bigarrenean, askoz ere gehiago izan behar du kontuan erabiltzailea, arreta jarri behar baitu jakiteko zein erabiltzaile motak erabiliko duen aplikazioa.

Aipatutako interfaze grafikoak diseinatzeko eta programatzeko bi hitzen inguruan kokatzen da Javaren azpiegitura: AWT (Abstract Windowing Toolkit) eta SWING. Bai bata eta bai bestea, Javaren pakete estandarrak dira, inportatu ondoren zuzenean erabil daitezkeenak. Pakete horietan, hainbat eta hainbat osagai eta edukiontzi ditu prest programatzaileak, hartu eta interfaze grafikoak eraikitzeko. Programazio-lan hori zeharo erraztuko da, programatzaileak software-tresna aurreratua[6] badu softwarea garatzeko.

Saltsa handiagotan sartu baino lehen azter dezagun AWT paketearen arkitektura orokorra. AWTren klase-hierarkia **1.1. irudia**n azaltzen da. `Component` klase nagusiaren azpian hainbat osagai[7] agertzen zaizkigu programatzaileak interfaze grafikoan koka ditzan, hala behar izanez gero. Erabilienak botoiak, etiketak, zerrendak (`List`), aukerak (`Choice`) eta testu-osagaiak (`TextComponent`) dira. Osagai horiek osagai zehatzak dira, zuzenean koka daitezkeenak edukiontzian (`Container`).



1.1. irudia. AWTren klase-hierarkia.

Klase-hierarkiak Component diseinu-patroia erabiltzen du osagaiak egituratzeko, alegia, interfaze grafiko bat diseinatzen dugunean osagaiz osatutako zuhaitz bat eraikitzen dugu. Zuhaitz horren hostoak osagai zehatzak izango dira, eta nodoak, edukiontziz hornitutako tokiak. **1.1. irudia**n ikusten den legez, edukiontzi bakoitza osagaiz osatuta egon daiteke, eta osagai horiek osagai zehatzak edo berriro edukiontzi berriak izan daitezke. Diseinu honekin beti dakigu non dagoen kokatua

6. Egun, nahitaezkoa da horrelako tresna bat erabiltzea softwarearen konplexutasuna ertaina edo handia denean. Programatzaile gutxi ikusten da konpiladorearekin gordin-gordinean lanean.

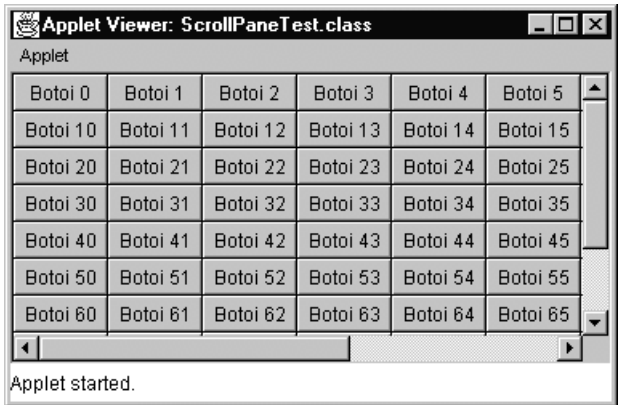
7. Interfaze grafikoen testuinguruan osagai bat `Component` klasetik eratortzen den klasea izango da. Beraz, ez da nahasi behar software-osagaiarekin, horrek beste esanahi bat baitu.

osagai bakoitza klase-hierarkian eta programatzailearentzat errazagoa da programatzea osagai bakoitzaren portaera, eta, aldi berean, Javaren interpretearentzat erosoagoa da erantzun egokia aukeratzea, gertaera bat sortzen denean interfaze grafikoan.

1.1. irudiak argi erakutsi digu osagai zehatzak edukiontzietan kokatzen direla eta beti oinarrizko edukiontzi bat eraiki behar dugula osagaiak bertan kokatzen joateko. Has gaitezen, beraz, edukiontzi horiek zer diren azaltzen. Javak hiru edukiontzi mota ditu: `Panel`, `Window` eta `ScrollPane`. *Azkenekoak lehenengoak izango direla* dioen atsotitzari kasu eginez, aurkez dezagun lehen edukiontzia bere sinpletasunari erreparaturik eta ez adierazitako ordenari.

`ScrollPane` edukiontziak korritze-barraz hornitzen du bere edukiontzi aita[8] garatzen ari garen interfaze grafikoan. Horrela, edukiontzi nagusiaren barna mugi gaitezke ikusten den area mugituz ezker-eskuin. Beharren arabera, korritze-barrak luzeran eta zabaleran koka daitezke. `ScollPane` klaseak ez du gertaera-kudeatzailerik[9] behar, alegia, ez du kontrolatuta egon beharrik; korritze-barra mugituz, luzeran edo zabaleran, `ScrollPane` bera eguneratzen da automatikoki, inoren laguntzarik gabe. Ez du gertaera bat sortu behar, ez eta gertaera hori tratatzen duen metodorik programatu behar. Hona hemen adibide bat:

```
ScrollPane pane = new ScrollPane();
Panel PanelNagusia = new Panel();
PanelNagusia.setLayout(new GridLayout(10, 10));
for(int i=0; i<100; i++) {
  PanelNagusia.add(new Button("Botoi " + i));
}
pane.add(PanelNagusia);
add(pane, BorderLayout.CENTER);
```



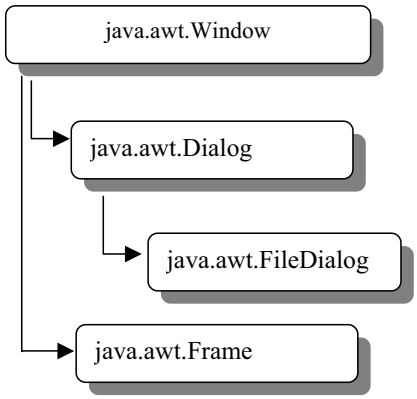
1.2. irudia. Adibidearen irteera grafikoa.

8. Edukiontzi aita hori, edukiontzi nagusia izango da eta `ScrollPane` bere edukiontzi mendekoa.

9. Gertaeren kudeaketa, geroxeago aztertuko dugu. Oraingoz, bakarrik jakin behar dugu osagaiaren egoera aldatzen denean gertaera bat sortzen dela eta ondoren, gertaera horri atxikitako metodoa egikaritzen dela.

Iturburu-kodeak 10 x 10 motako matrize birtuala (**1.2. irudia**) sortzen du `Panel` motako objektu batean. Gero, `Button` motako objektuz betetzen du matrize hori eta, azkenik, `ScrollPane` motako objektuari itsasten zaio aipatutako `Panel` objektu hori.

Javaren edukiontzi generikoak `Panel` motakoak izan ohi dira gehienetan. Oso egokiak dira osagai zehatzak multzokatzeko beste edukiontzi handiago batean. Panelak berak ere multzokatu daitezke beste panel handiago batean txertatzeko. Hala, panel batean beste azpipanel batzuk eduki ditzakegu, bakoitzak funtzio bat betetzen. Aurreko adibidean, `Panel` motako osagai bat eraikitzen da `ScrollPane` motako beste osagai batean txertatzeko. Hori bai, lehenengo, `Panel` motako objektua ehun botoiz hornitzen dugu.



1.3. irudia. `Window` osagaiaren klase-hierarkia.

Aplikazio informatiko baten interfaze grafikoaren hasierako edukiontzia eta egikaritu ahala agertuko diren interfazeak egituratzeko, `Window` motako edukiontzi orokorrak erabiltzen dira. **1.3. irudia**n ikus daitekeenez, `Window` klase horretatik beste hiru azpiklase eratortzen dira: `Frame` eta `Dialog` zuzen-zuzenean, eta `FileDialog` zeharka. Hiru azpiklaseak edukiontzi nagusiak izaten dira interfaze grafikoetan eta bertan kokatzen dira osagai zehatzak, edukiak itsatsiz hasierako *arbel* horiei. `Frame` bat leiho huts bat da eta sortu ondoren zain gelditzen da osagaiak bertan kokatzeko. Bistaratu ondoren, erraza izaten da identifikatzen: eraiki ondoren, goiburuan markoaren izena agertzen da, maximizatu, minimizatu eta ezabatu ikonoekin batera.

Hain ezagunak diren menu motako osagaiak eraiki nahi baditugu aplikazioaren interfaze grafikoan, nahitaez, `Frame` motako edukiontzian txertatu behar ditugu arazorik ez izateko aplikazioa egikaritzean.

`Frame` motako edukiontzi nagusia izanda, zertarako behar dugu `Dialog` motako beste bat? Leihoen arteko harremanak ondo egituratzeko. `Frame` motako

objektu bat sortzen dugunean metodo eraikitzaileari[10] gehienez `String` motako izenburu bat pasa diezaiokegu. Markoa eraiki ondoren eragiketak gauzatuko ditu programak marko horretan eta posible da une batean aplikazioak beste leiho bat eraiki nahi izatea. Egoera horretan, berriro `Frame` motako edukiontzi bat eraikitzen badugu, bigarren marko hori independentea izango da lehenengoarekiko. Ez dago marko bien artean inongo erlaziorik edo mendekotasunik.

Egoera bertsuan, bigarren leihoa eraikitzeko, `Dialog` motako objektua aukeratzen badugu, gauzak zeharo aldatuko dira. `Dialog` motako objektua eraiki nahi badugu, beti adierazi behar dugu zein den objektu horren edukiontzi aita[11] eta, gainera, metodo eraikitzaileak (**1.4. taula**) aukera ematen digu adierazteko ezin dela beste leiho batera joan, `Dialog` motako objektuak kontrola hartzen duenetik eta hura aktibaturik dagoen bitartean. Ezaugarri hori finkatzeko, metodoak daukan boolear parametroa `true` eran hasieratuko da.

Metodo eraikitzailea	Deskripzioa
Dialog (Dialog <aita>)	`Dialog` motako objektu eraiki berriak <aita> `Dialog` motako objektua izango du ugazaba.
Dialog (Dialog <aita>, String <goiburua>)	`Dialog` motako objektu eraiki berriak <aita> `Dialog` motako objektua izango du ugazaba, eta <goiburua> aldagaiaren balioa, goiburuan agertuko da.
Dialog (Dialog <aita>, String <goiburua>, boolean <modala>)	`Dialog` motako objektu eraiki berriak <aita> `Dialog` motako objektua izango du ugazaba, eta <goiburua> aldagaiaren balioa, goiburuan agertuko da. Aldagai boolearraren balioak adieraziko du elkarrizketa modala ala ez-modala izango den.
Dialog (Frame <aita>)	`Dialog` motako objektu eraiki berriak <aita> `Frame` motako objektua izango du ugazaba.
Dialog (Frame <aita>, String <goiburua>)	`Dialog` motako objektu eraiki berriak <aita> `Frame` motako objektua izango du ugazaba, eta <goiburua> aldagaiaren balioa, goiburuan agertuko da.
Dialog (Frame <aita>, boolean <modala>)	`Dialog` motako objektu eraiki berriak <aita> `Frame` motako objektua izango du ugazaba, eta aldagai boolearraren balioak adieraziko du elkarrizketa modala ala ez-modala izango den.

10. Adibidez: markoa `Frame(String <izenburua>)` metodo eraikitzaileaz eraikitzen dugu, `Frame markoa = new Frame("Gure marko berria");`

11. Edukiontzi aita, nahitaez, `Frame` motako objektua edo `Dialog` motako objektua izango da.

Dialog (Frame <aita>, String <goiburua>, boolean <modala>)	Dialog motako objektu eraiki berriak <aita> Frame motako objektua izango du ugazaba, eta <goiburua> aldagaiaren balioa, goiburuan agertuko da. Aldagai boolearraren balioak adieraziko du elkarrizketa modala ala ez-modala izango den.

1.4. taula. `Dialog` osagaiaren metodo eraikitzaileak.

Edukiontzien (`Container`) klase-hierarkia bukatzeko eta **1.1. irudia**n islatzen den bezala, `FileDialog` edukiontzia gelditzen zaigu aztertzeko. Edukiontzi horrekin erabiltzaileak bere fitxategi-sistema azter lezake eta adierazi zein eragiketa mota (`LOAD` edo `SAVE`) gauzatuko duen. `Dialog` klase baten antzera, leiho modalak eraikitzen ditu `FileDialog` klaseak. Hona hemen aplikazio txiki bat esandakoa sakontzeko:

```
import java.awt.*;

class Elkarrizketa extends Frame
{  FileDialog sistema;
   Elkarrizketa()
   {setSize(400,400);}

   void ireki()
   {  sistema=new FileDialog(this,"Elkarrizketa LOAD eran");
      sistema.show();
      setTitle("Fitxategia irekitzen:
"+sistema.getDirectory()+sistema.getFile());
   }
   void gorde()
   {  sistema=new FileDialog(this,"Elkarrizketa SAVE
eran",FileDialog.SAVE);
      sistema.show();
      setTitle("Fitxategia gordetzen:
"+sistema.getDirectory()+sistema.getFile());
   }
   public static void main(String args[])
   {  Elkarrizketa elkarrizketa =new Elkarrizketa();
      elkarrizketa.show();
      elkarrizketa.ireki();
      elkarrizketa.gorde();

         }   }
```

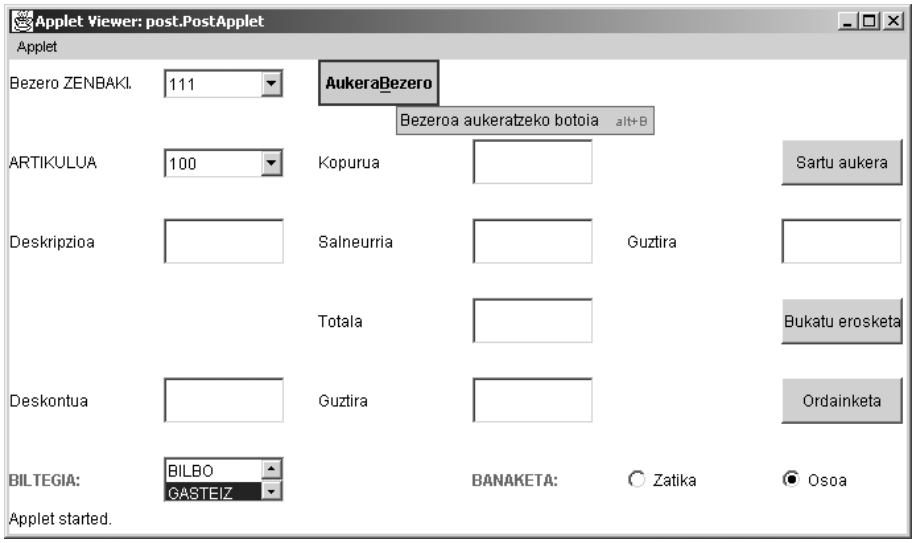
Aplikazio horrek marko bat eraikitzen du eta, ondoren, `ireki()` eta `gorde()` metodoak egikaritzen ditu. Lehenengo metodoak (`ireki()`) `FileDialog` motako objektua eraikitzen du eta metodo eraikitzailean ezer adierazten ez denez, `LOAD` erakoa izango da, horixe baita lehenetsiz hartzen dena. Hori egin ondoren, `gorde()` metodoak `FileDialog` motako beste objektu bat eraikitzen du. Oraingo honetan,

objektuaren era `SAVE` izango da, hala adierazten baitiogu metodo eraikitzaileari parametroen zerrendan.

1.1.2. Interfaze grafikoaren osagaiak (Component)

Orain arte ikusitakoarekin non kokatuko dituen osagaiak (`Component`) edukiontzietan erabaki du aplikazioaren interfaze grafikoaren diseinatzaileak. Erabaki hori hartu ondoren, osagaiak kokatzen hasiko da diseinatzailea diseinatutako eduki bakoitzean. Atal honetan Javak eskaintzen dituen osagaiak aztertuko ditugu, azalduz zeintzuk diren eta nola eraiki eta koka daitezkeen edukiontzi batean. Hori egiteko, lehenengo eta behin, adibide bat deskribatuko dugu, eta, ondoren, aplikazioak edukiko duen interfaze grafikoa diseinatuko dugu. Geroxeago ikusiko dugun bezala, interfaze grafikoa propio izango da diseinatua Javak eskaintzen dituen osagai gehienak urratsez urrats azaltzeko adibidean. Has gaitezen, bada, adibidea deskribatzen: aplikazio informatiko bat eraiki nahi dugu bezeroaren eskaerak kudeatzeko. Lana errazteko aplikazioaren erabiltzaileari, interfaze grafiko bat diseinatuko dugu eskaeraren datuak sartzeko. Interfazeak erosoa izan behar du erabiltzailearentzat eta eraginkorra sistemarentzat. Horretarako, erabiltzaileari laguntza batzuk eskainiko dizkiogu datuen sarrera errazteko eta akatsak saihesteko. Horrela, erabiltzaileak ezin izango ditu datu okerrak sartu sisteman, hark ez baitizkio onartuko.

Ezaugarri horiek guztiak kontuan izanda honako, behin behineko, interfaze grafiko hau diseinatu dugu:



1.5. irudia. Adibidearen interfaze grafikoa.

Interfaze grafikoak kontzeptualki sei lerro eta sei zutabe ditu eta 6x6 motako matrize horretan hainbat osagai kokatu ditugu. Jarraian, banan-banan[12] azalduko ditugu.

Lehen lerroan, lau osagai kokatu ditugu: etiketa (`Label`) bat, aukera (`Choice`) bat eta bi botoi (`Button`). Etiketa osagaiak jarraian datorren osagaiaren nondik norakoak argitzen laguntzen du. Bere metodo eraikitzailea (**1.6. taula**), osagai gehienena bezala, gainkargatua dago:

Metodo eraikitzailea	Deskripzioa
`Label( )`	Etiketa bat eraikitzen da hasierako baliorik gabe.
`Label(String etiketa)`	Etiketa bat eraikitzen da `<etiketa>` balioaz hasieratuta.
`Label(String etiketa, int lerrokatzeEra)`	Etiketa bat eraikitzen da lerrokatuta eta `<etiketa>` balioaz hasieratuta.

1.6. taula. `Label` osagaiaren metodo eraikitzaileak.

Edukiontzian etiketak hiru eratara lerrokatu daitezke:

- `Label.LEFT`: Ezkerrera lerrokatuta.
- `Label.CENTER`: Erdira lerrokatuta.
- `Label.RIGHT`: Eskuinera lerrokatuta.

Etiketak, esan dugun bezala, ondorengo osagaia aurkezten digu, gure kasuan aukera (`Choice`) motako osagai bat. Osagai horrek aukeran ditugun bezero-zenbakiak erakusten dizkigu. Konturatu osagai hori oso egokia dela aukera asko ditugunean eta behar-beharrezkoa ez denean denak begi-bistan edukitzea. Horrela, tokia aurrezten dugu interfaze grafikoan eta ulergarriagoa da sistemaren eta erabiltzailearen arteko elkarrizketa.

Jakina, mota honetako osagaia kudeatzeko ez da nahikoa metodo eraikitzailea egikaritzea. Horren bidez, mota horretako objektu bat eraikitzen dugu baina objektu hori hutsik egongo da, eta lehenago edo beranduago elementuz hornitu beharko dugu:

```
private Choice aukeraBezero = new Choice();
aukeraBezero.addItem("111");
aukeraBezero.addItem("222");
aukeraBezero.addItem("333");
aukeraBezero.addItem("444");
```

`Choice` motako objektua eraiki ondoren, hainbat metodo erabil ditzakegu objektuaren elementuak aukeratzeko, gehitzeko, ezabatzeko eta eskuratzeko:

12. Jakina, osagai mota errepikatzen denean txanda-pasa egingo dugu.

Metodoa	Deskripzioa
`getItem(int posizioa)`	Adierazitako posizioan dagoen elementua eskuratzen da.
`addItem(String balioa)`	Balio berri bat gehitzen da.
`getSelectedItem( )`	Aukeratuta dagoen elementua eskuratzen da.
`getSelectedIndex( )`	Aukeratuta dagoen elementuaren posizioa eskuratzen da.
`select(int posizioa)`	Aukeratu posizio horretan dagoen elementua.
`remove(int posizioa)`	Ezabatu posizio horretan dagoen elementua.
`removeAll( )`	Ezabatu osagaiaren elementu guztiak.

1.7. taula. `Choice` osagaiaren metodoak.

Gogoratu halaber, `Choice` klaseak ez dituela aukera anizkoitzak onartzen, hau da, edozein unetan gehienez aukera bakarra egongo da aktibatuta.

Buka dezagun lehen lerroaren azalpena sarri erabiltzen den osagai bat aurkeztuz: `Button`. Botoi bat interfaze grafikoan itsastea lan erraza da: nahikoa da, `Button` motako aldagai bat erazagutzea eta ondoren metodo eraikitzaileari deituta `Button` objektua eraikitzea. Objektu horren erreferentzia aldagaiari esleitzen diogu eta kito. Metodo eraikitzailea gainkargatua dago: `Button()` eta `Button(String <etiketa>)`. Lehenengoak botoia eraiki eta ez dio inongo identifikaziorik ipintzen; bigarrenak, berriz, parametroan jarri dugun `String`-a itsatsiko dio botoiari egikaritzean.

Hemen atsedenaldi txiki bat hartuko dugu eta `Button` klasearen anaia bat aurkeztuko dugu: `JButton`. `JButton` klasea aitzakia gisa dakargu hona, beste ahaide garrantzitsuago bat aurkezteko: SWING.

Urteak aurrera joan ahala lengoaiaren heldutasunak erronka berriak ekarri zizkien Javaren diseinatzaileei. Bertsio berrietan lengoaiaren zenbait ezaugarri hobetu egin behar izan zituzten batzuetan, lengoaiaren erabiltzaileek hala eskatuta eta, besteetan, diseinatzaileek detektatutako arlo pattalak hobetzeko asmoz. Batzuetan, ezaugarri berriak hasieratik diseinatu eta programatu behar zituzten, beste batzuetan, berrerabilera landuz, aurretik diseinatutako klaseak erabiltzen ziren oinarri gisa ezaugarri berriak lengoaian txertatzeko. SWING eraikitzeko orduan bigarren bide hori aukeratu zuten.

Erabiltzailearen interfaze grafikoak eraikitzeko, ez zuen inolako zentzurik aurretik diseinatutako AWT pakete osoa alde batera uzteak eta berriro hutsetik hasteak. Nahitaez, erruz erabili behar zuen SWINGek AWT paketea eta, gainera, programatzaileari aukera eman behar zitzaion, nahi bazuen, AWT paketea bere

jatorrizko eran erabiltzeko, argi baitago AWT lotuago dagoela lengoaiaren hasierako diseinuari SWINGi baino. Galdera, orduan, berehala sortzen da: zertarako pakete berri bat sortu zaharra ondo diseinatuta badago? Lengoaiaren diseinatzaileak ere antzeko galderaren aurrean aurkitu ziren eta erabaki zuten SWINGek AWTren klase-hierarkia erruz erabiliko zuela. Horretarako, bi bide aukeratu zituzten: berrikuntza eta berrerabilera.

Berrikuntzaren aldetik, osagai (`ComboBox`, `Slider`, `JTree`, `JTable`...) gehiago diseinatu ziren interfazearen diseinua aberasteko. Bestalde, erabiltzaileak nahi zuen erara egokitu zezakeen aplikazioaren interfaze osoaren itxura. Horrela, erabiltzailearen interfazea kudeatzen duen `UIManager` klasea erabilita, interfaze osoaren itxura alda daiteke programaren bidez eta interfaze grafiko osoari Motif, Windows edo beste itxura era bat eman. Hona hemen adibide bat:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// Interfazearen itxura aldatzeko paketeak
import com.sun.java.swing.plaf.windows.*;
import com.sun.java.swing.plaf.motif.*;
import javax.swing.plaf.metal.*;

public class NireAppleta extends JApplet implements ActionListener
{
       public void init() {
              Container contentPane = getContentPane();

              defaultLookAndFeel = UIManager.getLookAndFeel();

              botoiLehenetsia = new JButton("Lehenetsia");
              metalBotoia = new JButton("Metal");
              motifBotoia = new JButton("Motif");
              windowsBotoia = new JButton("Windows");

              // Osagaiak eraikitzen
              JPanel osagaiak = new JPanel();
    JCheckBox egiaztatzeKutxa = new JCheckBox("EgiaztatzeKutxa");
              osagaiak.add(egiaztatzeKutxa);
              JTextField textField = new JTextField("TestuEremua");

              osagaiak.add(textField);
              // Panel bat sortu botoiekin
              JPanel botoiak = new JPanel();
              botoiak.add(botoiLehenetsia);
              botoiak.add(metalBotoia);
              botoiak.add(motifBotoia);
              botoiak.add(windowsBotoia);
```

```
            // Panelak gehitu edukiontziari
            contentPane.add("North", osagaiak);
            contentPane.add("South", botoiak);

            // Osagaien entzuleak antolatzen
            botoiLehenetsia.addActionListener(this);
            metalBotoia.addActionListener(this);
            motifBotoia.addActionListener(this);
            windowsBotoia.addActionListener(this);
      }

      public void actionPerformed(ActionEvent ev) {
            try {
                  if (ev.getSource() == motifBotoia)
               UIManager.setLookAndFeel(new MotifLookAndFeel());
                  else if (ev.getSource() == windowsBotoia)
              UIManager.setLookAndFeel(new WindowsLookAndFeel());
                  else if (ev.getSource() == botoiLehenetsia)

UIManager.setLookAndFeel(defaultLookAndFeel);
                  else
               UIManager.setLookAndFeel(new MetalLookAndFeel());
                  SwingUtilities.updateComponentTreeUI(this);
                  validate();
            } catch (UnsupportedLookAndFeelException e) {

System.out.println("UnsupportedLookAndFeelException");
            }
      }

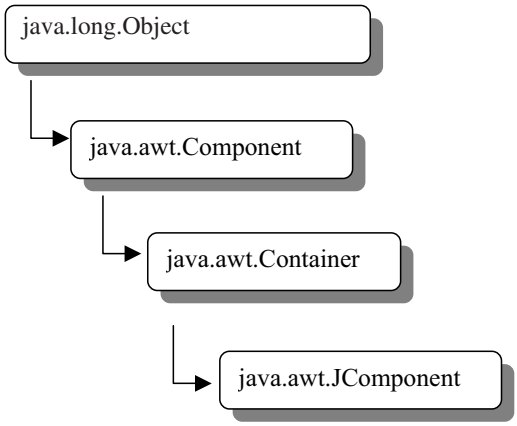
      LookAndFeel defaultLookAndFeel;
  JButton botoiLehenetsia, metalBotoia, motifBotoia, windowsBotoia;
}
```

Berrerabileraren aldetik, lengoaiaren diseinatzaileek SWING osagai berrien gertaera-trataera eta diseinu-kudeatzaileen portaera tratatzeko, AWT ingurunean erabiltzen zen berbera aukeratu zuten pakete berrirako. Horrela, programatzaileak ez zuen gauza berri askorik ikasi behar pakete berriko osagaiak programatzeko.

Berrikuntzetan sartuta, berrerabilerarekin eta berrikuntzarekin zerikusia duen arlo bat nabarmen hobetu zuten pakete berrian. SWING paketearen klase-hierarkian `JComponent`[13] klasea SWING osagaien superklase nagusia da. Baina superklase nagusi hori ez da AWTren `Component` klasea bezala Javaren `Object` superklase nagusitik eratortzen zuzenean, `Container` azpiklasetik baizik, **1.8. irudia**n ikusten den legez. Beraz, SWINGen osagaiek edukiontzi gisa funtziona

13. SWINGen `JComponent` klaseak eta AWTren `Component` klaseak betebehar bera dute, jakina, bakoitzak bere osagaiekin.

dezakete, alegia, SWING osagaietan beste osagai batzuk txertatzeko bidea irekita gelditzen da, osagai (`JComponent`) berri horiek edukiontzi (`Container` ← `JComponent`) batetik eratortzen baititugu.



1.8. irudia. `JComponent` osagaiaren klase-hierarkia.

SWING paketearen ezaugarri nagusiak aurkeztu ondoren itzul gaitezen gure bi anaiengana: `Button` eta `JButton`. Bata, AWTren osagaia da, eta bestea, SWINGena. `Button` klasea `Component` klasetik eratortzen da, eta `JButton`, `JComponent`-etik[14]. Botoi baten eta bestearen portaerak argitzeko erabiltzailearen interfaze grafikoan, itzul gaitezen atal honen hasieran aurkeztu dugun adibidera (**1.5. irudia**).

Adibidearen lehen lerroan bi botoi diseinatu ditugu. Lehenengo botoiaren aurretik jarri dugun aukera (`Choice`) osagaian erabiltzaileak bezero-kode bat aukeratuko du. Aukera egin ondoren, *`AukeraBezero`* izenburuarekin identifikatu dugun botoia zapaltzen badugu, sortuko dugun eskaeraren bezeroa zein izango den adierazi nahi diogu sistemari. Eraiki dezagun bada, botoi hori gure aplikazioan:

```
private JButton botoiBezero = new JButton("AukeraBezero");
```

Interfaze grafikoaren lehen lerroarekin bukatzeko eraiki dezagun, halaber, bigarren botoia; *`LAGUNTZA`* identifikazioa daukan botoia, alegia:

```
private Button botoiLaguntza = new Button("LAGUNTZA");
```

`J` letra larriaren erabilerak zeharo aldatuko du botoi bien portaera. Hasierako `J` letra larri[15] horrekin lehenengo botoia SWING paketearen osagaia dela adierazten dugu, horrek suposatzen duen guztiarekin. Alegia, `JButton` osagaia

14. Gogoratu `JComponent`, AWTren `Container` klasetik eratortzen dela.

15. Hasierako `J` letra larri hori erabilita, programatzaileak berehala identifikatzen ditu SWING-en osagaiak iturburuan. SWINGen: `JLabel`, `JButton`, `JList`, `JtextField` eta `JtextArea`. Aldiz, AWTn: `Label`, `Button`, `List`, `TextField` eta `TextArea`.

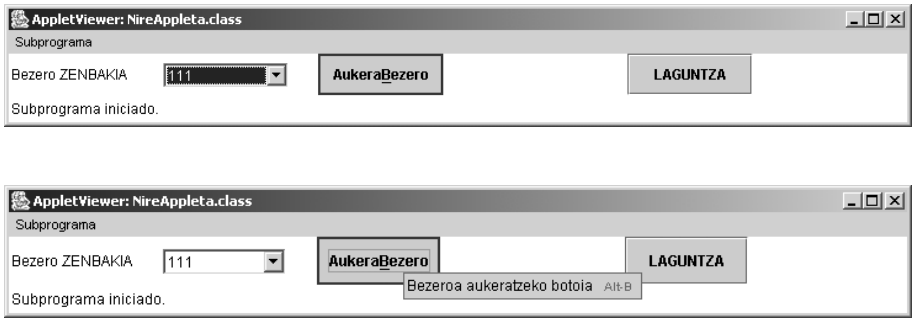
`JComponent`-etik eratortzen dugu eta, hori eginda, osagaiak edukiontzi baten portaera izango du `JComponent`, `Container` klasetik eratortzen baita.

Horrela, `botoiBezero` aldagaiak erreferentziatzen duen `JButton` motako objektua edukiontzi bat denez, hainbat eragiketa (**1.9. taula**) egin ditzakegu `Button` motako objektu batek onartzen ez dituenak:

Lerroa	Metodoa
1	`botoiBezero.setToolTipText("Bezeroa aukeratzeko botoia");`
2	`botoiBezero.setMnemonic('B');`
3	`botoiBezero.setBorder(new LineBorder(Color.red,2));`

1.9. taula. `JButton` osagaiaren portaera programatzen.

Hiru lerro horiekin portaera berria itsatsi diogu `JButton` motako objektuari. `ToolTipText()` metodoarekin erabiltzaileak kurtsorea botoiaren gainean jartzen badu bat-batean laguntza txiki bat azaleratuko zaio testu eran pantailan eta segundo batzuetan ikusgai egongo da. Laguntza txiki hori osatu nahi badugu eta adierazi, halaber, zer zapaldu behar dugun botoiaren aukera azkartzeko orduan, `setMnemonic()` metodoa programatu behar dugu. Gure kasuan, `<Alt-B>` zapaldu behar dugula adierazten ari gara bigarren lerroarekin. Bukatzeko, botoiaren itxura aldatu nahi badugu, ertzetan kolore gorriz hornitutako lerro polit batzuk marraztuz, `setBorder()` metodoa erabiliko dugu, hirugarren lerroan adierazten den modura. Aplikazioa egikaritu ondoren, pantailan agertuko litzatekeena jasotzen du **1.10. irudia**k.



1.10. irudia. `JButton` osagaiaren portaera interfaze grafikoan.

Itzul gaitezen aztertzen ari garen adibidera. Adibidearen (**1.5. irudia**) bigarren, hirugarren, laugarren eta bosgarren lerroetan erabiltzen ditugun osagai motak lehen lerrokoak direnez, ez dugu denborarik galduko berriz azaltzen osagaien ezaugarriak. Beraz, azken lerrora, seigarrenera, joango gara zuzen-zuzenean. Lerro horretan, bi osagai mota berri aurkitzen ditugu: zerrenda (`List`) bat eta bi egiaztapen-kutxatila (`CheckBox`). Zerrenda-objektua eraikitzeak eta kudeatzeak ez dio

arazo larririk sortu behar aukera (`Choice`) motako osagaia eraikitzen eta kudeatzen dakien programatzaileari, antzekotasunak handiak baitira, portaera eta egitura aldetik. Biek zerrendak kudeatzen dituzte, eta bien gertaera-trataera era berean gauzatzen da. Egiaztapen-kutxatila motako objektu bat eraiki nahi badugu, lengoaiak lau metodo eraikitzaile eskaintzen ditu:

Metodo eraikitzailea	Deskripzioa
`CheckBox( )`	Egiaztapen-kutxatila eraikitzen da eta egoerari `false` balioa esleitzen zaio.
`CheckBox(String etiketa)`	Egiaztapen-kutxatila etiketarekin eraikitzen da eta egoerari `false` balioa esleitzen zaio.
`CheckBox(String etiketa, boolean egoera)`	Egiaztapen-kutxatila etiketarekin eta egoera-balio batekin eraikitzen da.
`CheckBox (String etiketa, CheckBoxGroup talde, boolean egoera)`	Egiaztapen-kutxatila etiketarekin eta egoera-balio batekin eraikitzen da eta egiaztapen-kutxatila talde batekin asoziatzen da.

1.11. irudia. `CheckBox` osagaiaren metodo eraikitzaileak.

Lau metodo horietatik laugarrena aukeratu dezakegu gure beharretarako. Gogoratu, kutxatila horietan adierazten dela nola gauzatu nahi den bezeroaren banaketa. Erabiltzaileak ez badu ezer adierazten, eskaeraren banaketa, artikulu guztiak prest ditugunean, egingo da. Horretarako, `Osoa` identifikadorea daukan egiaztapen-kutxatila, aukeratuta (`true`) ezaugarriarekin, eraikiko dugu hasiera-hasieratik:

```
       private CheckboxGroup banaketa= new CheckboxGroup();
       private Checkbox zati = new
Checkbox("Zatika",banaketa,false);
       private Checkbox osoa = new Checkbox("Osoa",banaketa,true);
```

Konturatu, nola eraiki dugun, `osoa` aldagaiak erreferentziatzen duen egiaztapen-kutxatila motako objektua: identifikadore ("`Osoa`") bat ipini diogu, kutxatila multzokatzeko erabiliko dugun objektuaren[16] erreferentzia pasatzen diogu, eta, bukatzeko, boolear parametroa `true` gisa finkatzen dugu.

1.1.3. Intefaze grafikoen gertaeren kudeaketa

Aplikazioaren interfaze grafikoen osagaietan ekintza bat gauzatzen badu erabiltzaileak, hala nola botoi bat zapaldu, zerrenda bateko elementu bat aukeratu edo leiho bat itxi, gertaera bat sortzen du, zeina aplikazio informatikoak tratatu

16. Egiaztapen-kutxatilak taldekatu nahi baditugu beste osagai berri bat eraiki behar dugu: `CheckBoxGroup` motakoa. Osagai mota hori eraiki ondoren, metodo eraikitzailean gauzatzen da taldekatzea.

behar duen erantzuna emanez interfaze grafikoan gertatu den aldaketari. Javaren 1.2 bertsioak aldaketa bat ekarri zion gertaera-kudeaketari. Baldin eta aplikazioaren programatzaileak aurretik badaki erabiliko duen Java lengoaiaren bertsioa 1.2 edo ondorengoren bat izango dela, ez du zertan jakin nolakoa zen gertaera-kudeaketa zaharra, nahikoa izango zaio jakitea nola dagoen egituratuta gertaera-kudeaketa berria. Guk, gertaeren kudeaketa berri hori azaldu baino lehen, azaletik ez bada ere, gertaera-kudeaketa zaharraren ezaugarri nagusiak azalduko ditugu. Aldi berean, gertaera-kudeaketa zaharraren azterketak erraztuko digu berriaren berrikuntzen eta aldaketen berri ematea.

Gertaera-kudeaketa zaharrean, `Event` klaseak gertaeraren hainbat informazio eraikitzen du, hurrenez hurren nork sortu duen gertaera, noiz sortu den, zein motatakoa den, eta abar. Informazio hori guztia, `handleEvent()` metodoak kudeatzen du. Metodo hori `Component` klasearen metodoa da eta egokitu nahi badugu gure beharretara, birdefinitu egin behar dugu. `handleEvent()` metodoa birdefinitzea erabakitzen badugu, sistemak ulertzen du guk hartu nahi dugula gertaeren kudeaketaren kontrola, eta gure esku uzten du gauzatuko diren ekintzak programatzea.

Orain arte ikusitakoarekin oinarrizko eragiketak gauzatu ditugu gertaerekin eta ez dugu eduki behar izan `Event` klasearekin harreman handirik. Aplikazio errealetan, erabiltzailearen interfaze grafikoak konplexuak direnean, `Event` klasearen aldagaien balioak kontsultatu behar izango ditugu, eta `handleEvent()` metodoa askoz gehiago luzatuko da, interfaze grafikoaren osagaien portaera programatzeko.

Horretarako nahitaezkoa da jakitea nola dagoen egituratuta `Event` klasea aldagaien aldetik:

Aldagaia	Deskripzioa
`Object osagaia`	Gertaeraren osagaia.
`Object arg`	Gertaerarekin lotutako argumentua.
`public long noiz`	Gertaera sortu den unea.
`public int id`	Eman den gertaera mota
`public int x`	Gertaeraren x koordenatua.
`public int y`	Gertaeraren y koordenatua.
`public int tekla`	Teklaren identifikadorea.
`public int aldatzaileak`	Maskaratutako teklen egoera.

1.12. taula. `Event` klasearen aldagaiak.

Gertaeren kudeaketa berriak zaharraren diseinuaren hainbat *akats* zuzendu zituen, eta gertaeren kudeaketa errazten saiatu ziren lengoaiaren diseinatzaileak. Horrela, aplikazioaren kodea eta gertaeren kudeaketaz arduratzen den kodea modularizatuta programatu daitezke kode mota bakoitza bereiziz. Bestalde, aplikazioa egikaritzean, sortzen diren gertaerak ez dira, gertaera-kudeatzaile zaharrean bezala, `Event` klase bakarretik kudeatzen. Kudeaketa berrian, `EventObject` oinarrizko klasea erabilita azpiklaseak (`WindowEvent`, `ItemEvent`, `ActionEvent`, `TextEvent`...) diseinatu ziren gertaera mota guztiak tratatzeko. Horrela, osagai bakoitzak berari egokitutako gertaerak bakarrik izango ditu kontuan, sortzen diren gertaerak tratatzean.

Behe-mailako gertaerak nahiz goi-mailako gertaerak (gertaera semantikoak) tratatzeko, klase berezi batzuk erabiltzen ditu Javak. Klase horiek *entzule* klaseak dira, eta gertaera mota bakoitzeko entzule klase bat egongo da. Horrela, edozein `xxxEvent`-ek `xxxListener` entzule klase bat izango du esleitua, **1.13. taula**k adierazten digu zeintzuk diren osagaien entzuleak. Entzule klaseak interfazeak dira horrek suposatzen duen guztiarekin, alegia, bere metodo guztiak inplementatzeko daudela eta entzule bat programatu behar badu programatzaileak, diseinatu duen osagaiak hala eskatuta, haren metodo guztiak programatu behar ditu nahitaez.

Gertaera	Entzulea	Metodoa	Osagaia
ActionEvent	ActionListener	actionPerformed	Button List MenuItem TextField
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged	ScrollBar
ComponentEvent	ComponentListener	componentHidden componentMoved componentResized componentShown	Component
ContainerEvent	ContainerListener	componentAdded componentRemoved	Container
FocusEvent	FocusListener	focusGained focusLost	Component
ItemEvent	ItemListener	itemStateChanged	CheckBox Choice List
KeyEvent	KeyListener	keyPressed keyReleased keyTyped	Component
MouseEvent	MouseListener	mouseClicked mouseEntered mouseExited mouseReleased mousePressed	Component
MouseEvent	MouseMotionListener	mouseDraggedd mouseMoved	Component
TextEvent	TextListener	textValueChenged	TextComponent
WindowEvent	WindowListener	windowActivated windowClosed windowClosing	Window

1.13. taula. `Event` klaseen entzuleak.

Entzuleak metodo gutxi dituenean, ez da inongo arazoa metodo guztiak programatzea; ez da gauza bera gertatzen metodoak asko direnean. Demagun leiho

(`Window`) bat diseinatu dugula eta bakarrik itxiera (`windowClosing`) kontrolatu nahi dugula, beste funtzio guztiak (`windowActivated`, `windowDeactivated`, `windowOpened`...) ez ditugu kontrolatu nahi. Orain arte esandakoarekin nahitaez inplementatu behar ditugu beste funtzio horiek ere, zeren leihoa kontrolatuko duen entzulea `WindowListener` baita eta interfaze klase horrek guztira zazpi metodo baititu. Baina hemen arazo txiki bat sortzen da, guk metodo (`windowClosing()`) bat bakarrik erabiliko dugu gure aplikazioan baina, `WindowListener` klasea interfazea denez, beste metodoak ere inplementatu beharrean gaude ezinbestez, nahiz eta inplementazioa hutsa izan. Horrela, kodea alferrik luzatu behar dugu, entzuleak programatzea akats-iturri bihurtuz gainera, metodoren bat programatzeke geldi baitaiteke. Hori ekiditeko lengoaiaren diseinatzaileek *egokitzaile* (`Adapter`) klaseak diseinatu zituzten, entzuleen kudeaketa errazteko. Egokitzaile horiek entzuleak inplementatzen dituzte, baina nola liteke hori, ez baitakigu metodo bakoitzak zer egin behar duen? Erantzuna erraza da: metodo bakoitzari inplementazio hutsa itsatsita.

Konturatu egokitzaileek ez dutela ezer konpontzen, entzuleak metodo bakarra duenean. Bestalde, egokitzaileak erabiltzea oso diseinu egokia da, entzuleak kudeatzeko aplikazioan. Adibidez, lengoaiaren diseinatzaileek entzule bati beste metodo bat gaineratzen badiote lengoaiaren etorkizuneko bertsio batean, nahikoa izango da egokitzailea eguneratzea metodo berria itsatsiz. Gure aplikazioa ez da ohartu ere egingo aldaketaz.

1.1.4. Diseinu-administratzaileak

Osagaiak edukiontzietan paratu ahala tokia hartzen hasiko dira edukiontzian, adierazitako neurriak jagonez gero. Une horretan, diseinu-administratzaileak lan egiten hasiko dira eta guk esandako tokian edo lehenetsiz adierazitako tokian kokatuko dituzte osagaiak.

Lengoaiak bost diseinu-administratzaile eskaintzen dizkigu: `FlowLayout`, `BoderLayout`, `GridLayout`, `CardLayout` eta `GridBagLayout`. Diseinu-administratzailea erabiltzeko aplikazioan `setLayout()`[17] metodoa txertatu behar dugu iturburuan, adieraziz, zein diseinu-administratzaile erabili nahi dugun. Hona hemen adibide bat: `setLayout(new BoderLayout())`. Lerro horren bidez, `BoderLayout` motako diseinu-administratzailea erabiliko duela edukiontzian adierazten du programatzaileak.

Jarraian, administratzaile bakoitzaren ezaugarri nagusiak aurkeztuko ditugu:

`FlowLayout` administratzaileak ezkerretik eskuinera lerrokatzen ditu osagaiak edukiontzian. Osagaiak, edukiontziaren eskuin-muturrera heltzen direnean,

17. `setLayout()` metodoa `Container` klasearen metodo bat da:
```
void setLayout(LayoutManager    <kudeatzailea>);
```

ondorengo lerrora jauzi egiten du administratzaileak eta hor kokatzen dira beste osagaiak.

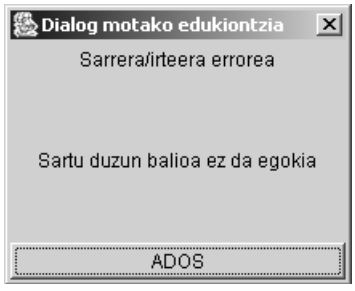
`FlowLayout` administratzaileak, **1.14. taula**n ikusten den legez, hiru metodo eraikitzaile ditu eta osagaiak lerrokatzean hiru eratara egituratzen ditu: `CENTER` (lehenetsitakoa), `RIGHT` eta `LEFT`.

Metodo eraikitzailea	Deskripzioa
`FlowLayout( )`	`FlowLayout` erako diseinu-administratzailea eraikitzen da erdira lerrokatuta.
`FlowLayout(in lerrokatzeEra)`	`FlowLayout` erako diseinu-administratzailea eraikitzen da lerrokatze mota zehatz batekin.
`FlowLayout(in lerrokatzeEra, int tarte_h, int tarte_b)`	`FlowLayout` erako diseinu-administratzailea eraikitzen da lerrokatze mota zehatz batekin. Gainera, halaber, osagaien arteko tarte horizontala eta bertikala zehazten da.

1.14. taula. `FlowLayout` klasearen metodo eraikitzaileak.

`BoderLayout` administratzaileak, bestalde, edukiontzien ertzak erabiltzen ditu osagaiak kokatzeko. Hala, programatzaileak bost gunetan egituratzen ditu osagaiak edukiontzian: iparraldean (`North`), hegoaldean (`South`), ekialdean (`East`), mendebaldean (`West`) eta erdialdean (`Center`).

Administratzaile hau oso egokia da erroreen berri emateko. Hona hemen adibide txiki bat: `Dialog` motako edukiontzi bat eraiki nahi dugu eta, bertan, errore mota, errorearen deskripzio txikia eta botoi bat kokatuko ditugu. Gainera, edukiontzia, modal gisa eraikiko dugu eta erabiltzaileari ez diogu utziko ezer egiten, gure kasuan, *`ADOS`* botoia zapalduta edukiontzitik irten arte. Horrela egiaztatuta geldituko da erabiltzaileak mezua irakurri duela.



1.15. irudia. Adibidearen irteera grafikoa.

```
import java.awt.*;

class Elkarrizketa extends Dialog
{
 Elkarrizketa()
 {
   /* Elkarrizketa(Dialog) edukiontzia modala bezala eraikitzen
dugu*/
    super(new Frame(),"Dialog motako edukiontzia",true);
    setLayout(new BorderLayout(0,40));
    Label errorea=new Label("Sarrera/irteera
errorea",Label.CENTER);
    Label mota=new Label("Sartu duzun balioa ez da
egokia",Label.CENTER);
    Button botoia=new Button("ADOS");

    add("North",errorea);
    add("Center",mota);
    add("South",botoia);
    pack();
    show();
  }

  public static void main(String args[])
  { new Elkarrizketa();
  }  }
```

`GridLayout` administratzaileak laukitxotan egituratzen du edukiontzia, eta laukitxo bakoitzean osagai bat kokatuko du. Beraz, administratzaile hau erabili behar badu edukiontziak, administratzaile mota hau eraikitzean `setLayout()` metodoan adierazi beharko du zenbat lerro eta zenbat zutabe izango dituen edukiontziak eta, nahi izanez gero, halaber, osagai batetik bestera zenbateko distantzia egongo den luzeran eta zabaleran. Distantzia hori pixeletan adierazten da.

Exekuzio-garaian osagaiak aldatu nahi baditugu edukiontzian, `CardLayout` administratzailea erabili behar dugu nahitaez. Administratzaile hori erabilita, edukiontzi batek osagaiak ikuska ditzake edukiontzian beste edukiontzi bat eraiki gabe. Hau da, edukiontzi bera erabiliko da irteera desberdinak sortzeko. Hona hemen adibide bat: `Panel` motako edukiontzi bat erabilita, exekuzio-garaian, `Panel` desberdinak ikuskatu nahi ditugu eta, horretarako, hasierako `Panel` nagusia euskarri gisa erabiliko dugu beste `Panel` desberdinak ikuskatzeko. Hona hemen egingo diren urratsak nahi duguna gauzatzeko:

1. Eraiki edukiontzi bat Panel motakoa CardLayout administratzailea daukana.

```
Panel PNagusi;
PNagusi=new Panel();
PNagusi.setLayout(new CardLayout());
```

2. Eraiki aplikazioaren `Panel` desberdinak eta txertatu `Panel` bakoitzean `Panel` horretako osagaiak.

```
Panel PBotoiak, PEremuak;
PBotoiak=new Panel();
PBotoiak.setLayout(new FlowLayout(FlowLayout.CENTER,60,0));
for (int i=1;i<4;i++)
       PBotoiak.add(new Button("Botoi "+i));

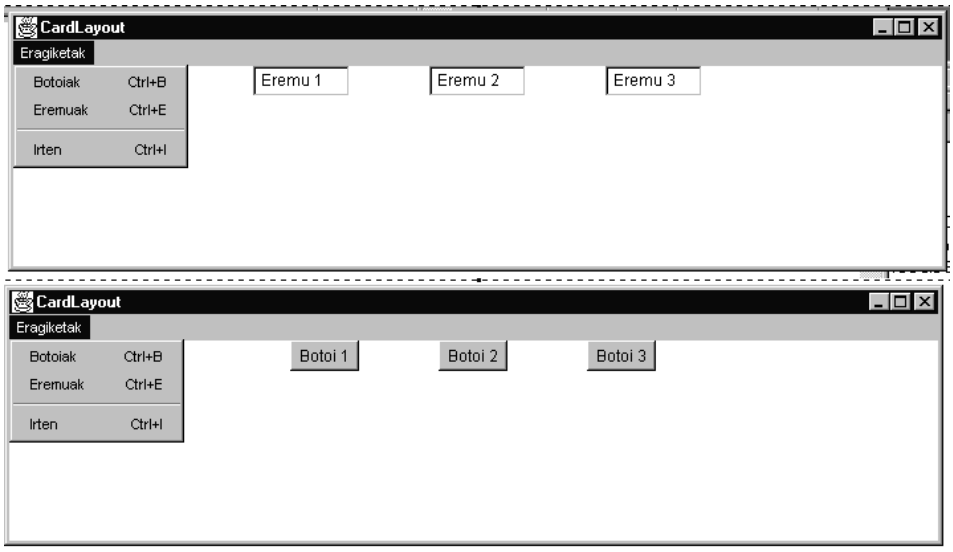
PEremuak=new Panel();
PEremuak.setLayout(new FlowLayout(FlowLayout.CENTER,60,0));
       for (int i=1;i<4;i++)
       PEremuak.add(new TextField("Eremu "+i));
```

3. `Container` klasearen `add()` metodoa erabilita, txertatu bigarren urratsean sortu ditugun `Panel` desberdinak, lehenengo urratsean eraiki dugun `Panel` nagusian.

```
       PNagusi.add("Botoiak",PBotoiak);
       PNagusi.add("Eremuak",PEremuak);
```

4. `CardLayout` klasearen `show()`, `next()`, `first()` edo `last()` metodoak erabilita, ikuskatu interesatzen zaizkigun `Panel` motako objektuaren osagaiak.

```
((CardLayout)PNagusi.getLayout()).show(PNagusi,"Botoiak");
((CardLayout)PNagusi.getLayout()).show(PNagusi,"Eremuak");
```



1.16. irudia. Adibidearen interfaze grafikoa.

Urrats guztietan ipini ditugun adibide guztiak kateatzen baditugu, aplikazio txiki bat eraiki dezakegu honako betekizun hauekin: aplikazio bat programatu nahi dugu marko (`Frame`) bat izango duena edukiontzi nagusi bezala. Marko nagusi horrek bi gune izango ditu. Gune bakoitza beste edukiontzi (`Panel`) bat balitz bezala egituratuko dugu. Markoaren azpian joango den menuan hiru osagai, `MenuItem` motakoak, kokatuko ditugu, aplikazioaren gertaerak sortuko dituztenak. `Botoiak` identifikadorearekin agertzen den aukera zapaltzen dugunean, marko nagusiaren goiko aldean hiru botoi agertzea nahi dugu; aldiz, `Eremuak` identifikadorearekin agertzen den aukera zapaltzen dugunean, hiru testu-eremu (`TextField`) agertuko dira. Hori guztia egiteko ez dugu edukiontziz (`Framez`) aldatu nahi, alegia, edukiontziaren eguneraketak exekuzio-garaian gauzatuko dira.

Betekizunak aztertu ondoren konturatu gara `CardLayout` administratzaileak zeharo erraztuko digula programazio-lana eta administratzaile hori erabiltzea erabakitzen dugu. Gertaeren trataera diseinatu eta programatu ondoren, aplikazio osoak honako iturburu-kode hau izango du:

```
import java.awt.*;
import java.awt.event.*;

class Administratzaile extends Frame implements ActionListener
{ Panel PNagusi;
 Administratzaile()
 {  Panel PBotoiak, PEremuak;

   setTitle("CardLayout");

   menu_egin();

   PBotoiak=new Panel();
   PBotoiak.setLayout(new FlowLayout(FlowLayout.CENTER,60,0));
   for (int i=1;i<4;i++)
      PBotoiak.add(new Button("Botoi "+i));

   PEremuak=new Panel();
   PEremuak.setLayout(new FlowLayout(FlowLayout.CENTER,60,0));
   for (int i=1;i<4;i++)
    PEremuak.add(new TextField("Eremu "+i));

   PNagusi=new Panel();
   PNagusi.setLayout(new CardLayout());
   PNagusi.add("Botoiak",PBotoiak);
   PNagusi.add("Eremuak",PEremuak);

   add(PNagusi);
   pack();
   show();
  }
```

```
  void menu_egin()
  {  Menu menu= new Menu("Eragiketak");

     MenuItem Botoiak=new MenuItem("Botoiak",new MenuShortcut('b' ));
     Botoiak.addActionListener(this);
     menu.add(Botoiak);

     MenuItem Eremuak=new MenuItem("Eremuak",new MenuShortcut('e' ));
     Eremuak.addActionListener(this);
     menu.add(Eremuak);

     menu.addSeparator();

     MenuItem irten=new MenuItem("Irten",new MenuShortcut('i' ));
     irten.addActionListener(this);
     menu.add(irten);

     MenuBar barra=new MenuBar();
     barra.add(menu);
     setMenuBar(barra);

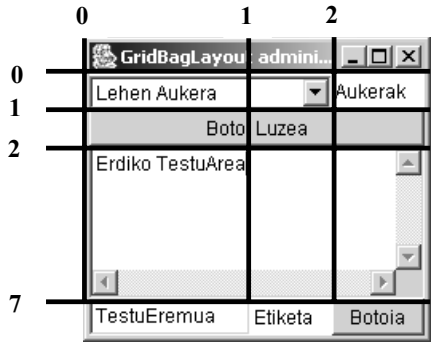
   }

   public void actionPerformed(ActionEvent e)
   {  String agindu=e.getActionCommand();
      if (agindu.equals("Botoiak"))
      {
      ((CardLayout)PNagusi.getLayout()).show(PNagusi,"Botoiak");
       }
      else if (agindu.equals("Eremuak"))
            {

((CardLayout)PNagusi.getLayout()).show(PNagusi,"Eremuak");
            }
           else if (agindu.equals("Irten"))
                {  dispose();
                   System.exit(0);
                }
    }
  public static void main(String args[])
  {  new Administratzaile();}
```

Bukatzeko diseinu-administratzaileen azalpena, azken administratzailea aztertuko dugu: `GridBagLayout`. Administratzaile hau denetatik konplexuena da baina behin ezagututa eta erabilita, emaitza oparoenak eman ditzake. Administratzailearen izenak berak aztarnak ematen dizkigu bere portaera nondik norakoa izango den jakiteko. Agerian dagoen bezala, hasierako `Grid` horrek lotura estuak ditu `GridLayout` administratzailearekin. Hemen ere, `GridLayout` administratzailearekin gertatu bezala, edukiontzia kutxatiletan egituratzen du administratzai-

leak baina administratzaile hau askoz ere malguagoa da eta, hemen, kutxatiletan txertatuko ditugun osagaien neurriak alda daitezke, egokituz gure nahietara. Hona hemen adibide bat:



1.17. irudia. Adibidearen interfaze grafikoa.

```
import java.awt.*;

class Administratzailea extends Frame
{
 Administratzailea()
 {  setTitle("GridBagLayout administratzailea");
    //administratzailea finkatzen dugu
    GridBagLayout admi=new GridBagLayout();
    setLayout(admi);
    // Objektu hau beharrezkoa da edukiontzian osagaiak txertatzeko
    GridBagConstraints con= new GridBagConstraints();
    // Osagaiak leku guztia hartuko du
    con.fill=GridBagConstraints.BOTH;

Choice menu=new Choice();
    menu.addItem("Lehen Aukera");
    menu.addItem("Bigarren Aukera");
    menu.addItem("Hirugarren Aukera");
    con.gridx=0;
    con.gridy=0;
    con.gridwidth=2;
    con.gridheight=1;
    admi.setConstraints(menu,con);
    add(menu);

Label lTestu=new Label("Aukerak");
    con.gridx=2;
    con.gridy=0;
    con.gridwidth=1;
    con.gridheight=1;
    admi.setConstraints(lTestu,con);
    add(lTestu);
```

```
Button botoiLuze=new Button("Botoi Luzea");
   con.gridx=0;
   con.gridy=1;
   con.gridwidth=3;
   con.gridheight=1;
   admi.setConstraints(botoiLuze,con);
   add(botoiLuze);

TextArea area=new TextArea("Erdiko TestuArea",5,10);
   con.gridx=0;
   con.gridy=2;
   con.gridwidth=3;
   con.gridheight=4;
   admi.setConstraints(area,con);
   add(area);

TextField eremu=new TextField("TestuEremua");
   con.gridx=0;
   con.gridy=7;
   con.gridwidth=1;
   con.gridheight=1;
   admi.setConstraints(eremu,con);
   add(eremu);

Label etiketa=new Label("Etiketa");
   con.gridx=1;
   con.gridy=7;
   con.gridwidth=1;
   con.gridheight=1;
   admi.setConstraints(etiketa,con);
   add(etiketa);

   Button botoia=new Button("Botoia");
   con.gridx=2;
   con.gridy=7;
   con.gridwidth=1;
   con.gridheight=1;
   admi.setConstraints(botoia,con);
   add(botoia);

   pack();
   show(); }

   public static void main(String args[])
   { new Administratzailea();
   }
 }
```

Programa hori egikaritu ondoren **1.17. irudia**n agertzen den irteera grafikoa lortuko genuke. Konturatu, osagaiak txertatzeko era, askoz ere malguagoa dela orain arte ikusitako administratzaileekin parekatuz gero. Malgutasun hori lortzeko `GridBagConstraints` motako objektu bat eraikiko da lehendabizi eta, ondoren, `gridx` eta `gridy` aldagaietan osagaiaren koordenatuak adieraziko dira. Osagaiaren zabalera eta luzera aldatu nahi baditugu, beste bi aldagai erabili behar ditugu: `gridwidth` eta `gridheight`.

Hori guztia egin ondoren, osagai bakoitzarentzat, bakarrik gelditzen zaigu adieraztea administratzaileari zeintzuk izango diren osagai bakoitzaren murriztapenak. Hori egiteko, `setConstraints(Component <osagaia>, GridBagConstraints <ezaugarriak>)` metodoa erabiliko dugu.

`GridBagLayout` administratzailearen ezaugarri horiek guztiak argitzeko azter dezagun edukiontzi nagusian txertatu dugun lehen osagaia:

```
Choice menu=new Choice();
menu.addItem("Lehen Aukera");
menu.addItem("Bigarren Aukera");
menu.addItem("Hirugarren Aukera");
con.gridx=0;
con.gridy=0;
con.gridwidth=2;
con.gridheight=1;
admi.setConstraints(menu,con);
add(menu);
```

Lehen lau lerroak, dagoeneko ezagunak zaizkigu. Lerro horietan `Choice` motako osagaia eraikitzen dugu eta ondoren balioekin betetzen dugu. Hori egin ondoren, osagaiaren koordenatuak[18] (`gridx` eta `gridy`) eta neurriak (`gridwidth` eta `gridheight`) adierazten ditugu, eta, bukatzeko, administratzailea ohartarazten dugu `setConstraints()` metodoa erabilita. Osagaiaren ezaugarri guztiak finkatu ondoren, osagaia edukiontziari gehitzea gelditzen zaigu bakarrik.

1.2. INFORMAZIOAREN IRAUNKORTASUNAREN KUDEAKETA: JBDC

1.2.1. Sarrera

Jakina da Javan objektuak sortu ahala bizirik irauten dutela, aplikazioa egikaritzen ari den bitartean. Askotan objektu horien informazioa biltegiratu nahi izaten dugu, etorkizunean eskuratzeko eta berriz aplikazioan erabiltzeko.

Aplikazioa txikia denean, nahikoa izaten da objektuaren aldagaiak fitxategietan gordetzea eta gero, eskuratu eta memorian eraiki. Bide hori egokia litzateke

18. Koordenatuak ezkerretik eskuinera eta goitik behera adierazten dira.

baldin objektuaren aldagaiak datu mota oinarrizkoak balira; gauzak zeharo aldrebesten dira, baldin aldagaien artean beste objektu-instantzia batzuk baditu biltegiratu nahi dugun objektuak. Beste batzuetan, ordea, Javak berak duen baliabideaz, hau da, serializazioaz baliatzen gara informazioa biltegiratzeko.

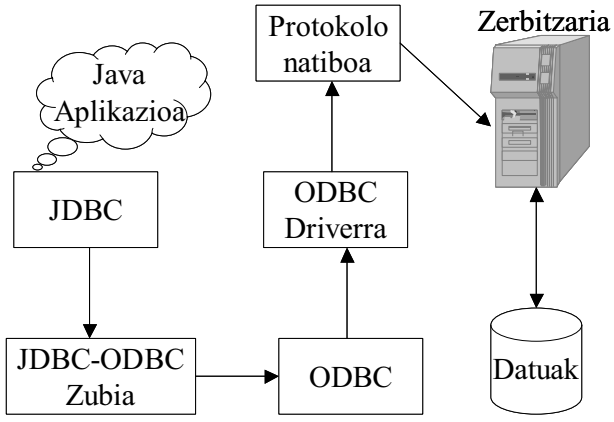
Windows sistema eragileak ODBC (Open Database Connectivity) estandarra erabiltzen du Windows pean egikaritzen diren aplikazioak datu-baseen kudeaketa-sistema desberdinekin komunikatzeko. Antzeko funtzionalitatea eskaintzen du Javaren JDBC (Java Database Connectivity) APIak Javaz idatzitako aplikazioentzat.

1.2.2. Kontrolatzaile motak

Datu-basera atzitu nahi badu Javaz idatzitako aplikazio batek, JDBC driver edo kontrolatzaile egoki bat behar du. Kontrolatzailea kargatu ondoren, klase egokiak erabil ditzake aplikazioak, atzipenak egiteko datu-basera. Hasierako bide hau nahitaezkoa da aplikazioan egikaritzea eta ondo ulertu behar du programatzaileak nola funtzionatzen duen, gero, atzipenak egiteko datu-basera.

Lau kontrolatzaile mota daude:

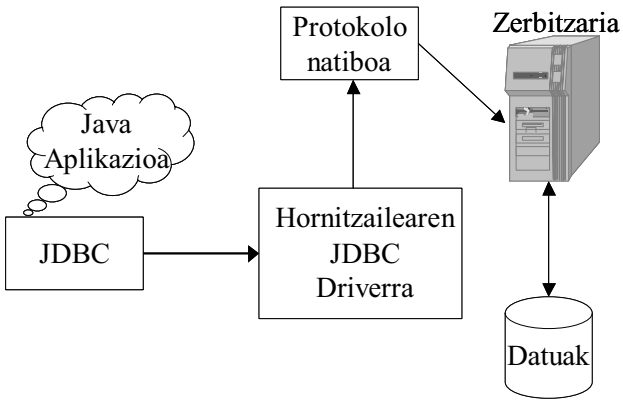
1. *JDBC-ODBC zubia*. Aukera errazena da baina agian ez eraginkorrena. Bere filosofia da aurretik egindakoa erabiltzea eta, horregatik, ODBCk eskaintzen duen azpiegitura guztiaz baliatzen da. Kontrolatzaile honek ODBCra bihurtzen ditu JDBCren dei guztiak eta ODBCren kontrolatzailea arduratzen da datu-baseen kudeaketa-sistema eraiki duen ekoizlearen liburutegi natibora dei egiteaz. Ikusten den legez, bidea erraztu egin dugu ODBCren esku utziz ardura gehienak baina, aldi berean, bidea luzatu eta murriztu egin dugu aplikazioaren ebazpena Windows plataformara lerrokatuz. **1.18. irudia**k JDBC-ODBC zubiaren diagrama funtzionala jasotzen du.



1.18. irudia. JDBC-ODBC zubiaren diagrama funtzionala.

2. *Java natiboa*. Kontrolatzaile mota honek ODBC maila baztertu egiten du eta zuzenean hitz egiten du datu-baseen kudeaketa-sistemaren liburutegi natiboarekin. Ikusten den legez, kontrolatzaile hau % 100 Java da baina liburutegi natiboak eduki behar ditu bezeroaren aldean, deiak ondo egikaritzeko. Horrek, murriztapenak sortuko ditu aplikazio mota batzuetan, adibidez, applet-ak erabili nahi dituganean.

3. *% 100 Java/Protokolo natiboa*. Hirugarren kontrolatzaile motak (**1.19. irudia**) gehiago laburtzen du bidea eta liburutegi natiboen beharra ere ezabatu egiten du. JDBCren eskaera guztiak datu-baseen kudeaketa-sistemaren zerbitzariaren eskaera bihurtzen dira. Kontrolatzaileak sarearen protokolo natiboa erabiltzen du datu-baseen kudeaketa-sistemarekin komunikatzeko, plataformarekiko independentzia lortuz. Murriztapen bakarra da, ezen, kontrolatzaile honekin, bezeroa datu-baseen kudeaketa-sistemaren zerbitzari mota horren menpe gelditzen dela. Adibidez, Microsoft SQL Server-ek erabiltzen duen sareko protokoloa eta PostGres-ek edo Oracle-k erabiltzen dutena desberdinak dira, eta horrek murriztapenak dakarzkio kontrolatzaile mota honi.



1.19. irudia. Java/Protokolo natibo kontrolatzailearen diagrama funtzionala.

4. *% 100 Java/Protokolo independentea*. Kontrolatzaile honetan, JDBC deiak ez dira protokolo natibo batera bideratzen aurreko kontrolatzaile motan bezala, hemen, bitartekari bat erabiltzen da eta protokoloa generikoa da. JDBC kontrolatzaileak bitartekariari egiten dizkio deiak lehenengo eta behin. Deia zerbitzarian jaso ondoren, datu-baseen kudeaketa-sistemaren ekoizlearen liburutegi natibora bideratzen du bitartekariak. Bitartekaria zerbitzarian instalatzen da eta protokolo generikoa erabiltzen da. Protokolo hori sareko protokoloa da, eta independentea, datu-baseen kudeaketa-sistemaren zerbitzariarekiko.

1.2.3. Prozesu orokorraren ezaugarri nagusiak

Aplikazio informatiko batek datu-base bat atzitu nahi badu eta lan egin berarekin, bi urrats gauzatu behar ditu hasieran:

1. Kontrolatzailea kargatu.
2. Konexioa sortu datu-basearekin.

1. Kontrolatzailea kargatu.

Kontrolatzailea kargatzeko nahikoa da `Class` klasearen `forName()` metodo estatikoa egikaritzea. `forName()` metodoaren parametroak, honako sintaxi hau izan beharko luke: *enpresaIzena.datu-baseIzena.kontrolatzaileIzena* eta, beraz, kontrolatzaileak, honako azpidirektorioan egon beharko luke: `enpresaIzena/datu-baseIzena/kontrolatzaileIzena.class`.

2. Konexioa sortu datu-basearekin.

Kontrolatzailea arazorik gabe kargatu ondoren, datu-basearekin konektatu behar du. Hori egiteko, `DriverManager` klasearen `getConnection()` metodoa egikarituko du. Metodoak hiru parametro onartzen ditu baina gainkargatuta dagoenez, nahikoa izaten da batzuetan bat erabiltzea. Hona hemen, `getConnection()` metodoaren adibide bat:

```
getConnection("jdbc:obdc:Irakasgaiak","irakaslea","irak020");
```

Lehen parametroak URL bat izan behar du, eta honako sintaxi hau eduki:

```
jdbc:azpiprotokoloa//zerbitzaria:portua/datu-basea
```

Gure kasuan, JDBC-ODBC zubia erabiltzea erabaki dugunez, hala adierazten dugu parametroaren zati horretan. Parametroa bukatzeko, `Irakasgaiak` adierazpena idatzi dugu. Izen hori datu-basearen izen logikoa da eta ez du zertan datu-base fisikoaren izenarekin bat etorri, gero ikusiko dugun bezala. Gure kasuan, ODBC kontrolatzailea konfiguratu beharko dugu, Access motako datu-baseetara atzipena lortzeko.

`DriverManager` objektuak lana ondo egin badu eta konexioa gauzatu bada, `getConnection()` metodoak `Connection` motako objektu bat itzuliko du. Datu-basearekin lan egiteko beharrezkoak diren metodo gehienak eskaintzen ditu `Connection` klaseak, adibidez, konexioa itxi (`close()`), commit bat gauzatu (`commit()`), transakzioak kudeatu (`getTransactionIsolation()` `/setTransactionIsolation (int` maila)) edo `Statement` objektua sortu SQL aginduak bidaltzeko (`createStatement()`).

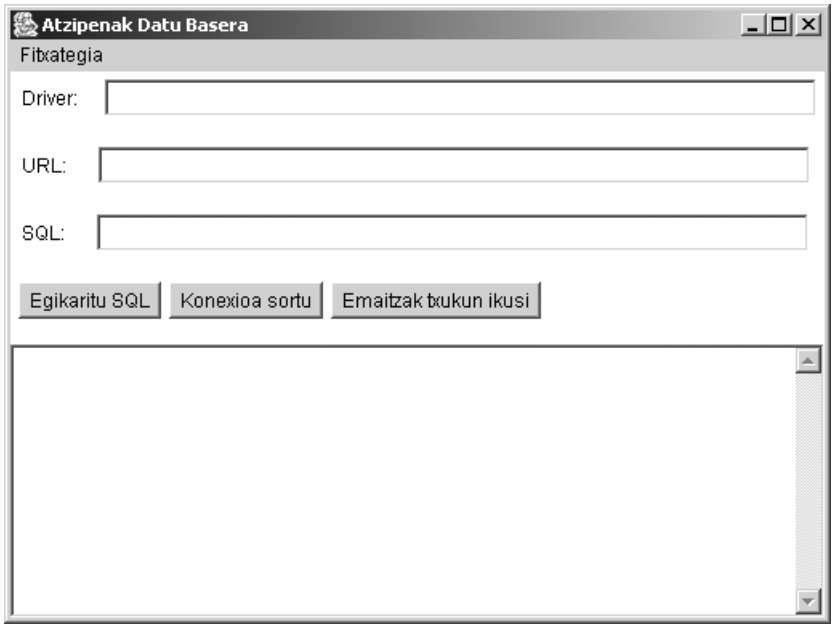
Guk, oraingoz, `createStatement()` metodoa erabiliko dugu. `Statement` interfazearen instantzia bat eraikitzen du metodo horrek eta prest uzten digu azpiegitura teknologiko guztia SQL aginduak egikaritzeko. SQL agindua egikaritu ondoren, `ResultSet` motako objektu batean gelditzen dira emaitzak. `getXXX()` metodoak erabilita eskuratzen ditugu emaitzak. Metodoaren `XXX`-k Javaren datu mota bat adierazten du, adibidez, `getString()`[19], `getFloat()`, `getInt()` eta abar.

Ur handiagotan sartu baino lehen, datu-base bat erabiltzen duen aplikazio txiki bat eraikiko dugu. Aplikazioa erabat generikoa izango da eta aukera emango dio erabiltzaileari berak adierazteko erabili nahi duen kontrolatzailea, baita atzitu nahi duen datu-basea ere. Erabiltzaileak idatzitako SQL aginduak egikarituko ditu aplikazioak eta testu-area batean utziko ditu emaitzak. Nahi adina SQL agindu egikaritu dezake erabiltzaileak eta aplikazioak ematen dituen emaitzak ikusi gero. **1.20. irudia**k erakusten du zein izango den aplikazioaren interfaze grafikoa.

Interfaze grafikoak hiru botoi nagusi ditu datu-basearen kudeaketa gauzatzeko. Lehenengo eta behin, `Driver` eta `URL` eremuak beteko dira aplikazioa martxan jartzeko eta *`konexioa sortu`* botoia zapaldu. Horrela, arazorik ez bada sortzen, hau da, ez bada salbuespenik altxatzen, kontrolatzailea kargatuko da eta konexioa sortuko da datu-basearekin. Gauzak ondo joan direla jakiteko, "*konexioak ondo gauzatu dira*" mezua aterako du aplikazioak testu-arean eta prest geldituko da SQL aginduak jasotzeko.

Konektatu garen datu-basearen egitura ezagututa, hau da, jakinda datu-basearen taulak zeintzuk diren eta nola dauden diseinatuta, SQL aginduak idazten ditugu `SQL` eremuan. Agindua, adibidez, SELECT bat izan bada, bere emaitzak testu-arean azaltzen dira. Testu-arearen lehen lerroan, emaitzaren zutabe bakoitzaren identifikadoreak agertuko dira. Identifikadore horiek zuzenean eskuratzen dira datu-basetik eta bertako atributuen izenak izango dira. Prozesu hori guztia gauzatzeko, *`Egikaritu SQL`* botoia zapaldu behar du erabiltzaileak. Hirugarren botoiaren funtzionalitatea programatzea har ezazu ariketatzat, irakurle, gutxi baita egin beharrekoa.

19. Benetan, datu mota bakoitzeko bi metodo `getXXX()` daude:
    - `getXXX(String <zutabea>)`. Parametro horrekin, emaitzaren lerroaren barnean izen horrekin identifikatuta agertzen den balioa eskuratzen du metodoak. Adibidea: `getString("Izena")`.
    - `getXXX(int <zutabeTokia>)`. Parametro horrekin, emaitzaren lerroaren barnean toki horretan dagoen balioa eskuratzen du metodoak. Adibidea: `getString(2)`.



1.20. irudia. Adibidearen interfaze grafikoa.

Aplikazioaren betekizunak zeintzuk diren aztertuta, hona hemen aplikazioaren iturburu-kodea:

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class AtzipenakDB extends Frame {
 TextField kontrolatzailea = new TextField(60);
 TextField url = new TextField(60);
 TextField sql = new TextField(60);
 Button ESql = new Button("Egikaritu SQL");
 Button Konexioa = new Button("Konexioa sortu");
 Button Emaitzak = new Button("Emaitzak txukun ikusi");

 TextArea emaitzaArea = new TextArea(10,60);
 Connection connection;

 public static void main(String args[]){
  AtzipenakDB app = new AtzipenakDB();
 }

 public AtzipenakDB() {
  super("Atzipenak Datu-basera");
  hasieraketa();
```

```
  pack();
  addWindowListener(new WindowEventHandler());
  show();
 }
 void hasieraketa() {
  hasieraketaMenuBar();
  setLayout(new GridLayout(2,1));
  Panel goiPanela = new Panel();
  goiPanela.setLayout(new GridLayout(4,1));
  Panel panelak[ ]=new Panel[ 4] ;
  for(int i=0;i<panelak.length;++i){
   panelak[ i] =new Panel();
   panelak[ i] .setLayout(new FlowLayout(FlowLayout.LEFT));
  }
  panelak[ 0] .add(new Label("Driver:"));
  panelak[ 0] .add(kontrolatzailea);
  panelak[ 1] .add(new Label("URL: "));
  panelak[ 1] .add(url);
  panelak[ 2] .add(new Label("SQL: "));
  panelak[ 2] .add(sql);

  BotoiakKudeatzailea botoiKu = new BotoiakKudeatzailea();

  ESql.addActionListener(botoiKu);
  Konexioa.addActionListener(botoiKu);
  Emaitzak.addActionListener(botoiKu);

  panelak[ 3] .add(ESql);
  panelak[ 3] .add(Konexioa);
  panelak[ 3] .add(Emaitzak);

  for(int i=0;i<panelak.length;++i)
  goiPanela.add(panelak[ i] );
  add(goiPanela);
  add(emaitzaArea);
 }

 void hasieraketaMenuBar() {
  MenuBar menuBar = new MenuBar();
  Menu MenuFitxategia = new Menu("Fitxategia");
  MenuItem IrtenFitxategia = new MenuItem("Irten");
  IrtenFitxategia.addActionListener(new MenuItemHandler());
  MenuFitxategia.add(IrtenFitxategia);
  menuBar.add(MenuFitxategia);
  setMenuBar(menuBar);
 }

class BotoiakKudeatzailea implements ActionListener {
  public void actionPerformed(ActionEvent ev){
   String s=ev.getActionCommand();
```

```
   if(s==”Egikaritu SQL”) atzituDB();
      else if(s==”Konexioa sortu”) konexioaDB();
      else if(s==”Emaitzak txukun ikusi”) txukunIkusiDB();
  }
 }

 void atzituDB() {
  try{
   Statement statement = connection.createStatement();
   boolean ukanEmaitzak = statement.execute(sql.getText());
   if(ukanEmaitzak){
    ResultSet emaitza = statement.getResultSet();
    if(emaitza!=null) ikuskatuEmaitzak(emaitza);
   }else emaitzaArea.setText(“”);
   connection.close();
  }catch(Exception ex){
   emaitzaArea.setText(ex.toString());
  }
 }
 void ikuskatuEmaitzak(ResultSet r) throws SQLException {
  ResultSetMetaData rmeta = r.getMetaData();
  int kopuruZutabe=rmeta.getColumnCount();
  String testu=””;
  for(int i=1;i<=kopuruZutabe;++i) {
   if(i<kopuruZutabe)
    testu+=rmeta.getColumnName(i)+” | “;
   else
    testu+=rmeta.getColumnName(i);
  }
  testu+=”\n”;
  while(r.next()){
   for(int i=1;i<=kopuruZutabe;++i) {
    if(i<kopuruZutabe)
     testu+=r.getString(i)+” | “;
    else
     testu+=r.getString(i).trim();
   }
   testu+=”\n”;
  }
  emaitzaArea.setText(testu);
 }

void konexioaDB() {
   try{
   Class.forName(kontrolatzailea.getText());
   connection =DriverManager.getConnection(url.getText());
   emaitzaArea.setText(“Konexioak ondo gauzatu dira”);
   }catch(Exception ex){
   emaitzaArea.setText(ex.toString());
  }
```

```
 }

void txukunIkusiDB(){  }


  class MenuItemHandler implements ActionListener {
  public void actionPerformed(ActionEvent ev){
   String s=ev.getActionCommand();
   if(s==”Irten”){
    System.exit(0);
   }
  }
 }
 class WindowEventHandler extends WindowAdapter {
  public void windowClosing(WindowEvent e){
   System.exit(0);
  }}}
```

Aplikazioa martxan jartzeko lehen urratsa interfaze grafikoa eraikitzea da, eta hori egiteko, `hasieraketa()` metodoa darabilgu. AWT motako osagai desberdinak sortzen ditu metodoak eta paneletan kokatzen ditu. Osagaiak kokatzeko, `GridLayout` motako diseinu-administratzailea erabili dugu. Osagaiak eraiki eta botoien kudeatzailea prestatu eta gero, aplikazioa zain gelditzen da erabiltzailearen gertaerak jasotzeko.

Gertaeren trataera, `BotoiakKudeatzailea` inner[20] klaseak kudeatzen du `actionPeformed()` metodoa erabilita. Hemen erabakitzen da zein metodo egikarituko den botoi batek gertaera bat sortzen duenean.

Kontrolatzailea kargatu eta datu-basearekiko konexioa sortzen du `konexioaDB()` metodoak. Prozesu guztia, `try-catch` bloke batek babestuko du. Konexioa sortu ondoren, `Statement` objektu bat instantziatzen du `atzituDB()` metodoak SQL agindua egikaritzeko. `execute()` metodoak egikaritzen du SQL agindua eta `true` balio boolearra itzultzen du, `ResultSet` motako emaitza itzultzen badu SQL aginduak. Egiaztatu ondoren `true` balio boolearra itzuli dela, SQL aginduak itzuli dituen emaitzak eskuratzen ditugu `Statement` motako objektuaren `getResultSet()` metodoa egikarituta. Emaitza, `ResultSet` motako objektu batean biltegiratzen dugu.

SQL aginduaren emaitzak interfaze grafikoan erakusteko, `ikuskatuEmaitzak()` metodoa erabiliko dugu. Metodoak begizta nagusi bi ditu balioak ikuskatzeko. Lehenengo begiztarekin, emaitzaren metainformazioa eskuratzen dugu eta, horretarako, `ResultSetMetaData` motako objektu bat instantziatzen dugu eta `getColumnCount()` eta `getColumnName()` metodoak erabilita `ResultSet` emaitzaren metainformazioa eskuratzen dugu. Zenbat zutabe

20. Inner klaseak beste klase baten barnean bizi dira bizimodu sinbiotikoan.

itzuli diren emaitzan eskuratzen ditu `getColumnCount()` metodoak eta zutabe bakoitzaren izena zein den datu-basean adierazten du `getColumnName()` metodoak. Informazio horrekin, interfaze grafikoaren testu-arean idatziko dugun irteera nagusiaren goiburua eraikitzen dugu.

Goiburua idatzita, falta dugun gauza bakarra, `ResultSet` objektuak dituen balioak bistaratzea da. Horretarako, `ikuskatuEmaitzak()` metodoaren bigarren begizta egikaritzen da eta `getString()` metodoa egikarituz itzulitako tupla bakoitzaren atributu bakoitzaren balioak eskuratzen ditugu. Konturatu ezen, `ResultSet` motako objektuak `next()` metodoa egikaritzen duenean, itzulitako emaitzen kurtsorea hurrengo lerroan kokatzen dela, edo lehenengoan, lehenengo aldiz egikaritzen bada. `true` balioa boolearra itzuliko du metodoak eragiketa gauzatzen badu, eta, `false` bestela.

Laburbil dezagun orain arte egindakoa, kontzeptu teoriko asko atera baitira eta batzuk arrapaladan. Kontrolatzailea kargatu ondoren eta datu-basearekiko konexioa finkatuta, harremanetan jarri gara berarekin SQL aginduak egikarituta. Hemen egingo dugu lehen etenaldia; SQL aginduak egikaritzeko `execute` motako beste bi metodo baititugu:

- `executeQuery(String` <SQL_Agindua>)
- `executeUpdate(String` <SQL_Agindua>)

Aplikazio batzuetan, `execute` metodoak egikaritzen duen SQL aginduaren egitura behin eta berriz errepikatzen da balioak aldaturik. Demagun aitzinean erabili dugun `Irakasgaiak` datu-baseak `Irakasgaia` taula duela honelako diseinuarekin:

IrakIzena	Tokia	Irakaslea	Kredituak
Softwarearen Ingeniaritza	G123		
Datu-baseen Garapena	LAB321		
Sistema Eragileak	G444		
Inf. S. Analisia eta Diseinua	GELAT114		

1.21. taula. `Irakasgaia` taularen diseinua eta balioak.

Taularen lehenengo bi atributuak hasieratuta daude datorren ikasturterako baina oraindik ez dakigu zein izango den irakasgai bakoitzaren irakaslea eta zenbat kreditu izango duen irakasgaiak. Bi atributu horiek hasieratzeko, aplikazio informatiko bat programatzen badugu, honelako SQL aginduak agertuko dira iturburuan:

```
UPDATE Irakasgaia SET Irakaslea = "Xabier Zubizarreta",
                      Kredituak = 15
   WHERE IrakIzena LIKE "Softwarearen Ingeniaritza"
```

Beste irakasle bat esleitu nahi badiogu beste irakasgai bati, agindua errepikatu egingo da balio berriekin. Horrelako egoeretan, komeni izaten da `Statement` objektu bat erabili ordez, `PreparedStatement` objektua erabiltzea.

Beste abantaila bat ere badu `PreparedStatement` objektuak: SQL agindua aurrekonpilatu egiten dela. Horrela, `PreparedStatement` bat egikaritzen denean datu-basearen kontrolatzaileak zuzenean egikarituko du aurretik konpilatu gabe.

`PreparedStatement` objektu bat sortzea oso erraza da, nahikoa da `?` ikurra jartzea aldatuko diren balioetan. Horrela, aurreko SQL agindua, honela gelditu litzateke:

```
UPDATE Irakasgaia SET Irakaslea = ?,
                      Kredituak = ?
            WHERE IrakIzena LIKE ?
```

Bestela, `PreparedStatement` objektua sortzeko nahikoa izango da kontuan edukitzea nola sortzen zen `Statement` objektu bat, jarraituko diren urratsak berberak baitira, alegia, `Connection` objektu bat eraiki eta `Connection` motako objektu horri `prepareStatemen()` metodoa egikaritzeko agindu:

```
Connection con = DriverManager.getConnection
("jdbc:obdc:Irakasgaiak","irakaslea","irak020");

PreparedStatement eguneratuIrak = con.prepareStatement
("UPDATE Irakasgaia SET Irakaslea = ?,Kredituak = ? WHERE IrakIzena
LIKE ?");
```

Prozesu guztia bukatzeko gelditzen zaigun gauza bakarra da SQL aginduaren `"?"` hutsuneak betetzea, agindua egikaritu baino lehen. Horretarako, `PreparedStatement` objektuaren `setXXX()` metodoak erabiliko ditugu. Gure kasuan, lehenago jarri dugun adibidea berreraiki nahi badugu, honako hau egikaritu beharko dugu:

```
eguneratuIrak.setString(1, "Xabier Zubizarreta");
eguneratuIrak.setInt(2,15);
eguneratuIrak.setString(3, "Softwarearen Ingeniaritza");
```

Bukatzeko, `executeUpdate()` metodoari deitzen diogu, parametrorik gabe, SQL agindua egikaritzeko:

```
eguneratuIrak.executeUpdate();
```

Lehen aipatutako etenaldiekin jarraituz, hona hemen bigarrena. Javak erabiltzen dituen datu motak eta JDBCk erabiltzen dituenak ez datoz bat batzuetan eta, horregatik, komeni izaten da jakitea `ResultSet` emaitzen edukia zein formatutan dagoen, asoziazio bat gertatzen baita JDBCren eta Javaren datu moten artean.

Komeni da asoziazio hori aurrean edukitzea SQL agindua egikaritzen denean. Gauzak zertxobait aldrebesteko, asoziazioa ez da simetrikoa eta taula bi beharko ditugu datu motak kontrolatzeko.

JDBCk erabiltzen dituen datu motak `Types` klasean daude definituta eta **1.22. taula**n azaltzen dira. **1.23. taula**n JDBCren datu motekin asoziatzen diren Javaren datu motak azaltzen dira.

JDBC datu mota	Java datu mota
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

1.22. taula. JDBCk erabiltzen dituen datu motak.

Java datu mota	JDBC datu mota
String	VARCHAR/LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY /LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME

1.23. taula. Javak erabiltzen dituen datu motak datu-basearen informazioa egokitzeko.

1.2.4. JDBC 2.0 APIaren ezaugarri nagusiak

Javaren teknologiaren baliabide gehienek bezala, bilakaera emankorra izan du JDBC APIak azken urteetan. JDBC 2.0 APIa bilakaera horren emaitza ezagunena da. API berriarekin, besteak beste, emaitzen taularen kurtsorea mugi dezakegu, eguneraketak egin emaitzen taulan, txertatu eta ezabatu lerroak programa bidez eta eguneraketak gauzatu SQL agindu sortak bidalita datu-baseari.

Aipatutako ezaugarri berriak inplementatu nahi baditugu gure aplikazio informatikoan, `Statement` objektua sortzeko erabiltzen genuen `createStatement()` metodoa beste era batera egikaritu behar dugu. Oraingoan, parametro birekin erabiliko dugu: lehenengoan, `ResultSet` motako objektuaren `Type` mota adieraziko dugu eta, bigarrenean, emaitzen taula irakurgarria ez ezik, eguneragarria ere den ala ez adieraziko dugu.

`createStatement()` metodoaren lehen parametroak: `TYPE_FORWARD_ONLY`, `TYPE_SCROLL_INSENSITIVE` edo `TYPE_SCROLL_SENSITIVE` balio konstantea har dezake. Bigarrenak, aldiz: `CONCUR_READ_ONLY` edo `CONCUR_UPDATABLE` balio konstantea hartuko luke, bere aldetik.

`TYPE_FORWARD_ONLY` konstantea erabiltzen badugu metodoaren lehen parametroan, emaitzen taularen kurtsorea aurrera bakarrik mugitu nahi dugula adierazten ari gara. Metodoan ezer adierazten ez badugu, hori izango da balio lehenetsia, programak erabiliko duen `Type` mota.

Emaitzen taula irekita dagoen bitartean aldaketak egiten badira eta `TYPE_SCROLL_INSENSITIVE` konstantea jarri badugu lehen parametroan, ez dira aldaketa horiek islatuko. Aldaketak islatzeko, `TYPE_SCROLL_SENSITIVE` konstantea jarriko da.

`CONCUR_READ_ONLY` balio konstantearekin emaitzen taula irakurgarria izango da bakarrik eta `CONCUR_UPDATABLE` balio konstantearekin, berriz, emaitzen taula eguneratu ere egin daiteke. Ez badugu ezer adierazten metodoan, `CONCUR_READ_ONLY` balio konstantea hartuko da balio lehenetsia.

`ResultSet` motako objektuaren ezaugarriak adierazi ondoren `createStatement()` metodoarekin, JDBC 2.0 APIaren metodo berriak egikaritu ditzakegu. Orain arte emaitzen taularen kurtsorea mugitzeko `next()`[21] metodoa erabiltzen genuen, hori baitzen kurtsorea mugitzeko era bakarra, orain, `previous()` metodoa ere erabil dezakegu eta taula atzetik aurrera irakurri. Taula atzetik aurrera edo toki zehatz batetik hasita irakurtzeko, `first()`, `last()`, `beforeFirst()` eta `afterLast()` metodoak eskaintzen dizkigu API berriak. Hona hemen nola eskura ditzakegun aurreko adibidean erabili dugun `Irakasgaia` taularen balioak, baina oraingoan, atzetik aurrera mugituz SQL aginduaren emaitzen taularen kurtsorea:

```
      Statement                    stmt                    =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet emaitza = stmt.executeQuery("SELECT * FROM Irakasgaia");
      emaitza.afterLast();
      while (emaitza.previous()) {
            String irakasgaia = emaitza.getString("IrakIzena")  ;
            int tokia =  emaitza.getInt("Tokia")     ;
            String irakaslea = emaitza.getString("Irakaslea") ;
      System.out.println(irakasgaia  +"            "+tokia+"
"+irakaslea);
      }
```

`absolute(int)` metodoak argumentuan adierazitako baliora mugituko du kurtsorea. Balioa negatiboa bada, emaitzen taularen bukaeratik hasita argumentuaren balioaren neurrian mugituko du kurtsorea metodoak. `relative()` metodoak kurtsorea mugitu baino lehen une horretan kurtsorea kokatuta dagoen tokia hartuko du erreferentzitzat. Horrela, kurtsorea lehenengo eta behin hamargarren lerroan kokatu nahi badugu, gero zazpigarrenean, eta azkenik bederatzigarrenean, honako kode sorta hau idatziko dugu:

21. Gogoratu irakurle, emaitzak jasotzean, hasieran, kurtsorea lehen lerroaren aurretik kokatzen dela emaitzen taulan. Nahiz eta emaitza lerro bat izan, `next()` metodoa egikaritu behar da taularen lehen tupla eskuratzeko.

```
emaitza.absolute(10);
.....
emaitza.relative(-3);
......
emaitza.relative(2);
```

Gora eta behera, zorabiatu egin bagara, unean uneko kurtsorearen balioa eskuratzen digu `getRow()` metodoak:

```
emaitza.absolute(10);
int lerroa = emaitza.getRow();
.....
emaitza.relative(-3);
int lerroa = emaitza.getRow();
......
emaitza.relative(2);
int lerroa = emaitza.getRow();
```

Kurtsorearen kokapenaz informatuta egon nahi badugu `isFirst()`, `isLast()`, `isBeforeFirst()` eta `isAfterLast()` metodoak erabiliko ditugu. Metodo horiek guztiek balio `boolear` bat itzultzen dute eta oso egokiak dira begiztak kontrolatzeko. Aurreko adibidea hobetu nahi badugu kurtsorea azken lerroaren ondoren kokatuta dagoela ziurtatzeko, `isAfterLast()` metodoa erabiliko dugu lehenago. Hona hemen kode berria `isAfterLast()` metodoa txertatuta hasieran:

```
if (emaitza.isAfterLast() == false)
 {emaitza.afterLast() };
while (emaitza.previous()) {
String irakasgaia = emaitza.getString("IrakIzena")  ;
int tokia =  emaitza.getInt("Tokia")     ;
String irakaslea = emaitza.getString("Irakaslea") ;
System.out.println(irakasgaia   +"            "+tokia+"
"+irakaslea);
}
```

Hasierako baldin aginduak ziurtatu egiten digu ezen kurtsorea `while` begizta nagusian sartu baino lehen, toki egokian kokatuta dagoela, gure kasuan azken lerroaren atzean.

JDBC 2.0 API berriarekin emaitzen taulan bertan gauzatu daitezke eguneraketak SQL agindurik egikaritu gabe. Horretarako, `Statement` motako objektua sortzen dugunean `createStatement()` metodoa erabilita, `CONCUR_UPDATABLE` balio konstantea hartu behar du bigarren parametroak. Horrela, `ResultSet` motako objektua eguneragarria izango da aplikazio informatikoan erabiltzen dugunean:

```
      Connection con = DriverManager.getConnection
("jdbc:obdc:Irakasgaiak","irakaslea","irak020");
      Statement stmt = con.createStatement
(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
      ResultSet emaitza = stmt.executeQuery("SELECT * FROM
Irakasgaia");
```

Eguneragarria izateak hauxe esan nahi du: emaitzen taularekin zuzenean egiten dugula lan SQL agindurik erabili gabe, eta gero, bukaeran, eguneraketak datu-basean txertatu nahi baditugu nahikoa dela `updateRow()` metodoa egikaritzea. Hona hemen adibide bat:

```
emaitza. first();
emaitza.updateInt("Kredituak", 14);
.....
emaitza. updateRow();
```

Agindu sorta horrekin, lehenengo eta behin, emaitzen taularen lehen lerroan kokatzen dugu kurtsorea. Lehen lerroan, *Softwarearen Ingeniaritza* irakasgaiaren informazioa gordetzen dugu. Bigarren lerroan, eguneratu nahi dugun atributua `Kredituak` dela adierazten dugu, `ResultSet` motako objektuaren `updateInt()` metodoa egikarituta. Bukatzeko, eguneraketak eragina izan dezan datu-basean `updateRow()` metodoa egikaritzen dugu. Gogoratu kurtsorearen toki-aldaketa guztiak (`first()`, `last()`, `previous()`...) emaitzen taularen gainean egiten direla eta ez datu-basearen tupletan.

Ez dago esan beharrik hori ez dela bide bakarra eguneraketak egiteko datu-basean. Honela ere egin genezakeen:

```
Stmt.executeUpdate("UPDATE Irakasgaia SET Kredituak = 14" +
"WHERE IrakIzena LIKE ïSoftwarearen IngeniaritzaÏ");
```

Emaitzen taulan dauden bitartean, deuseztatu egin daitezke eguneraketak hala eskatzen badu programaren fluxuak. Eguneraketak deuseztatzeko, `cancelRowUpdates()` metodoa erabiltzen du `ResultSet` motako objektuak:

```
emaitza. first();
emaitza.updateInt("Kredituak", 14);
........
emaitza. cancelRowUpdates();
emaitza.updateInt("Kredituak", 20);
emaitza. updateRow()
```

Aurreko lerroetan ikusi dugu nola gauzatzen den eguneraketa bat SQL agindua erabili gabe. Beste lerro bat txertatu edo baten bat ezabatu nahi badugu, antzera egin beharko dugu.

Lerro berri bat txertatzeko nahikoa da lehenengo eta behin kurtsorea "txertaketa lerrora" mugitzea. Txertaketa lerro hori `ResultSet` motako objektu bati esleitzen zaio eta ez da emaitzen taulako lerro bat. Lerro bat sortzeko toki bat dela pentsa dezakegu. Lerro horretara mugitu ondoren, `moveToInsertRow()` metodoa egikarituta, eguneraketetan erabiltzen genituen `updateXXX()` metodoak erabiliko ditugu, balioak txertatzeko lerro berrian.

Txertaketa gauzatzeko, emaitzen taulan eta datu-basearen taulan `insertRow()` metodoa egikaritzen dugu:

```
emaitza.moveToInsertRow();
emaitza.updateString("IrakIzena","Sistema Banatuak");
emaitza.updateString("Tokia","GELAT1100");
emaitza.updateString("Irakaslea","Pau BeirasMuxia");
emaitza.updateInt("Kredituak", 14);
emaitza. insertRow();
```

Adibidean, lerro berriaren balio guztiak hasieratu ditugu `updateXXX()` metodoarekin. Batzuetan ez da beharrezkoa izaten atributu bat hasieratzea esplizituki, atributu horrek `NULL` balioa har badezake datu-basearen taulan. Atributu hori `NULL` balioarekin hasieratzen da inplizituki, ez badiogu balio bat eman `updateXXX()` metodoarekin, bestela, `SQLException` salbuespena altxatuko da programa egikaritzean.

Lerro bat ezabatzeko nahikoa da emaitzen taulan kurtsorea mugitzea ezabatu nahi dugun lerrora eta `deleteRow()` metodoa egikaritzea:

```
emaitza.absolute(10);
emaitza.deleteRow();
```

Datu-base bat eguneratu nahi dugunean, batzuetan komeni da eta eraginga-rriagoa izaten da eguneraketak banan-banan egin beharrean multzotan gauzatzea. Hala eguneraketak egikaritu nahi ditugunean, `addBatch()` eta `executeBatch()` metodo berriak eskaintzen ditu JDBC 2.0 APIak. Eguneraketa hasi baino lehen, `Connection` motako objektuaren auto-commit era desaktibatzen da, `setAutoCommit(Boolean)` metodoari, parametro gisa `false` balioa pasatuta. Horrela, `executeBatch()` metodoa egikaritzen denean, eguneraketaren bukaeran, ez da arazorik sortuko transakzio mailan. Hona hemen nola egikarituko den eguneraketa multzo bat API berriaren baliabide berri horiekin:

```
Connection con =
DriverManager.getConnection("jdbc:obdc:Irakasgaiak",
"irakaslea","irak020");
Statement stmt = con.createStatement
(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
con.setAutoCommit(false);
stmt.addBatch("INSERT INTO Irakasgaia" + "VALUES('Programazioa I',
'G2211', 'Jon Etxabe', 10)");
stmt.addBatch("INSERT INTO Irakasgaia" + "VALUES('Programazioa II',
'G2200', 'Amaia Etxabe', 15)");
stmt.addBatch("INSERT INTO Irakasgaia" + "VALUES('Programazioa
III', 'G2233', 'Nerea Azurmendi', 8)");
stmt.addBatch("INSERT INTO Irakasgaia" + "VALUES('Metodologia I ',
'G1144', 'Mikel Go•i', 10)");
int [] eguneraKontuak = stmt.executeBatch();
```

Lau aginduak txertatutako ordenan egikarituko dira eta, ondo egikaritzen badira, datu-basearen kontrolatzaileak "eguneraketen kontua" itzuliko du,

adieraziz zenbat lerro eguneratu dituen SQL agindu bakoitzak. Gure kasuan `eguneraKontuak` taulak lau osagai izango ditu eta bat balioa hartuko du osagai bakoitzak, ondo egin bada eguneraketa.

Bi salbuespen mota altxa ditzake eguneraketa mota horrek: `SQLException` eta `BatchUpdateException`. Lehenengoa, ezaguna zaigu dagoeneko, eta datubasearen atzipenarekin dago lotua. Bigarrena, SQL aginduren bat egikaritu ezin denean sortzen da.

`BatchUpdateException` salbuespenak taula bat itzultzen du eta salbuespena altxatzen denean eguneraketen kontua nola dagoen islatzen du. Salbuespen mota horrek, `SQLException` salbuespenetik eratortzen denez, `SQLException` klasearen metodoak erabil ditzake, salbuespenaren nondik norakoak aztertzeko eta, gero, programatzaileak, programa, taxuz uzteko.

Hona hemen nola programatuko genukeen aurreko adibidean `catch` aldea, salbuespenaren informazioa jasotzeko eguneraketaren try-catch blokean:

```
catch(BatchUpdateException b) {
  System.err.println("SQLSalbuespena: " + b.getMessage());
  System.err.println("SQLEgoera: " + b.getSQLState());
  System.err.println("Mezua: " + b.getMessage());
  System.err.println("Ekoizlea: " + b.getErrorCode());
  System.err.print("Eguneraketen kontuak: ");
int [] eguneraKontuak = b.getUpdateCounts();
   for (int i = 0; i < eguneraKontuak.length; i++) {
         System.err.print(eguneraKontuak [i] + " ");
         }
```

1.3. LABURPENA

Objektuetara zuzendutako garapena erabiltzen badute erakunde informatikoek aplikazio informatikoak eraikitzeko, baliteke aplikazioaren analisia eta diseinua UML metodoa erabilita gauzatzea. Analistak, aplikazioaren betekizunak aztertuta, *de facto* estandarra bihurtu den metodoa erabilita aplikazioaren Erabilera Kasu Diagrama eraikiko du aplikazioaren erabilera-kasu nagusiak identifikatzeko eta egituratzeko. Hasierako identifikazio-lanak aplikazioaren erabilera-kasuak azaleratuko ditu, baina, erabilera-kasuen egituraketa gauzatzeko, informazio zehatza eduki behar du diseinatzaileak, diagramaren `extend`-ak eta `use`-ak marrazteko. Azken ezaugarri horiek ondo egituratu direla egiaztatzeko, erabilera-kasu bakoitzaren interfaze grafikoak diseinatu eta eraiki behar ditu diseinatzaileak. Javaren teknologiak AWT eta SWING baliabideak eskaintzen ditu aipatutako interfaze grafikoak eraikitzeko, azkar eta objektuetara zuzendutako paradigman oinarrituta.

Bestalde, aplikazioa egikaritu ahala, objektu-instantziez betetzen joango da memoria. Objektu batzuek aplikazioren exekuzio-fluxua kontrolatzen dute eta

aplikazioaren exekuzioa amaitzen denean desagertu egingo dira lan txukuna beteta. Baina, badaude beste objektu batzuk, domeinuaren Klase Diagrama antolatzeko erabiliko direnak, eta bertako informazioa biltegiratuta eduki nahi da aplikazioaren exekuzioa amaitu baino lehen. Jakina, objektuetara zuzendutako garapenak objektuetara zuzendutako biltegiratzea beharko luke. Zoritxarrez iraunkortasunaren bilakaera historikoak datu-base erlazionalen bidea hartu zuen eta sendotzen joan da urtetik urtera. Hala, memorian instantziatu diren domeinuko objektuak tauletan biltegiratzen dira mapaketa eragiketa eginda. Alderantzizko bidea egiten denean, hau da, datu-basetik memoriara doana, eskuratutako informazioa egokitu egingo da objektu-instantziak eraikita memorian. Javaren teknologiak JDBC APIa eskaintzen dio programatzaileari SQL aginduak egikaritzeko eta eskuratutako informazioa egituratzeko.

1.4. OINARRIZKO BIBLIOGRAFIA

Adamson, C. eta Marinacci, J. (2005): *Swing Hacks*, O´Reilly.

Avondolio, D.; Richardson, W. C.; Schrager, S.; Mitchell, M. W. eta Scanlon, J. (2007): *Professional Java JDK 6 Edition*, Wrox.

Date, C. J. eta Darwen H. (2006): *Databases, Types and the Relational Model*, Addison Wesley.

Elliott, J.; Loy David, M. eta Brian Cole, W.(2002): *Java Swing*, O´Reilly.

Elmasri R. eta Navathe S. B.(2007): *Fundamentals of Database Systems*, Addison Wesley.

Lans, R. F. van der (2006): *Introduction to SQL: Mastering the Relational Database Language*, Addison Wesley Professional.

Parsian, M. (2006): *JDBC Metadata, MySQL, and Oracle Recipes: A Problem-Solution Approach*, Apress.

Parsian, M. (2005): *JDBC Recipes: A Problem-Solution Approach*, Apress.

Robinson, M. eta Vorobiev P. (2003): *Swing*, Manning.

Sperko, R. (2003): *Java Persistence for Relational Databases*, Apress.

1.5. WEB GUNE INTERESGARRIAK

Zer dago bertan	Web gunea
Swing-en tutoretza	`http://java.sun.com/docs/books/tutorial/uiswing/`
Sarrera Swing APIra	`http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=swing`
JDBCren tutoretza	`http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html`
Sarrera JDBC APIra	`http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html`
Adibideak	`http://www.exampledepot.com/`

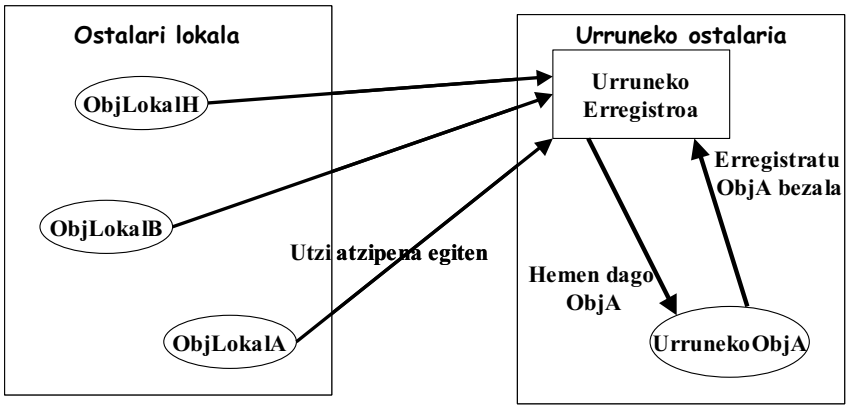
2. Bigarren hurbilketa: oinarrizko bezero/zerbitzari arkitektura: (AWT-SWING/RMI/JDBC)

2.1. RMI (REMOTE METHOD INVOCATION)

Javaren teknologiak eskaintzen duen tresna sistema banatuak eraikitzeko RMI (Remote Method Invocation) da. Sistema banatuetan, objektu lokalek urruneko objektuak erabiltzen dituzte urruneko ostalariko zerbitzuez baliatzeko. Bi objektu horien arteko lotura finkatzeko, RMI APIa erabiltzen da Javan.

Funtsean bi eragiketa nagusi gauzatuko dira: lehenengo eta behin, objektu lokalak urruneko objektuaren erreferentzia beharko du, jakiteko non dagoen eta nola eskatu eta, gero, mezuak bidaliko zaizkio urruneko objektuari, eragiketak gauzatu ditzan. Azkeneko eragiketa hori, geroago adibideetan ikusiko dugun bezala, erabat gardena izango da programatzailearentzat. RMIren diseinatzaileek, irizpide ona erabiliz, ez zuten bereizketarik egin nahi izan dei lokalen eta urruneko deien artean. Horrela, urruneko objektuaren metodoa egikaritu nahi badu programatzaileak, programatzeko orduan objektu lokala izango balitz modura programatuko du.

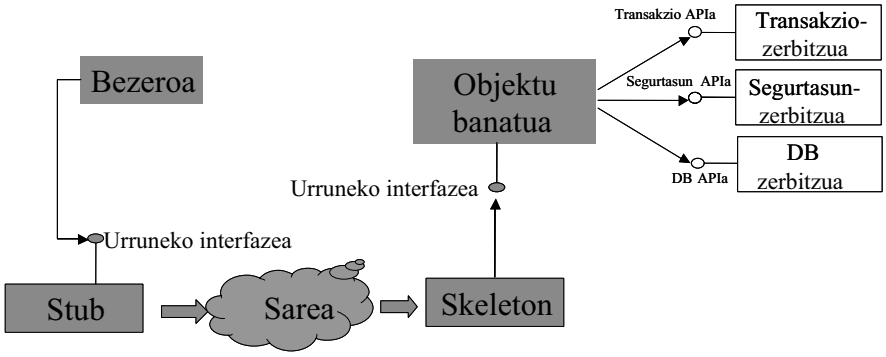
Prozesua hain gardena izateko, aurretik, urruneko objektua erregistratu behar du programatzaileak urruneko zerbitzarian. Hori egin ondoren, eta erregistro-kudeatzailea martxan jarrita, ostalari lokalean dagoen edozein objektuk atzitu eta bere baliabideak erabil ditzake. Orain arte esandakoa, **2.1. irudia**k laburbiltzen du.



2.1. irudia. RMI baliabidearen prozesu nagusia.

Beraz, urruneko objektu batera atzitzeko ez dugu haren kokapen fisikoa ezagutu behar, nahikoa zaigu jakitea urruneko ostalarian zein izenekin izan den erregistratua. Izenen kudeatzaileak erlazionatuta uzten ditu izena eta urruneko objektua, objektua erregistratzen denean.

Objektu lokalak eta urruneko objektuak azpiegitura teknologiko bat beharko dute elkarren berri izateko, bata bestearengandik ehunka kilometrotara egon baitaitezke. Aipatutako azpiegitura, hainbat geruza teknologikotan banatuko dugu, abstrakzio mailaren arabera.



2.2. irudia. Geruza teknologikoen azpiegitura nagusia.

2.2. irudiak, geruza teknologiko bakoitzaren kokapena eta sortzen diren harremanak islatzen ditu. Urruneko objektuaren proxy lokal baten lana gauzatuko du klase egokitzaile (`stub`) lokalak eta objektu lokalak urruneko eskaera egikaritzen duenean, klase egokitzaileak harrapatzen du eskaera eta komunikazioan jartzen da urruneko zerbitzariarekin. Irudian ikusten den legez, eskaera ez doa zuzenean urruneko objektura baizik eta urruneko beste klase egokitzaile (`skeleton`)

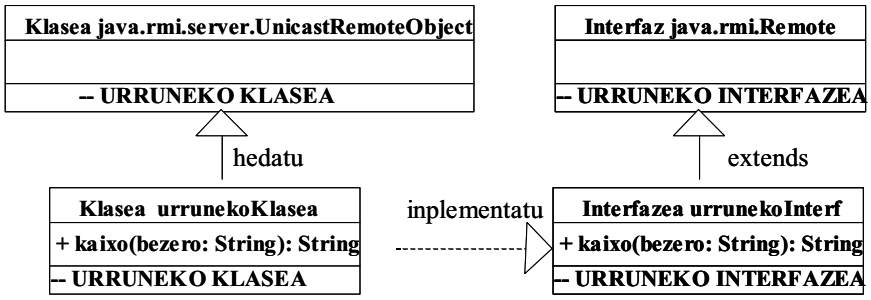
batera. Urruneko klase egokitzailea urruneko objektuaren proxy-a izango da, eta, irudian ikusten den legez, klase egokitzaile lokalarekin harremanetan egongo da. Bai urruneko klase egokitzaileak eta bai klase egokitzaile lokalak, erreferentzia-geruza erabiliko dute komunikazioa eratzeko. Komunikazioaren kudeaketa eta konfigurazioa antolatzeko, garraio-geruza erabiliko du erreferentzia-geruzak. Garraio-geruzak TCP socket-ak erabiliko ditu lehenetsita, urruneko ostalaria eta ostalari lokala harremanetan jartzeko.

Programatzaileak ez ditu zertan klase egokitzaileak programatu, Javaren `rmic` konpiladoreak automatikoki sortzen ditu biak urruneko objektua erabilita. Hortaz, programatzailea aplikazioaren objektuez arduratzen da bakarrik, eta aplikazioa programatzeko orduan, ez du kontuan hartzen beharrezko den azpiegitura teknologikoa kontuan izan behar duela.

Has gaitezen, beraz, RMI teknologia erabilita, bezero/zerbitzari arkitektura izango duen aplikazio banatu bat programatzen.

Erabaki ondoren urruneko objektuak erabiliko ditugula aplikazioan, ezaugarri zehatz batzuk izan beharko dituzte urruneko klaseek programatzean eta nolabaiteko erreferentzia eduki beharko du bezeroak klase horien ezaugarriak jakiteko. Informazio hori interfaze batean edukiko du bezeroak, eta bertan, erabilgarri dituen metodoak adieraziko dira. Horrela, bezeroaren aldea konpilatzeko orduan, nahikoa izango da urruneko interfazea edukitzea, arazorik ez izateko. Urruneko interfazeak `Remote` interfazea hedatu behar du eta, hori eginda, interfazea inplementatzen duen edozein objektu urruneko objektu bihurtuko da automatikoki. Gainera, interfazearen metodoek `RemoteException` salbuespena izan behar dute adierazita `throws` klausulan. RMIk, konexioa galtzen denean edo protokolo-erroreren bat gertatzen denean, salbuespen hori altxatzen du arazoak sortzen direlako urruneko zerbitzarian.

Ikusi ondoren zer egiten den bezeroaren aldean, azter dezagun nola antolatzen den urruneko objektua. Lehenengo eta behin, urruneko interfazea inplementatu behar du urruneko klaseak, eta horretarako, urruneko interfazearen metodo guztiak programatu behar ditu nahitaez interfazearen signatura mantenduta, hau da, parametroen mota eta kopurua errespetatuta. Hurrena, `RemoteServer` klasea hedatu behar du urruneko klaseak. Hori egiteko, normalean `RemoteServer` klasearen `UnicastRemoteObject` azpiklasea erabiltzen da. **2.3. irudia**k esandakoa laburbiltzen du.



2.3. irudia. Bezeroaren eta zerbitzariaren programazio-betekizunak.

Aurreko eragiketekin bezeroaren eta zerbitzariaren aldeko programazio-betekizunak azaldu ditugu baina hori ez da nahikoa aplikazio banatua martxan jartzeko. Hirugarren urratsean, klase egokitzaile lokala eta urruneko klase egokitzailea sortu behar ditugu `rmic` konpiladorea erabilita.

Klase egokitzaileak sortu eta gero, aplikazioa egikaritzen dugu. Aplikazioak bi zati ditu funtzionalki desberdinak, eta ordena jakin bati jarraitu behar zaio egikaritzean. Lehenengo eta behin, zerbitzaria jarri behar dugu martxan eta ezer egin baino lehen urruneko erregistroa hasieratu behar dugu, zerbitzariak ez dezan arazorik izan urruneko objektua erregistratzeko. Urruneko erregistroa hasieratzeko RMI teknologiak `rmiregistry` tresna erabiliko du. Urruneko erregistroak `1099` portua darabil lehenetsita, eta aiduru gelditzen da eskaerak jasotzeko bezeroaren aldetik. Eskaerek izen bat ekarriko dute itsatsita eta urruneko erregistroak izen hori erabiliko du urruneko objektura atzitzeko. Azken prozesu hori automatikoki gerta dadin zerbitzariaren aldean urruneko objektua erregistratu egin behar dugu, `Naming` klasearen `rebind()` metodoa[22] erabilita. Bezeroak, bere aldetik, erregistratutako objektu batera atzitu nahi badu, `Naming` klasearen `lookup()` metodoa erabili behar du. `lookup()` metodoaren sintaxia honako hau da: `Naming.lookup(String <izena>)`.

JNDI (Java Naming and Directory Interface) APIa erabiltzen du Javak izenak eta direktorioak kudeatzeko. Telefono-gidaren antzera, izenak erabiltzen ditu Javak urruneko objektuak bilatzeko eta atzitzeko sistema banatuetan.

Honaino heldu da RMIren aurkezpen orokorra. Gerorako utziko ditugu nola konpontzen diren segurtasun-arazoak eta nola gertatzen den objektuen arteko komunikazioa, balioak bidali eta itzultzen direnean objektu lokalaren eta urruneko objektuaren artean.

Zehaztu ditzagun, beraz, adibide batean, aurkeztu ditugun RMIren ezaugarri teoriko guztiak. Adibidearekin kalkulu-makina generiko bat inplementatu nahi

22. `rebind()` metodoaren sintaxia honako hau izango da:
`Naming.rebind(String <izena>, Object <objektua>)`

dugu, atazak egikarituko dituena. Horretarako, bezeroak `Kalkulu` motako urruneko interfazea inplementatzen duen urruneko objektu bat erabiliko du atazak egikaritzeko. Atazak egikaritzea, beraz, urruneko objektuaren ardura izango da.

Has gaitezen, beraz, urruneko interfazea inplementatzen:

```
package kalkulu;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Kalkulu extends Remote {
    Object egikarituAtaza(Ataza ataza) throws RemoteException;
}
```

Interfazeak, espero genuen bezala, RMIren aurkezpenean hala azaldu baitugu, `Remote` interfazea hedatzen du hura inplementatzen duen objektua urrunetik atzigarria izan dadin. Interfaze horren metodoak `RemoteException` salbuespena altxa dezake `throws` klausulan arazoak sortzen badira objektuen arteko komunikazioan. Gure kasuan metodoa ondo egikaritzen bada, `Object` motako objektu generiko bat itzuliko du. Metodoaren parametro bakarra `Ataza` motako objektu bat izango da. Parametro horretan egikaritu nahi dugun ataza jarriko dugu. Ataza hori ahalik eta generikoena izatea nahi dugunez, beste interfaze bat sortuko dugu atazak tratatzeko:

```
package kalkulu;

import java.io.Serializable;

public interface Ataza extends Serializable {
    Object egikaritu();
}
```

Interfaze hori lokala denez, ez du `Remote` interfazea hedatzen eta bere metodoak ez du zertan `RemoteException` motako salbuespen bat altxatu. Horren ordez, interfazeak, `Serializable` interfazea hedatzen du. Horrela, objektuen serializazio-mekanismoa erabiliko du RMIk objektuak garraiatzeko makina birtualen artean. `Serializable` inplementatzen dugunean, klasea bytez osatutako stream autodeskribatua bihurtzen da, horrela objektuaren kopia bat berreraikitzeko nahikoa izango da stream-a irakurtzea serializazio-mekanismoa aplikatuta.

Konturatu, halaber, `egikarituAtaza()` metodoaren antzera `egikaritu()` metodoak `Object` motako objektu bat itzultzen duela. Hori dela-eta, arazoak ekiditeko, ataza bat egikaritu ondoren oinarrizko datu mota (`int` edo `float` adibidez) bat itzuli nahi badugu, datu mota hori kapsulatu egin beharko dugu eta objektu (`Integer` edo `Float`, adibidez) bihurtu.

Falta dugun gauza bakarra, `Ataza` interfazea inplementatzea da. Gogoratu kalkulu-makina generikoak ataza motak egikaritzeko sortu dugula `Ataza` interfaze hori, eta nahikoa izango dela `egikaritu()` metodoa inplementatzea, ataza bakoitzaren portaera non kokatuko den zehazteko.

Gure kasuan, `Ataza` interfazea inplementatzeko `Ataza1` klasea erabiliko dugu:

```
package bezero;

import kalkulu.*;
import java.util.*;

public class Ataza1 implements Ataza {

    private String izena;
    private Date data;
    private Date data1;

       /**
       * Atazaren eraikitzailea
     */
    public Ataza1(String izena,Date data) {
        this.izena = izena;
         this.data = data;
    }
    /**
     * Interfazearen Metodo nagusia
     */
    public Object egikaritu() {
        Calendar cal = Calendar.getInstance();
        cal.add( Calendar.DAY_OF_YEAR,2 );
        data1 = cal.getTime();
        System.out.println("Ataza : "+izena+"\n");
        System.out.println("Data : "+data.toString()+"\n");
        return data1;
    } }
```

Atazaren izena eta atazaren sortze-data biltegiratzen ditu klasearen metodo eraikitzaileak objektuan, eta `egikaritu()` metodoak gaurko data eskuratzen du eta bi egun gehitzen dizkio, bezeroari emaitza itzuli baino lehen. Ataza egikaritzen den makina birtualean, hau da, zerbitzarian, atazaren izena eta sortu zen eguna eta ordua aterako dira irteera estandarretik.

Atazak tratatzeko azpiegitura prestatu ondoren, kalkulu-makina inplementatzen duen klasea programatu behar dugu. Klase horrek urruneko `Kalkulu` interfazea inplementatuko du eta, aldi berean, urruneko zerbitzarian egikarituko den programa txikia edukiko du kodetuta `main()` metodo nagusian. Has gaitezen errazenetik, gehiena eginda eta esanda baitugu aurreko lerroetan. Gogoratu,

`Kalkulu` interfazea inplementatzeko nahikoa izango zaigula `egikarituAtaza()` metodoa inplementatzea, hori baitzen haren metodo bakarra. Hona hemen nola programatu dugun aipatutako metodoa:

```
public Object egikarituAtaza(Ataza ataza) {
  System.out.println("KalkuluMakina ATAZA !!!!");
  return ataza.egikaritu();
```

Metodoak kalkulu-makinaren eta bezeroaren artean erabiliko den protokolo txiki bat inplementatzen du. Bezeroek, `Ataza` interfaze lokala inplementatzen duen objektu bat pasatuko diote kalkulu-makinari parametro gisa, eta metodoak, `Ataza` inplementatzen duen objektuaren `egikaritu()` metodoa egikaritu eta emaitza itzuliko du.

Hona hemen zerbitzarian martxan jarriko dugun programaren iturburu-kodea:

```
package makinaGenerikoa;

import java.rmi.*;
import java.rmi.server.*;
import kalkulu.*;

public class KalkuluMakina extends UnicastRemoteObject
                           implements Kalkulu
{
    public KalkuluMakina() throws RemoteException {
        super();
    }

    public Object egikarituAtaza(Ataza ataza) {
        System.out.println("KalkuluMakina ATAZA !!!!");
        return ataza.egikaritu();
    }

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        String izena = "//localhost/Kalkulu";
        try {
            Kalkulu makinaGenerikoa = new KalkuluMakina();
            Naming.rebind(izena, makinaGenerikoa);
            System.out.println("KalkuluMakina erregistratuta");
        } catch (Exception e) {
  System.err.println("KalkuluMakina salbuespena: " +
e.getMessage());
            e.printStackTrace();
        }
    } }
```

Dakigunez, `Kalkulu` interfazea inplementatzen du klaseak eta RMIren `UnicastRemoteObject` klasea hedatzen du. `UnicastRemoteObject` klaseak inplementazio egokiak ematen ditu urruneko objektuaren hainbat metodo (`toString()`, `equals()`, `hashCode()`...) arazorik gabe egikaritzeko. Halaber, metodo eraikitzaileak eta metodo estatiko egokiak ditu urruneko objektuak esportatzeko. Horrela, urruneko objektuak eskaerak jaso ditzake arazorik gabe.

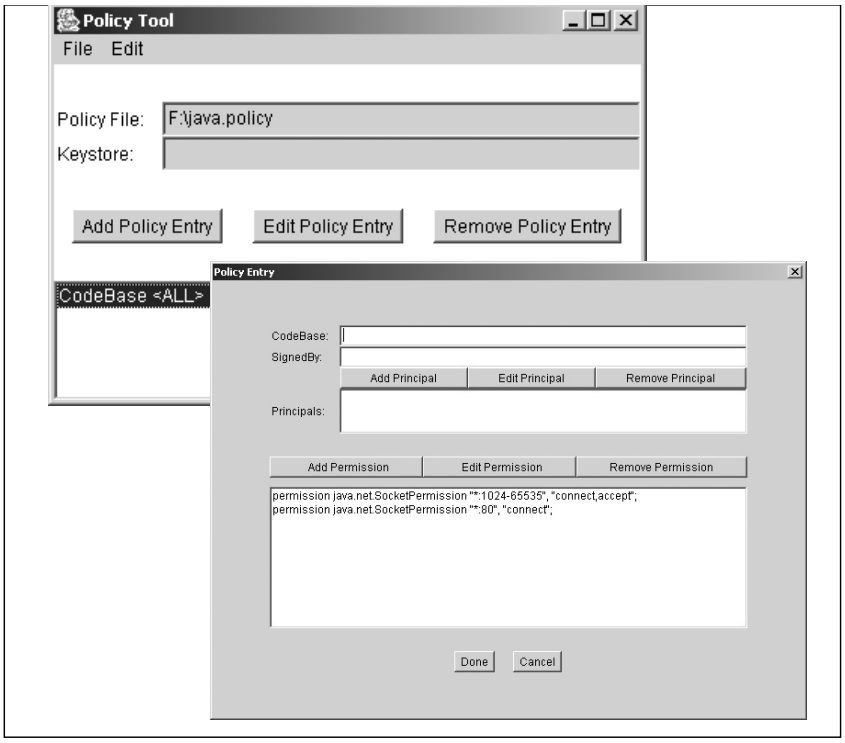
Programatu dugun klasearen metodo eraikitzaileak `super()` metodoa egikaritzen du bakarrik. Horrela, klaseak `UnicastRemoteObject` klasearen eraikitzaileari deitzen dio eta horrek `UnicastRemoteObject` motako objektu esportatua eraikitzen du.

Objektu mota hori prest gelditzen da zerbitzariaren portu anonimotik bezeroaren deiak jasotzeko. `UnicastRemoteObject` klasearen metodo eraikitzaileak `RemoteException` salbuespena erazagutzen duenez `throws` klausulan, gure klaseak ere `RemoteException` salbuespena altxa daitekeela erazagutu egin behar du metodoa egikaritzean.

Horrenbestez, azpiegitura prestatu dugu urruneko objektuentzat zerbitzarian; orain falta dugu, azpiegitura horretaz baliatuta, urruneko objektuak sortzea eta erregistratzea zerbitzarian. Hori egiteko, gure kasuan, `main()` metodoa erabiliko dugu, konturatu ez dela bide bakarra, beste klase bat sor baitaiteke bigarren urrats hori gauzatzeko. Horrela, erabat kapsulatuko genituzke urruneko objektuaren inplementazioa eta zerbitzariaren lanak gauzatuko lituzkeen klasea. Horra hor, luzatzen dizudan ariketa, irakurle.

Besteak beste, urruneko objektua eraiki eta erregistratu egiten du klasearen `main()` metodo nagusiak `rebind()` metodoa erabilita. Prozesuan salbuespenak altxa daitezkeenez, `try-catch` bloke bat erabiliko dugu salbuespenak tratatzeko eta haien jatorria arakatzeko.

Halaber, `main()` metodoak segurtasun-kudeatzailea sortu eta instalatu behar du, atzipenak kontrolatzeko eta kanpotik datorren kodeak sistemaren baliabideak atzitu ez ditzan. Adibidez, appletetan gertatzen zen bezala, segurtasun-kudeatzaileak erabakitzen du ea deskargatu den kodeak atzipena duen ala ez urruneko zerbitzariaren fitxategi-sistemara.



2.4. irudia. **`Policy Tool`** **tresnaren interfaze grafikoa.**

Java 1.2 bertsiotik aurrera segurtasun-eredua aldatu egin da, eta fitxategi bat erabiltzen da segurtasun-politika eraikitzeko urruneko objektuen erregistroarentzat. Nahiz eta posible den beste izen bat jartzea, gehienetan segurtasun-politika jasotzen duen fitxategiari `java.policy` izena ematen zaio. Fitxategian, aplikatu nahi diren segurtasun-politikak txertatzen dira. Gure adibidean[23] nahikoa izango da honelako sarrera txertatzea segurtasun-fitxategian:

```
grant {
Permission java.net.SocketPermission "*:1024-65535",
"connect, accept";
      } ;
```

23. Adibidez, aplikazioak `irteera.txt` fitxategian idatzi nahi badu eta `sarrera.txt` fitxategitik irakurri nahi badu, honako grant sarrera hau behar du:

```
grant{
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "${user.home}/irteera.txt", "write";
    permission java.io.FilePermission "${user.home}/sarrera.txt", "read";
    };
```

Sarrera horrekin, `1024` portutik aurrerako portuetan konexioak gauzatzea onartzen da. Javaren teknologiak `policytool` (**2.4. irudia**) tresna eskaintzen du segurtasun-fitxategi horiek kudeatzeko. Aplikazio bati segurtasun-politika jasotzen duen fitxategi bat erabili behar duela adierazteko, `-D` parametroa erabiltzen da Javaren interpretean. Gure kasuan, zerbitzaria martxan jarri nahi badugu, honela egikarituko dugu interpretea:

```
java -Djava.security.policy=java.policy makinaGenerikoa.KalkuluMakina
```

`java.policy` fitxategia interpretea egikaritzen dugun azpidirektorioan kokatu dugu. Fitxategia beste azpidirektorio batean balego, atzipen-bidea adierazi beharko genuke `-D` parametroan.

Bezero/zerbitzari arkitektura duen aplikazio banatuaren zerbitzariaren aldea programatuta, bezeroaren aldea programatzen bukatuko dugu lana. Esandakoak esanda, gelditzen zaigun lana txikerra da.

Bezeroari gelditzen zaion gauza bakarra urruneko objektua bilatzea da, eta horretarako nahikoa izango du `Naming` klasearen `lookup()` metodoa erabiltzea. Pasatu behar diogun parametro bakarrean zein den urruneko zerbitzaria eta nola deitzen den urruneko objektua adierazten diogu metodoari. Gehienetan urruneko zerbitzaria IP helbide bat izango da, eta bilatu beharreko izenak erregistratuta egon behar du zerbitzariaren urruneko erregistroan. Horregatik, zerbitzarian egikaritu behar den programa aurretik egikaritzen da, bezeroak atzitzeko eta erabiltzeko urruneko objektuak.

Zerbitzarian bezala, segurtasun-kudeatzailea instalatu behar du bezeroak ostalari lokalean, segurtasun arazorik ez izateko urruneko objektuekin. Hori guztia programatu eta gero, hona hemen nola gelditzen den bezeroaren aldean egikarituko den programa nagusia:

```
package bezero;

import java.rmi.*;
import java.math.*;
import kalkulu.*;
import java.util.*;

public class KalkuluProba {

    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            Date data2;
```

```
            Date data1 = new Date();
           String izena = "//localhost/Kalkulu";
           Kalkulu kalkuluO = (Kalkulu) Naming.lookup(izena);


            Calendar cal = Calendar.getInstance();
           cal.add( Calendar.DAY_OF_YEAR,10 );
           data2 = cal.getTime();

            Ataza1 ataza1 = new Ataza1("-Lehen ATAZA--",data1);
            Ataza1 ataza2 = new Ataza1("-Bigarren ATAZA--",data2);

           Date data3 = (Date) (kalkuluO.egikarituAtaza(ataza1));
            Date data4 = (Date) (kalkuluO.egikarituAtaza(ataza2));

           System.out.println(data3.toString());
           System.out.println("\n");
            System.out.println(data4.toString());
       } catch (Exception e) {
      System.err.println("Ataza egikaritzen salbuespena: " +
e.getMessage());
           e.printStackTrace();
       } }}
```

Konturatu ezen, hemen ere, `try-catch` blokea darabilgula programaren salbuespenak kontrolatzeko. Dena ondo badoa eta `lookup()` metodo estatikoak urruneko objektuaren erreferentzia itzultzen badigu, ostalari lokalean egikaritzen dugu urruneko objektuaren metodoa. Horretarako nahikoa zaigu interfazean agertzen den informazioa jakitea, hau da, metodoaren signatura. Programatzeko era horrek erabat erraztuko digu konpilazioa, nahikoa baita urruneko objektuaren interfazea[24] eta ez urruneko objektua, konpilazioa gauzatzeko.

Aplikazioa egikaritu ondoren, ostalari lokalean eta urruneko ostalarian zer ikusiko litzatekeen jasotzen dute **2.5.** eta **2.6. irudi**ek.

```
C:\Java-RMI>java -Djavaa.security.policy=java.policy
bezero.KalkuluProba

         Mon Aug  25  11:30:46  GMT+02:00  2003
         Mon Aug  25  11:30:47  GMT+02:00  2003
```

2.5. irudia. Ostalari lokalean ateratzen den informazioa.

24. Ez dago esan beharrik klase egokitzaile (`stub`) lokala ere behar izango dugula nahitaez, RMIren aurkezpenean esan dugun legez.

```
C:\Java-RMI>java -Djavaa.security.policy=java.policy
makinaGenerikoa.KalkuluMakina

KalkuluMakina erregistratua
KalkuluMakina ATAZA !!!!
Ataza : --Lehen ARAZA----

Data : Sat Aug  23  11:30:43  GMT+02:00  2003

KalkuluMakina ATAZA !!!!
Ataza : --Bigarren ARAZA----

Data : Tue Sep  02  11:30:45  GMT+02:00  2003
```

2.6. irudia. Urruneko ostalarian agertuko den informazioa.

Aurreko orrietan ikusi dugu RMI teknologia erabiltzean, urruneko objektuaren metodoak egikaritzen direla eta informazioa joan-etorrian ibiltzen dela. Askotan informazio hori ez dakigu kopia bat den, eta, hortaz, nahiz eta guk aldatu ostalari lokalean, ez du inongo eraginik izango zerbitzarian eta urruneko objektuan. Beste batzuetan, erreferentzien bidez ari bagara lanean, bezeroaren aldean gauzatzen diren aldaketak islatuta gelditzen dira urruneko zerbitzarian.

Badakigu jakin ere, Javaren teknologia erabilita idatzitako aplikazio arruntetan metodo bati dei egiten diogunean parametroetan, oinarrizko datu mota aldagaiak erabilita, balioz egiten dela parametroen arteko lotura, hau da, metodoak aldagaiaren kopia jasotzen du parametroan. Metodoaren barnean aldagaiaren kopiaren balioa aldatzen bada, aldaketak ez du inongo eraginik izango jatorrizko aldagaian. Aldiz, parametroa objektu bat bada, objektuaren erreferentzia pasatzen da, eta, beraz, objektuaren balioak alda daitezke objektuaren gainean ari baikara lanean.

RMIn berriro, gauzak zertxobait aldrebesten dira. Egikaritu ondoren objektu arrunt bat itzultzen badu urruneko objektuaren metodo batek, objektuaren kopia bat serializatu egiten da eta bezeroari bidaltzen zaio, hau da, balio bidez itzultzen zaio kopia. Objektuaren balioa aldatzen bada bezeroaren aldean, ez du inongo eraginik izango zerbitzariaren jatorrizko urruneko objektuan.

Aldiz, egikaritu ondoren, urruneko objektu bat itzultzen badu urruneko objektuaren metodo batek, gauzak beste era batera irakurtzen dira, hasteko ez da objektuaren kopia serializatua itzultzen, erreferentzia baizik, hau da, objektuaren balioa aldatzen bada bezeroaren aldean, jatorrizko urruneko objektuarena ere aldatzen ari gara aldi berean.

Beraz, kontu handia eduki behar dugu urruneko objektuen metodoekin ari garenean lanean. Batzuetan, urruneko objektua aldatzen ari garela pentsa dezakegu eta benetan aldatu duguna bere kopia izan da eta ez du inongo eraginik izango urruneko objektuan.

Adibide batek lagunduko digu esandakoa argitzen. Aplikazio banatu bat inplementatuko du adibideak, RMI teknologia erabilita.

Aurreko lerroetan esan dugunez, joan-etorrian ibiltzen diren urruneko objektuak mota askotakoak izan daitezke eta horrek eragina izango du objektu horiek eguneratzean. Adibidean ikusiko dugu, urruneko objektua arrunta bada, hau da, ez badu `Remote` interfazea inplementatzen, bezeroak objektuaren kopia bat jasoko duela eta bezeroaren aldean egiten diren aldaketak ez direla islatuko urruneko zerbitzarian. Aldiz, urruneko objektuak `Remote` interfazea inplementatzen badu, aldaketek iraunkortasun bat izango dute eta islatu egingo dira zerbitzariaren urruneko objektuan. Aipatutako bi objektu motak, aplikazioaren urruneko interfazean erazagutuko ditugu:

```
import java.rmi.*;
import java.util.*;

public interface ONagusiaI extends Remote {

  Date lortuData() throws RemoteException;

  LangileaI lortuLangilea() throws RemoteException;
  }
```

Interfazearen lehen metodoak, `Date` motako objektua itzuliko du eta datak tratatzeko erabiliko dugu. Objektu hori izango da `Remote` interfazea inplementatuko ez duena. Bigarren metodoak, `Langilea` motako objektu bat itzuliko du eta langilearen informazioa kudeatzeko erabiliko dugu. Konturatu erazagupenean metodoak `LangileaI` itzultzen duela, hau da, `Langilea` klaseak inplementatuko duen interfazea. Geroago ikusiko dugun legez, `Remote` interfazea hedatzen du interfaze horrek eta inplementatzen duen objektuak urruneko objektu baten ezaugarriak izango ditu. Langilearen informazioa urruneko objektu baten bidez kudeatuko dugunez, objektuaren erreferentzia erabiliko dugu, eta ez bere kopia, horrela, edozein aldaketak eragina izango du zerbitzariaren urruneko objektuan.

`ONagusiaI` urruneko interfaze nagusiaren inplementazioa, `ONagusia` klaseak egingo du. Klaseak, espero genuen bezala, interfazearen `lortuData()` eta `lortuLangilea()` metodoak inplementatzen ditu. Gauzak ondo egiteko, lehenengo eta behin, metodo eraikitzaileak langilea sortuko du. Konturatu hasieran sortu dugun langilearen soldata 1000 dela eta hori izango dela aldatuko dugun langilearen informazioa. Bere poztasunerako soldata erruz igoko diogu. Hona hemen nola inplementatu dugun aipatutako urruneko interfazea:

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
```

```
public class ONagusia extends UnicastRemoteObject implements
ONagusiaI {

  private Date data = new Date();
  private LangileaI urrunekoObjektua;

  // Metodo eraikitzailea
  public ONagusia() throws RemoteException{
    urrunekoObjektua = new Langilea("Koldo Sarriugarte",1000 );
    }

  // Objektuaren data itzultzen duen metodoa
  public Date lortuData() throws RemoteException {
    return( data );
    }

  // Urruneko objektua itzultzen duen metodoa
  public LangileaI lortuLangilea() throws RemoteException {
    return( urrunekoObjektua );
    }
  }
```

Langilearen informazioa urruneko objektu bihurtzeko, `LangileaI` interfazea erabiliko dugu. Interfazeak `Remote` interfazea hedatzen du eta bere metodo guztiek `RemoteException` motako salbuespena altxa dezakete. Inplementatuko diren metodoak metodo atzitzaile arruntak izango dira eta langilearen aldagai pribatuak kudeatzeko erabiliko ditugu:

```
import java.rmi.*;

public interface LangileaI extends Remote{

  void ipinIzena( String izena ) throws RemoteException;
  String lortuIzena() throws RemoteException;

  void ipinSoldata( int soldata ) throws RemoteException;
  int lortuSoldata() throws RemoteException;
  }
```

`Langilea` klaseak aurreko interfazea inplementatzen du eta, zeharka, baita `Remote` interfazea ere. Horregatik, mota horretako objektuari aldaketa bat egiten bazaio bezeroaren aldean, aldaketa horrek urruneko objektu hori aldatuko du eta eguneratuta utziko du urruneko zerbitzarian. Bestela, `Langilea` klasearen iturburuak ez du misterio handirik:

```
import java.rmi.*;
import java.rmi.server.*;

public class Langilea extends UnicastRemoteObject implements
LangileaI {
```

```
  private String izena;
  private int soldata;

  // Metodo Eraikitzailea
  public Langilea( String izena,int soldata ) throws
RemoteException{
    // Izena eta soldata esleitzen diogu langileari
     this.izena = izena;
     this.soldata = soldata;
    }

public void ipinIzena( String izena ) throws RemoteException {
    this.izena = izena;
    }
  public String lortuIzena() throws RemoteException{
    return( izena );
    }

  public void ipinSoldata( int soldata ) throws RemoteException {
    this.soldata = soldata;
    }
  public int lortuSoldata() throws RemoteException{
    return( soldata );
    }
  }
```

Aplikazioaren urruneko objektuen azpiegitura prestatu eta gero, bezeroa eta zerbitzaria martxan jarriko ditugu. Zerbitzaria martxan jartzeko bi gauza egin behar ditugu: urruneko objektu nagusia eraiki eta urruneko erregistroan erregistratu eskuragarria izan dadin beste makina birtual batetik:

```
import java.rmi.*;
import java.rmi.server.*;

public class Zerbitzaria {

  public static void main( String args[] ) {
    // Segurtasun kontrolatzailea instalatzen da
    if( System.getSecurityManager() == null ) {
      System.setSecurityManager( new RMISecurityManager() );
      }

    try {
      // Urruneko objektu nagusia eraikitzen da
      ONagusia urrunekoObjektua = new ONagusia();

      Naming.rebind( "UrrunekoObjektua",urrunekoObjektua );

      System.out.println( "Urruneko Objektua prest dago mezuak
jasotzeko" );
```

```
    }catch( Exception e ) {
      e.printStackTrace();
     } } }
```

Bestalde, urruneko objektu nagusia bilatuko du bezeroak eta bere metodoak egikarituko ditu. Metodoak egikarituz, pantailan (**2.7. irudia**) ikusiko dugu nola egikaritzen den programaren agindu-fluxua. Batetik, `Date` motako objektua aldatu egingo du programak eta aldaketak eragina izango du ostalari lokalean baina bakarrik hor. Aztertuko bagenu ea urruneko objektua aldatu den zerbitzarian, konturatuko ginateke hasierako balioarekin jarraitzen duela.

```
C:\Java-RMI>java -Djavaa.security.policy=java.policy Bezeroa
Bezeroa egikaritzen
Data eskatzen diogu zerbitzariari
Urruneko objektuaren data : Sat Aug 23 12:47:17 GMT+02:00 2003
Data 10 egun igotzen dugu
Orain data (gaur+10) berria:  Tue Sep 02 12:48:53 GMT+02:00 2003
Data eskatzen diogu zerbitzariari
Urruneko objektuaren data orain : Sat Aug 23 12:47:17 GMT+02:00 2003
 ---Langilearekin hasten gara lanean:
Langilearen izena: Koldo Sarriugarte
Langilearen soldata: 1000
Langilearen soldata aldatzen dugu
Langilearen soldata berria: 5000
Zerbitzariari berriro eskatzen diogu Langilea
Urruneko objektuaren soldata orain: 5000
```

2.7. irudia. Adibidearen emaitzak irteera estandarrean.

Bestetik, langilearen soldata aldatzen badugu ostalari lokalean, orain bai, eragina izango du aldaketak zerbitzariaren urruneko objektuan, **2.7. irudi**ko azken lerroak adierazten duen legez. Irudian ikusten diren irteera guztiak lortzeko honako programa hau erabili dugu:

```
import java.rmi.*;
import java.util.*;

public class Bezeroa {

  public static void main( String[] args ) {
    System.out.println( "\nBezeroa egikaritzen..." );

    String helbidea = "rmi://localhost/";
    try {
      ONagusiaI ObjektuRef =
        (ONagusiaI)Naming.lookup( helbidea+"UrrunekoObjektua" );
```

```
      System.out.println( "Data eskatzen diogu zerbitzariari" );
      Date data = ObjektuRef.lortuData();
      System.out.println( "Urruneko objektuaren data : " +
        data.toString() );

      System.out.println( "Data 10 egun igotzen dugu" );
      Calendar cal = Calendar.getInstance();
      cal.add( Calendar.DAY_OF_YEAR,10 );
      data = cal.getTime();
      System.out.println( "Orain data(gaur+10) berria: " +
        data.toString() );

      System.out.println( "Data eskatzen diogu zerbitzariari" );
      data = ObjektuRef.lortuData();
      System.out.println( "Urruneko objektuaren data orain : " +
        data.toString() );

     System.out.println( "\n --Langilearekin hasten gara lanean: "
);
      LangileaI objektu = ObjektuRef.lortuLangilea();
      System.out.println( "Langilearen izena: " +
        objektu.lortuIzena()+"\n" );
      System.out.println( "Langilearen soldata: " +
        objektu.lortuSoldata()+"\n" );

      System.out.println( "Langilearen soldata aldatzen dugu" );
      objektu.ipinSoldata( 5000 );
      System.out.println( "Langilearen soldata berria: " +
        objektu.lortuSoldata() );

      System.out.println( "Zerbitzariari berriro eskatzen diogu
Langilea" );
      objektu = ObjektuRef.lortuLangilea();
      System.out.println( "Urruneko objektuaren soldata orain: " +
        objektu.lortuSoldata() );
    }catch( Exception e ) {
      e.printStackTrace();
      }
    System.exit( 0 );
    }
 }
```

2.2. LABURPENA

Lehenengo atalburuan garapen monolitikoaren nondik norakoak azaldu dira Javaren AWT/SWING eta JBDC baliabideak aurkeztuta. Adibideetan ikusi den bezala, aplikazioak, ordenagailu bakarrean egikaritzen ziren inongo urruneko komunikaziorik sortu gabe. Dena zegoen hurbil eta eskura eta aplikazioaren arkitekturak ez zuen buruhauste handirik ematen. Gaur egun, alabaina, softwarea,

egitura banatuetan antolatzen da bai programazio mailan, bai arkitektura mailan, malgutasuna areagotuz aplikazioaren bizi-zikloari.

Oinarrizko bezero/zerbitzari arkitektura inplementatu nahi denean Javaren teknologia erabilita eraikitako aplikazioan, ez dugu asko urrundu beharrik objektuetara zuzendutako paradigma estandarretik. Javaren teknologiaren RMI APIa erabilita, klase eta interfaze arruntak erabiltzen dira arkitektura banatu horiek eraikitzeko. Programatzaileak nahikoa du oinarrizko arau batzuk erabiltzea, urruneko objektuak erregistratzeko eta bilatzeko arazorik gabe. Interfazeen bidez, bilaketak eta atzipenak gardenak dira programatzailearentzat, iturburu-kodea ez baita *zikintzen* kode arrotzarekin eta intrusiboarekin.

2.3. OINARRIZKO BIBLIOGRAFIA

Caballe, S. (2007): *Aplicaciones distribuidas en Java con tecnologia RMI*, Delta Publicaciones.

Froufe, A.(2005): *Java 2. Manual de usuario y tutorial*, RA-MA.

Grosso, W.(2001): *Java RMI*, O´Reilly.

McNiff K. eta Pitt. E. (2001): *Java RMI: The Remote Method Invocation Guide*, Addison Wesley.

Öberg, R. (2001): *Mastering RMI: Developing Enterprise Applications in Java and EJB*, John Wiley & Sons.

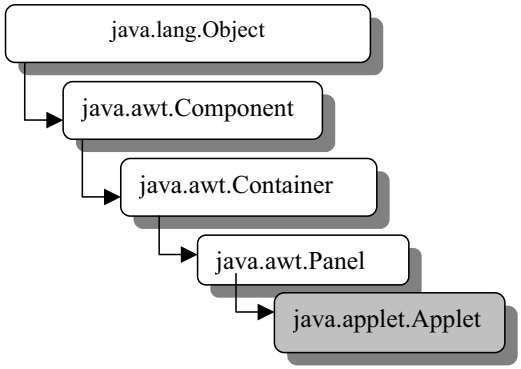
2.4. WEB GUNE INTERESGARRIAK

Zer dago bertan	Web gunea
RMIren tutoretza (I)	`http://java.sun.com/docs/books/tutorial/rmi/index.html`
RMIren tutoretza (II)	`http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp`
Sarrera RMI APIra	`http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=rmi2`
RMIren adibideak	`http://www.exampledepot.com/egs/java.rmi/pkg.html`
Sarrera RMI APIra	`http://programacion.com/tutorial/rmi/`

3. Hirugarren hurbilketa: hiru mailako arkitektura

3.1. APPLET/HTML

Applet bat aplikazio txiki bat da Internet-zerbitzari batek kudeatzen duena eta bezeroaren ordenagailuan egikaritzen dena. Horretarako, programa txiki hau, saretik garraiatzen da eta automatikoki instalatzen da bezeroaren ordenagailuan. Prozesu guztia ondo gauzatzeko, HTML kodea duen fitxategi batean txertatzen da appleta eta web nabigatzaile egokia erabiliz egikaritu egiten du bezeroak bere ordenagailuan. Txertatutako Java kodea ulertu eta egikaritu egiten du nabigatzailearen Java makina birtualak (Java Virtual Machine-JVM). Hala eginez gero, asko aberasten dira HTML erabilita kodetutako web dokumentuak, zeren erabiltzailearen interfaze grafiko bat sar baitaiteke HTML orrian eta egikaritzen jarraitu bezeroaren ordenagailuan.



3.1. irudia. Applet osagaiaren klase-hierarkia.

Urrunetik ekarritako Java programak matxurarik sortu ez dezan (bezeroaren fitxategiak irakurri, birus bat txertatu...) makina birtualak babestu egiten du bezeroaren ordenagailua.

Aitzinean esan dugun bezala, appleta aplikazio txiki bat da eta hura programatzea aplikazio arrunta eraikitzea bezala izango da programatzailearentzat. Dena ondo joan dadin, `Applet` (**3.1 irudia**) klasea hedatu behar du appleta programatuko duen klaseak eta metodo berezi batzuk (`init()`, `start()`, `paint()`...)

birprogramatu behar ditu programatzaileak gainkargatuz, haien portaera gure beharretara egokitzeko.

Azter dezagun azaldutako guztia adibide txiki batekin:

```
import java.applet.*;
import java.awt.*;

public class GureAppleta extends Applet
{ String izena;
  public void init()
   {
   izena=new String("Gure appleta hasieratzen");
   }
   public void paint(Graphics g)
   {
   g.drawString(izena,10,10);
   }  }
```

Adibidean aurretik esan dugun guztia gauzatu dugu: `Applet` klasea hedatu eta `init()` eta `paint()` metodoak birprogramatu. Gure lehen appleta prest dago egikaritzeko, baina horretarako, beste hiru urrats gauzatu beharko ditugu aurretik:

1. Sortu dugun programaren kodea (`aplikazioa.java`) konpilatu eta `.class` fitxategia sortu.
2. Sortu HTML kodea, eraiki dugun `.class` fitxategiari deitzeko.
3. Bukatzeko, appleta egikaritu eta ikuskatu Javarekin bateragarria den web nabigatzaile bat erabilita. Hasierako uneetan, appleta programatzen eta arazten ari garenean, komeni izaten da programa azkar probatzea. Horretarako Javaren teknologiak `appletviewer` agindua eskaintzen digu. Nahikoa da, `appletviewer` aginduari, HTML fitxategia pasatzea parametro gisa appleta ikuskatzeko egikaritzen, sortzen duen leiho grafikoan.

Adibideko Java kodea ondo konpilatu badugu eta errorerik ez bada agertu `.class` fitxategia sortua izango dugu. Ondoren, HTML fitxategia eraikitzen da:

```
<html>
        <head>
       </head>
  <body>
<applet code="GureAppleta.class" width="300" height="300">
</applet>
  </body>
</html>
```

Fitxategian, ikusten den legez, `APPLET` etiketan adierazi behar ditugu appletak izango dituen neurriak (`WIDTH` eta `HEIGHT`) eta erabili behar duen `.class` fitxategia (`CODE`). **3.3. taula**n `APPLET` etiketan txerta daitezkeen atributuak azaltzen dira.



3.2. irudia. Adibidearen irteera grafikoa.

Atributua	Deskripzioa	Nahitaezkoa
`CODE`	Klasearen izena jartzen den tokia	Bai
`WIDTH`	Appletaren zabalera pixel-etan neurtua	Bai
`HEIGHT`	Appletaren altuera pixel-etan neurtua	Bai
`CODEBASE`	`.class` fitxategia bilatzeko helbidea	Ez
`NAME`	Appletaren izena	Ez
`ALIGN`	Appletaren alineazio mota HTML orrian	Ez
`VSPACE`	Appletaren marjina bertikala	Ez
`HSPACE`	Appletaren marjina horizontala	Ez

3.3. taula. APPLET etiketaren atributuak.

Batzuetan, komeni izaten da balioak appletari pasatzea, egikaritze-garaian erabiltzeko. Horretarako, `PARAM` etiketa erabiliko dugu:

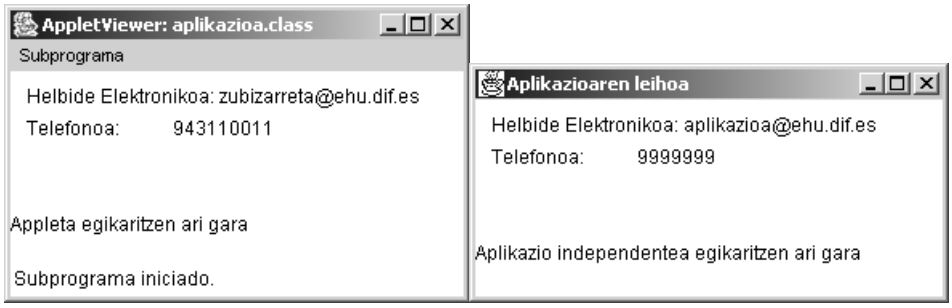
```
<PARAM NAME = "........."   VALUE = "........." >
```

Etiketa hori erabilita izena-balioa bikoteak sortzen ditugu eta nahi adina `PARAM` etiketa txertatu, HTML orrian. `getParameter()` metodoa erabilita balio horiek erraz eskuratzen ditu programak eta exekuzio-fluxuan txertatu egikaritze-garaian.

Applet bat eraikitzen denean nahitaezkoa da jakitea nola gauzatzen den appletaren bizi-zikloa. Appleta egikaritzen hasten denean, ataza zehatz batzuk gauzatzen ditu nabigatzaileak:

1. Appletari memoria esleitzen dio.
2. Appletaren `init()` metodoa egikaritzen du. Metodo horri behin deitzen zaio: appleta kargatzen denean.
3. Appletaren `start()` metodoari dei egiten dio egikaritzeko. Appleta txertatuta dagoen HTML orria aktibatzen den bakoitzean deitzen dio web nabigatzaileak metodo horri.
4. Appletaren `paint()` metodoa egikaritzen du. Leiho grafikoa marraztu behar duen bakoitzean, `paint(Graphics g)` metodoari deitzen dio appletak.
5. Erabiltzaileak, appleta ikusezina uzten duenean, appletaren `stop()` metodoari deitzen dio nabigatzaileak. Appleta txertatuta dagoen HTML orria desaktibatzen den bakoitzean, deitzen dio web nabigatzaileak metodo horri.

Irakurlea, honezkero konturatuko zinenez, hemen jarri ditugun metodoak gehiago dira adibidean agertzen direnak baino. Nola liteke hori? Gogoratu irakurle applet guztiak `Applet` klasetik eratortzen direla eta appletak egiten duen gauza bakarra aipatutako metodoak gainkargatzea izaten dela. Horratx, eskatzen zen azalpena.



3.4. irudia. Adibidearen bi irteera grafikoak.

Appletaren kodea aztertzen badugu, argi geratzen da applet bat ezin dugula aplikazio independente gisa egikaritu. Appletaren Java kodean `main()` metodoa egikaritzen duen klasea ez da inondik agertzen eta `java` interpretea erabilita appleta egikaritzen saiatzen bagara errorea azalduko zaigu aipatutako gabeziaz ohartarazteko. Beraz, applet bat aplikazio independente gisa egikaritu nahi badugu, aldaketa batzuk gauzatu beharko ditugu appletaren kodean:

1. Appletari `main()` metodoa gehitu.
2. `main()` metodoaren barnean appletaren instantzia sortu.
3. Leiho bat (`Frame`) sortu appletak izango dituen edukiontziak jasotzeko.

4. Sortu dugun leihoa (`Frame`) `Window` klasearen eratorpena denez, ez dugu inongo arazorik izango appleta (`Panel`) gehitzeko edukiontzi (`Component`) bat besterik ez delako.
5. Aplikazioaren kontrolpean jartzen ditugu appletaren metodoak (`init()`, `start()`, `stop()`) aplikaziotik kontrolatzeko appletaren bizi-zikloa.
6. Bukatzeko, leihoaren neurriak zehazten ditugu eta ikusgaitasuna aktibatzen dugu.

Hori guztia argitzeko, klase bat sortuko dugu, `Applet` klasea hedatzen duena eta, aldi berean, aplikazio independente eran ere funtziona dezakeena. Bi batean, aplikazio independente bat eta applet bat programa berean. Hau da hau, pagotxa !!!

```
import java.applet.*;
import java.awt.*;

public class aplikazioa extends Applet
{ String informazioa;
  private String helbideElektronikoa;
  private long telefonoa;

 public void init()
 {
  informazioa=new String("Appleta egikaritzen ari gara");
  helbideElektronikoa= getParameter("E_HELBIDEA");
  if (helbideElektronikoa  == null) helbideElektronikoa = "@@@@@@@@";
    String telef = getParameter("TELEFONOA");
    if (telef == null) telef = "000000";
    telefonoa= Long.parseLong(telef);

 }
 public void init2()
 {
   informazioa=new String("Aplikazio independentea egikaritzen ari
gara");

  helbideElektronikoa= "aplikazioa@ehu.dif.es";
  telefonoa = 9999999;
 }
 public void start()
 {
 /* Aplikazioaren hasierako metodoak*/
 }
 public void stop()
 {
 /* Aplikazioaren bukaerako metodoak*/
 }
 public void paint(Graphics g)
 {
```

```
   g.drawString(informazioa,0,100);

    g.drawString("Helbide Elektronikoa: " + helbideElektronikoa, 10,
20);
   g.drawString("Telefonoa:           " + telefonoa, 10, 40);

 }
 public static void main(String args[])
 {
  /* Applet bat den aplikazio objektu bat instantziatzen dugu */
  aplikazioa gureAplika= new aplikazioa();
  /* Leiho bat sortzen dugu*/
  Frame leihoa=new Frame("Aplikazioaren leihoa");
  /* Leihoari appleta gehitzen diogu */
  leihoa.add(gureAplika);
  /* Aplikazioaren init2() metodoa egikaritzen dugu */
  gureAplika.init2();
  /*Leihoaren neurriak finkatzen ditugu
       eta ikuskatu egiten dugu*/
  leihoa.setSize(300,150);
  leihoa.setVisible(true);
 } }
```

3.2. JSP/Servlet

3.2.1. Servletak

3.2.1.1. Sarrera

Aurreko orrialdeetan argi gelditu da ezen, appletak, web nabigatzailearen testuinguruan egikaritzen direla; servletak, aldiz, web zerbitzarian edota Java2EE (Enterprise Edition) edukiontzian egikarituko dira. Servleten ohiko erabileran, web orriak, era dinamikoan eraikitzen dira.

Servletak Javaren teknologiaren `javax.servlet.Servlet` interfazea inplementatzen du. Interfaze hori inplementatzeko, servlet generikoa eraiki daiteke `javax.servlet.GenericServlet` klasea erabilita. Horrela eraikitzen den servleta, independentea da protokoloarekiko. Eraiki nahi den web aplikazioak HTTP protokoloa erabiliko badu, HTTP motako servleta eraikiko da `javax.servlet.http.HttpServlet` klasea hedatuta. Egoera horretan, web bezeroak bidaltzen dituen galderak, `HttpServletRequest` eta `HttpServletResponse` motako objektuak erabilita kudeatzen dira. `HttpServletRequest` motako objektuak, web bezeroak zerbitzariari bidaltzen dizkion datuak kapsulatzen ditu, eta `HttpServletResponse` motako objektuak, zerbitzariak web bezeroari bidaltzen dizkionak kapsulatzen ditu.

Hurrengo orrialdeetan, servleten bizi-zikloaren eta barne-portaeraren nondik norakoak aztertuko ditugu.

3.2.1.2. Servleten bizi-zikloa

Appletek bezala, servletek ere, bizi-ziklo aurremugatua dute egikaritzean. Urrats guztiak zehatz-mehatz azaldu baino lehen, esan dezagun bizi-zikloak hiru fase nagusi izango dituela: hasieraketa, eskaeren prozesua eta bukaera.

Hasieraketa fasea hiru eratara gauzatu daiteke:

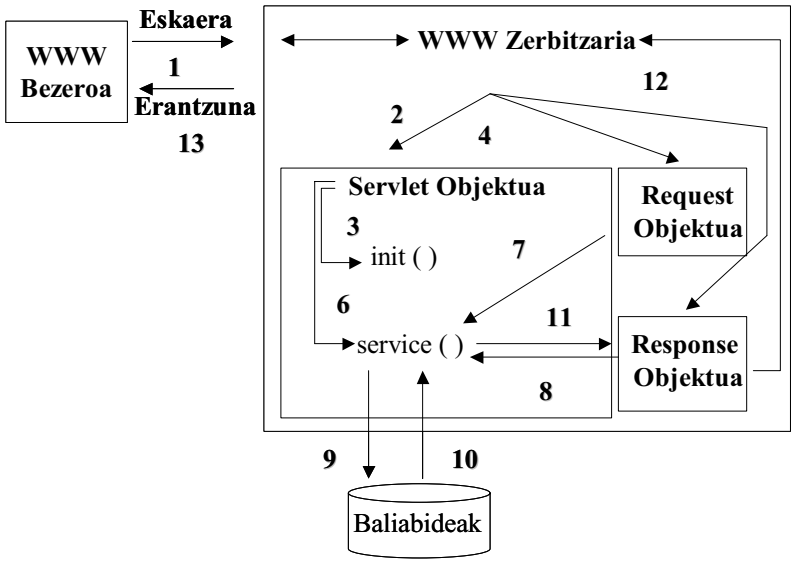
- Zerbitzaria hala konfiguratu badugu, automatikoki hasiera daiteke servleta, zerbitzaria martxan jarri ondoren.
- Berez, egoerarik arruntena bigarren hau da. Hemen, lehen eskaera egin ondoren, bezeroak hasieratzen du servleta.
- Hirugarren egoeran servleta berriro kargatzen da. Servleta aldatu badugu, hobetzeko edo erroreren bat konpontzeko, berriro konpilatu behar dugu. Egoera horretan zerbitzaria konturatu egiten da servleta aldatu dela, eta bertsio berria kargatzen du memorian.

Servleta kargatu ondoren, bere instantzia eraikitzen du zerbitzariak eta `init()` metodoari deitzen dio hasieraketak gauzatzeko.

Hasieraketa eginda bezeroaren eskaeren zain gelditzen da zerbitzaria. Eskaera bat jasotzen duenean, `Request` objektu bat eta `Response` objektu bat sortzen ditu zerbitzariak.

Bezeroarengandik jasotako eskaera, `service()` metodoak interpretatzen du, bidalitako eskaera HTTP metodotzat hartuta. HTTP metodoek ez dute zer ikusirik Javaren programazio-lengoaiaren metodoekin, izen bera erabiltzen da baina ez dute interpretazio semantiko bera. HTTP metodoak bidaltzen dio zerbitzariari mezu bat erantzun mota bat eraikitzeko. Zortzi HTTP metodo daude: `GET`, `POST`, `HEAD`, `PUT`, `DELETE`, `TRACE`, `CONNECT` eta `OPTIONS`. Geroxeago ikusiko dugun bezala, erabilienak `GET` eta `POST` metodoak dira.

`service()` metodoak, interpretatu (**3.5. irudia**) ondoren jasotako eskaera zein HTTP metodo den, `Request` objektua erabiltzen du eskaeraren datuak jasotzeko. Ondoren, eskaera prozesatzen du, eta `Response` objektuaren metodoak erabiltzen ditu informazioa bezeroari bidaltzeko. **3.5. irudia**k azaldutako prozesu guztia laburbiltzen du:



3.5. irudia. Servlet osagaiaren prozesu nagusia.

3.2.1.3. Servleten barne-portaeraren azterketa

Normalean HTTP protokoloa erabiltzen denez bezeroaren nabigatzailearen eta zerbitzariaren arteko komunikazioa kudeatzeko, `HttpServlet` klasetik eratortzen dira servletak zuzenean, eta ez `GenericServlet` klasetik. Hori dela-eta, servlet gehienetan ez da `service()` metodoa programatzen, eta horren ordez, `HttpServlet` klasearen `doGet()` edota `doPost()`[25] metodoak birdefinitzen dira.

Servleta suntsitu nahi duenean, `destroy()` metodoa erabiliko du zerbitzariak. Bukaera-fase horretan, servletak kontsumitzen dituen baliabide guztiak askatzen ditu zerbitzariak `destroy()` metodoa erabilita. Aurreko metodoekin gertatzen zen bezala, `destroy()` metodoa birdefini dezakegu gure nahietara egokituz erabiltzailearen azken eragiketak gauzatzeko, servleta suntsitu baino lehen.

Eraikiko dugun lehen servletak, appletekin egin genuen antzera, `Kaixo Patxi !!!` mezu ezaguna bistaratuko du nabigatzailean ondo egikaritzen bada. Servletak, lehen adibide honetan, ez du inongo elkarrizketarik izango erabiltzailearekin. Horretarako, bezero/zerbitzari arkitektura duen programa txiki bat eraiki behar dugu.

Bezeroaren aldetik, zerbitzu bat eskatuko dio erabiltzaileak servletari, eskaera bat eginez, eta zerbitzariaren aldetik, servleta egikaritu egingo da, erantzun bat sortzeko. Hori guztia egiteko, honako kode hau programatzen da:

25. `doGet()` eta `doPost()` jartzen ditugu bakarrik, erabilienak direlako, baina ez bakarrak. Gogoratu `doDelete()`, `doOptions()`, `doPost()` eta `doTrue()` ere birdefini daitezkeela.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class KaixoPatxiServlet extends HttpServlet {

  protected void doGet(
    HttpServletRequest req,HttpServletResponse res )
    throws ServletException,IOException {

    res.setContentType( "text/html" );
    PrintWriter out = res.getWriter();
    out.println(
      "<HTML><HEAD><TITLE>KAIXO PATXI!</TITLE>" +
                "</HEAD><BODY>Kaixo Patxi!</BODY></HTML>" );
    out.close();
    }

  public String getServletInfo() {
    return "KaixoPatxiServlet, Lehen Servleta";
    }
  }
```

Hasierako lerroetan pakete berezi batzuk inportatzen ditugu servletak arazorik izan ez dezan erabiliko dituen klaseekin:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Beharrezko paketeak inportatu eta gero, servleta joango den klasea erazagutuko dugu. Horretarako, `HttpServlet` klasea hedatuko dugu, HTTP protokoloa erabiliko baita, bezero/zerbitzari arkitektura duen aplikazioa eraikitzeko.

Eraikitzen ari garen servletari funtzionalitate txiki bat emateko, `doGet()` metodoa programatuko dugu. Metodo hori, berez, `HttpServlet` klasearen metodo bat da guk gainkargatu egingo duguna, metodoa, gure nahietara egokitzeko. Bi salbuespen jasotzen ditu metodoak `HttpServlet` klasetik: `ServletException` eta `IOException`.

Servletera heltzen diren eskaerak tratatuko ditu `doGet()` metodoak eta horretarako bi parametro erabiliko ditu. Lehen parametroa, `HttpServletRequest` motakoa izango da eta eskaeraren nondik norakoak zehaztuko ditu. Bezeroak zerbitzariari bidaliko dizkion datuak kapsulatzeko erabiliko da. Zerbitzariak bezeroari datuak bidali behar dizkionean ere kapsulatu egingo dira eta horretarako `HttpServletResponse` motako objektua erabiliko da, hain zuzen, `doGet()` metodoaren bigarren parametroa:

```
protected void doGet( HttpServletRequest req,HttpServletResponse res )
          throws ServletException,IOException
```

`HttpServletResponse` motako objektu batekin bi eratara itzul diezazkioke datuak bezeroari zerbitzaritik. Lehenengoan, `getWriter()` metodoa erabilita, eta bigarrenean, `getOutputStream()` metodoa erabilita. Lehenengo metodoa testua itzultzeko erabiltzen da, eta bigarrena, datu bitarrak.

Zehazturik nor arduratuko den servletaren eskaeraz eta nor erantzunaz, has gaitezen servletak bidaliko duen erantzuna eratzen. Lehenengo eragiketarekin, erantzunaren mota zein izango den adieraziko diogu bezeroari, horretarako `setContentType()` metodoan "`text/html`" balioa pasatzen diogu. Balio horrekin, bezeroari adierazten diogu ezen servletaren erantzunak HTML kodearen formatua izango duela. Zehaztu ondoren bidaliko den informazioa zein motatakoa izango den, has gaitezen informazioa bidaltzen. Horretarako, irteera-kanal bat eraikitzen da bidali nahi dugun mezuaren testua bidaltzeko. Kanala ireki ondoren, informazioa bidaltzen diogu bezeroari HTML formatuan, nabigatzaileak jaso eta interpreta dezan. Ondo joan badira gauzak, nabigatzailean ikusiko da HTML orria bidali dugun informazioarekin. Informazioa bidali ondoren, irteera-kanala itxi egiten dugu, `close()` metodoa erabilita.

Azaldu dugun servleta egikaritu nahi badugu, gure ordenagailuan web zerbitzari bat izan behar dugu instalatuta. Guk gure adibidea egikaritzeko, JSDK (Java Servlet Development Kit) erabiliko dugu, irakurleak beste zerbitzari bat erabiltzen badu, zerbitzari horren dokumentazioa aztertu beharko du zerbitzaria martxan jartzeko.

Zerbitzariak arazorik izan ez dezan, servletak aurkitzeko edo ostalariaren izena zein izango den jakiteko, `default.cfg` fitxategia egokitzen dugu lehendabizi. Hori eginda, azpidirektorioen zuhaitza eraikitzen dugu, servletak direktorio egokian kokatzeko. Guk honako konfigurazio-fitxategia erabili dugu:

```
server.port=8080
server.docbase=webpages
server.tempdir=tmp
server.webapp.examples.mapping=/examples
server.webapp.examples.docbase=examples
```

`WEB-INF` azpidirektorioan zerbitzariaren honako fitxategi hauek kokatzen ditugu:

```
mapping.properties
mime.properties
servlet.properties
webapp.properties
```

Lauetatik, `servlet.properties` fitxategia da guri gehien interesatzen zaiguna. Fitxategi horretan, eraikiko ditugun servleten informazioa jarriko dugu, zerbitzariak ezagutu ditzan egikaritzean. Servletak parametrorik ez badu, honako formatu hau dauka:

```
<servletIzena>.code = <servletKlasea>
```

Adibiderako honako lerro hau idazten dugu:

```
KaixoPatxi.code = KaixoPatxiServlet
```

Servletak parametroak izan balitu, beste lerro bat idatzi beharko genuke formatu honetan:

```
<servletIzena>.initparams = <izena1 = balioa1>,
                            <izena2 = balioa2>,.....
```

Gure adibideak parametrorik ez duenez ez du lerro hori behar.

Servletaren konpilazioarekin arazorik ez izateko, `servlet.jar` fitxategia non dagoen zehaztu behar du `CLASSPATH` inguruneko aldagaiak. Horretarako, honako hau egiten dugu:

```
set classpath = c:\<zerbitzariaren_instalazioa>\servlet.jar
```

Hori guztia egin ondoren, `KaixoPatxi.java` fitxategia konpilatzen dugu eta emaitza `servlets` azpidirektorioan kokatzen dugu.

Servleta egikaritu baino lehen zerbitzaria martxan jartzen dugu `startserver.bat` fitxategia egikarituta. Ondo joan bada guztia, MS-DOS leiho (**3.6. irudia**) bat irekitzen da, eta leiho horretan informazioa jarriko du zerbitzariak servletak egikaritzen diren heinean.

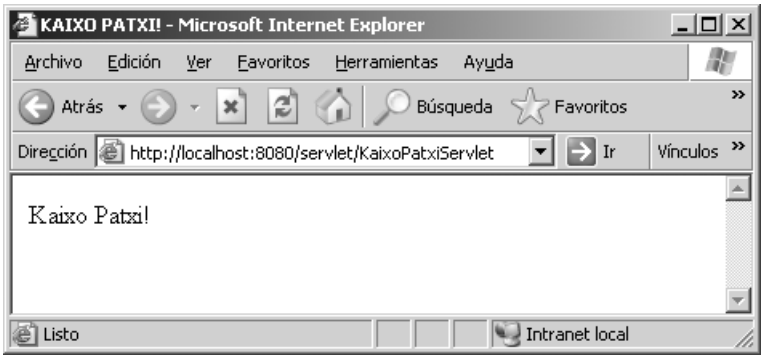


3.6. irudia. Adibidearen MS-DOS irteera.

Orain prest gaude gure lehen servleta egikaritzeko. Horretarako, nabigatzailea irekitzen da eta servletaren helbidera atzitzen gara eskaera bat sortzeko. Eskaera jaso eta dagokion metodora atzitzen da servleta erantzuna sortzeko eta nabigatzailean ikusteko. Esan dugun guztia egikaritzeko nahikoa izango zaigu ireki dugun nabigatzailean honako hau idaztea:

```
http://localhost:8080/servlet/KaixoPatxi
```

Exekuzioaren emaitza **3.7. irudia**n ikus daiteke:

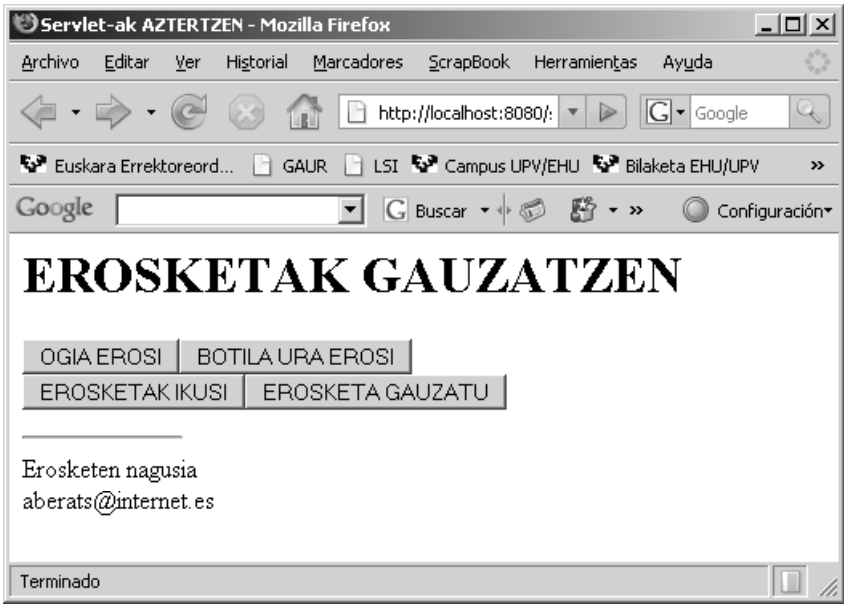


3.7. irudia. Adibidearen irteera nabigatzailean.

Oraintsu azaldu dugun servletak, nahiz eta esanguratsua izan servleten ezaugarriak aurkezteko, ez zuen gauza handirik egiten programa gisa. Nabigatzailean servletari dei egiten zitzaion eta, `doGet()` metodoa egikarituta, mezu xume bat bistaratzen zuen soilik. Abagunea heldu zaigu, beraz, servleten eta, aldi berean, bezero/zerbitzari arkitekturaren funtzionalitatea igotzeko.

Horretarako, ohiko aplikazio txiki bat inplementatuko dugu, servletak erabilita. Ohiko aplikazioa diogu, salerosketa elektroniko xume bat inplementatuko baitugu. Salerosketa gauzatzeko, lehen konexioa sortu eta gero, erabiltzailea behin baino gehiagotan konekta daiteke servletarekin erosketa eguneratzeko. Konexioa irekita mantenduko du servletak, erosketa amaitu arte. Hori lortzeko sesio (`HttpSession`) bat irekiko du servletak hasieran eta irekita mantenduko du erosketak irauten duen bitartean. Aurreko servletean bezala, HTTP protokoloa erabiliko dugu servleta diseinatzeko, eta erabiltzaileak ikusiko duen interfaze grafikoa HTML eran egongo da kodetuta eta interpretatua nabigatzailean.

Servleta martxan jarri ondoren **3.8. irudia**n agertzen den interfaze grafikoa aurkeztuko dio erabiltzaileari. Irudian ikusten den legez, erabiltzaileak botila urak edota ogiak eros ditzake botoi egokiak zapalduta, eta une oro erosketaren egoera azter dezake, jakiteko nola dagoen erosketa-saskia.



3.8. irudia. Aplikazioaren hasierako irteera grafikoa.

Gure lehen salerosketa-aplikazioa denez, ez dugu gehiago nahasiko programa eta salerosketa amaitu ondoren salerosketari buruzko informazio guztia galdu egingo da. Jakina, egoera erreal batean informazio hori biltegiratu egingo litzateke, ordainagiriak ateratzeko eta ordainketak kontrolatzeko, baina hori guztia ondoko adibideetarako utziko dugu.

Aplikazioaren betekizunak eta diseinua azaldu ondoren, hona hemen servletaren kodea:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class OgiUraServlet extends HttpServlet {

  // Bezeroari bidaltzen diogun orria sortzen da hemen
  // botoiak sortzen dira erabiltzaileak zapaltzeko eta ekintzak sortu
  protected void doGet( HttpServletRequest req,
    HttpServletResponse res ) throws ServletException,IOException {
    res.setContentType( "text/html" );
    PrintWriter out = res.getWriter();
    out.print( "<HTML><HEAD><TITLE>Servlet-ak
AZTERTZEN</TITLE></HEAD>"+
      "<BODY BGCOLOR=\"white\" TEXT=\"black\" LINK=\"blue\" "+
      "VLINK=\"purple\" ALINK=\"yellow\">"+
      "<H1>EROSKETAK GAUZATZEN</H1>"+
```

```
      "<FORM METHOD=POST>"+
           "<INPUT TYPE=SUBMIT NAME=ogi VALUE="+
           "\"OGIA EROSI\">"+
           "<INPUT TYPE=SUBMIT NAME=ura VALUE="+
           "\" BOTILA URA EROSI\"><BR>"+
           "<INPUT TYPE=SUBMIT NAME=ikusi VALUE="+
           "\"EROSKETAK IKUSI\">"+
           "<INPUT TYPE=SUBMIT NAME=erosi VALUE="+
           "\"EROSKETA GAUZATU\">"+
      "</FORM><HR WIDTH=\"20%\" ALIGN=\"LEFT\">"+
      "Erosketen nagusia"+
      "<BR>aberats@internet.es"+
      "</BODY></HTML>" );
    out.close();
    }

  // Hemen bezeroaren eskaerak prozesatzen dira.
  // Aurreko doGet()metodoa erabiliz erabiltzaileak formularioa
  // bidaliko dio servletari
  protected void doPost( HttpServletRequest req,
    HttpServletResponse res ) throws ServletException,IOException {
    String mezua = "";
    HttpSession sesioa = req.getSession( true );

    // Lehen aldia bada hasieraketak gauzatzen ditugu
    if( sesioa.isNew() ) {
      sesioa.putValue( "ogi",new int[] { 0 } );
      sesioa.putValue( "ura",new int[] { 0 } );
      }

    // Aurreko konexioetan sortu diren balioak eskuratzen ditugu
    int[] ogi = (int[])sesioa.getValue( "ogi" );
    int[] ura = (int[])sesioa.getValue( "ura" );

    // Formulario bidez eskatutako ekintza kudeatzen dugu hemen
    // OGI balio bat gehitzen dugu saskian
    if( req.getParameter( "ogi" ) != null ) {
      ogi[0]++;
      mezua = "OGIA EROSI DUZU. Orain duzu <B>"+ogi[0]+"</B>.";
      }
    // Botila URA gehitzen dugu saskian
    else if( req.getParameter( "ura" ) != null ) {
      ura[0]++;
      mezua = "Botila bat UR erosi duzu. Orain duzu
<B>"+ura[0]+"</B>.";
      }
    // Saskian dagoena erosten dugu
    else if( req.getParameter( "erosi" ) != null ) {
      sesioa.invalidate();
      mezua = "Zure agindua bete dugu  <B>"+ogi[0]+"</B> OGI"+
        "  eta  <B>"+ura[0]+"</B> botila UR.<BR>"+
```

```
        "EROSKETA BURUTUTA!!!";
      }
    // Saskian ditugun balioak ikusten ditugu
    else {
      mezua = "Orain arte duzuna: <B>"+ogi[ 0] +"</B> OGI "+
        " eta <B>"+ura[ 0] +"</B> botila UR.";
      }

    // Bezeroak egin duen eskariaren erantzuna
    // HTML kodean itzultzen da nabigatzailean ikusteko
    res.setContentType( "text/html" );
    res.setHeader( "pragma","no-cache" );
    PrintWriter out = res.getWriter();
    out.print( "<HTML>\n" );
    out.print( "<HEAD>\n" );
    out.print( "<TITLE>Erosketak gauzatzen</TITLE>\n" );
    out.print( "</HEAD>\n" );
    out.print( "<BODY BGCOLOR=\"white\" TEXT=\"black\"
LINK=\"blue\" VLINK=\"purple\" ALINK=\"yellow\">\n");
    out.print( "<H1>Erosketak Internet erabilita</H1>\n" );
    out.print( mezua );
    out.print( "<HR><A HREF=\"" );
    out.print( req.getRequestURI() );
    out.print( "\">ITZULI EROSKETETARA</A></BODY></HTML>" );
    out.close();
    }

  public String getServletInto() {
    return "OGIURA_Servlet. Erosketak gauzatzen Internet bidez";
      }  }
```

Kodearen lehen lerroak ezagunak zaizkigu, dagoeneko lehen adibidearen antzekoak direlako. Aplikazioaren aurkezpenean esan dugun legez, servletak HTTP protokoloa erabiliko duenez, bezeroa eta zerbitzaria komunikatzeko, `HttpServlet` klasetik eratortzen dugu servleta, klasearen baliabideez baliatzeko. Bestela, servletak, kodean ikusten den bezala, bi metodo izango ditu aipatu dugun funtzionalitatea inplementatzeko: `doGet()` eta `doPost()`.

`doGet()` metodoa hasierako interfaze grafikoaz arduratzen da, eta zapaldu daitezkeen botoiak aurkezten zaizkio erabiltzaileari, HTML orri bat eraikita eta bezeroari erantzunean bidalita. Horretarako, lehen servletean bezala, `HttpServletResponse` motako objektu bat erabiliko du metodoak. Konturatu sortzen den HTML orria formulario bat izango dela, eta, erabiltzaileak botoi bat zapaltzen duenean, `POST` balioa hartuko duela `METHOD` atributuak:

```
<FORM METHOD = POST>
```

Horrela, servletak badaki aurkezpeneko interfaze grafikoan botoia zapaltzen denean `doPost()` metodoa egikaritu behar duela. `doPost()` metodoa zertxobait

konplexuagoa da, hortxe baitago aplikazioaren logika nagusia. Bezeroak bidalitako datuak formularioa erabilita, metodo horretan prozesatuko dira. Lehenengo eta behin, zerbitzariak sesio bat sortzen du erabiltzaile bakoitzarentzat:

```
    HttpSession sesioa = req.getSession( true );
  // Lehen aldia bada hasieraketak gauzatzen ditugu
  if( sesioa.isNew() ) {
    sesioa.putValue( "ogi",new int[] { 0 } );
    sesioa.putValue( "ura",new int[] { 0 } );
    }
  // Aurreko konexioetan sortu diren balioak eskuratzen ditugu
  int[] ogi = (int[])sesioa.getValue( "ogi" );
  int[] ura = (int[])sesioa.getValue( "ura" );
```

`HttpSession` motako objektu bat eskuratu ondoren, `putValue(String izena, Object balioa)` metodoa erabiltzen dugu balioak hasieratzeko.

Sesioa irekita mantentzen da, `invalidate()` metodoa egikaritu arte. Gure kasuan erosketa amaitzen denean ixten dugu sesioa: `sesioa.invalidate()`.

Sesioa berria ez bada, ezer egin baino lehen, erabiltzailearen balioak eskuratzen ditugu sesioaren `getValue()` metodoa erabilita. Horrela, erosketaren egoera eguneratuta dugu konexio bakoitzean.

Hasierako eragiketa horiek egin ondoren, formularioak eskatzen digun eragiketa funtzionala gauzatzen da. Beti, HTML orri bat eraikitzen da irteera gisa, goiburu- eta bukaera-formatu berarekin. Tartean jartzen den mezua (**3.9. irudia**) egokitu egiten da eskatzen den eragiketaren tankerara:

```
    res.setContentType( "text/html" );
    res.setHeader( "pragma","no-cache" );
    PrintWriter out = res.getWriter();
    out.print( "<HTML>\n" );
    out.print( "<HEAD>\n" );
    out.print( "<TITLE>Erosketak gauzatzen</TITLE>\n" );
    out.print( "</HEAD>\n" );
    out.print( "<BODY BGCOLOR=\"white\" TEXT=\"black\"
LINK=\"blue\"     VLINK=\"purple\" ALINK=\"yellow\">\n");
    out.print( "<H1>Erosketak Internet erabilita</H1>\n" );
    out.print( mezua );
    out.print( "<HR><A HREF=\"" );
    out.print( req.getRequestURI() );
    out.print( "\">ITZULI EROSKETETARA</A></BODY></HTML>" );
    out.close();
```

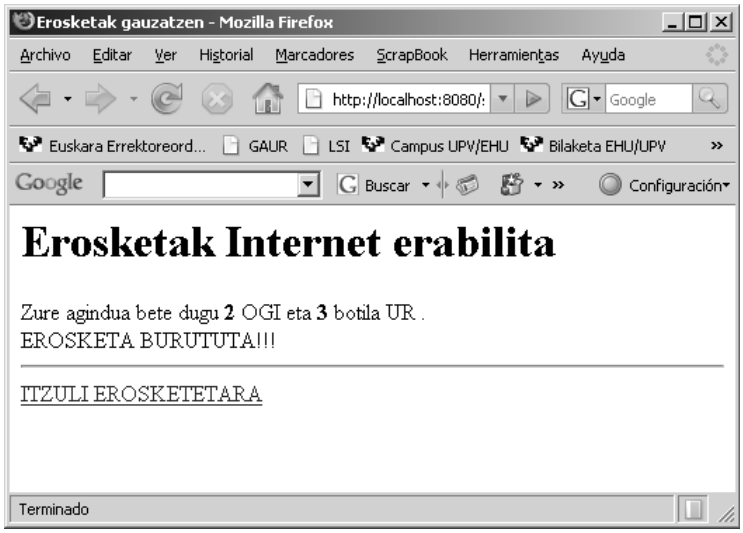
Zein eragiketa funtzional gauzatuko duen erabakitzeko, `getParameter()` metodoa erabiltzen du `HttpServletRequest` motako objektuak. Gogoratu parametro gisa pasatzen zaiola `doPost()` metodoari objektu hori. Adibide gisa, ikus dezagun nola prozesatzen den erosketaren bukaera:

```
 if( req.getParameter( "erosi" ) != null ) {
sesioa.invalidate();
mezua = "Zure agindua bete dugu  <B>"+ogi[ 0] +"</B> OGI"+
  "  eta  <B>"+ura[ 0] +"</B> botila UR.<BR>"+
  "EROSKETA BURUTUTA!!!";
}
```

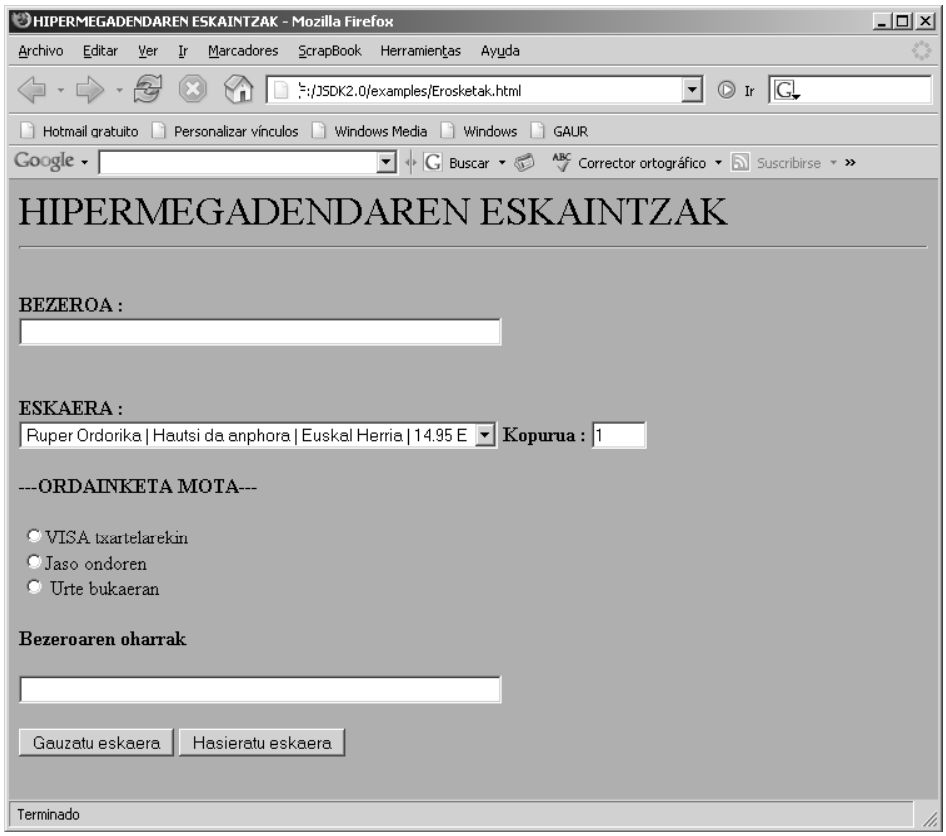
Servletak erosketa gauzatu nahi duenez, sesioa (**3.10. irudia**) ixten du erabiltzaileak eta mezu egokia eraikitzen du adieraziz zenbat ogi eta zenbat botila ur erosi duen.



3.9. irudia. Erabiltzailearen erosketen interfaze grafikoa.



3.10. irudia Aplikazioaren azken irteera grafikoa.



3.11. irudia. Adibidearen interfaze grafikoa.

Bigarren adibidean, oinarrizko salerosketa elektroniko bat nola egituratu aztertu dugu, bezero/zerbitzari arkitektura bat erabilita. Gogoratu aplikazioa nahiko oinarrizkoa zela; izan ere, ez zuen eskaera biltegiratzen eta erabiltzailearen ekintza bakarra botoiak zapaltzea zen. Hirugarren adibidean bi ezaugarri horiek hobetu egingo ditugu eta fitxategi batean biltegiratuko dugu eskaera, bezeroaren informazioa gordeta. Hirugarren aplikazio horretan, erabiltzailearen lehen interfaze grafikoa HTML orri bat (**3.11 irudia**) izango da eta servleta orri horretatik jarriko da martxan.

Servleta formulario batean egituratuko dugu, `action` atributuan adierazita zein den eta non dagoen egikaritu nahi dugun servleta. `Method` atributuak `POST` balioa izango du, eta horrela, zerbitzariak badaki servleta egikaritzen denean `doPost()` metodoa bilatu behar duela eta egikaritu.

Bestalde, HTML kodea `name-value` bikotez beterik egongo da. Bikote horien bidez, balioak pasatzen dizkiogu servletari, parametroak erabilita. Parametro horietan, zein den bezeroa, zer den eskatu duena, zein ordainketa mota aukeratu duen eta ohar orokorrak adieraziko ditu erabiltzaileak. Hona hemen HTML orriaren kode guztia, servleta martxan jartzeko:

```
      <html>
        <head>
        <title>HIPERMEGADENDAREN ESKAINTZAK</title>
      </head>
      <body bgcolor="#33CCFF">
        <font face="Times New Roman,Times" size="+3">
          HIPERMEGADENDAREN ESKAINTZAK
        </font>
        <hr><p>
        <center>
        </center>
<form action=http://localhost:8080/servlet/eskaeraServlet method=POST>
            <input type=hidden name=eskaera value=EskaerarenBalioak>
      <BR> <b>BEZEROA :</b><BR> <input type="text" name="bezeroa"
SIZE=60> <BR> <BR>
          <BR> <b>ESKAERA :</b><BR>
          <select name=CD>
<option>Ruper Ordorika | Hautsi da anphora | Euskal Herria | 14.95
E</option>
<option>EKM| Gogoz eta asmoz | Euskal Herria| 16.95 E</option>
          <option>Kaira | Tounami Diabate| Mali | 16.95 E</option>
<option>Shakuhachi| Kohachiro Miyata | Japonia | 13.95 E</option>
<option>Dance the Devil Away | Outback | Australia | 14.95
E</option>
<option>Record of Changes | Samulnori | Korea | 12.95 E</option>
          <option>Djelika | Tounami Diabate | Mali | 14.95 E</option>
<option>Rapture | Nusrat Fateh Ali Khan | Pakistan | 12.95 E</option>
<option>Cesaria Evora | Cesaria Evora | Cabo Verde | 16.95 E</option>
          <option>Ibuki | Kodo | Japonia | 13.95 E</option>
        </select>
<b>Kopurua : </b><input type="text" name="kopurua" SIZE="3" value=1>

            <BR><BR> <b>--ORDAINKETA MOTA--</b><BR>
              <BR><input type=radio name=ordainketa value=Txartela>
VISA txartelarekin
             <BR><input type=radio name=ordainketa value=JasOndoren>
Jaso ondoren
<BR><input type=radio name=ordainketa value=Urtean> Urte bukaeran

            <BR><BR><b>Bezeroaren oharrak</b> <BR>
              <BR><input type=text name=oharra SIZE=60>
             <BR><BR><input type=submit value="Gauzatu eskaera">
                    <input type=reset value="Hasieratu eskaera">
          </form>
        </body>
      </html>
```

Erabiltzaileak, hutsuneak bete ondoren, *Gauzatu eskaera* botoia zapaltzen badu, zerbitzariak servleta kargatzen du eta martxan jartzen da. Eskaera hasieratu

nahi bada, *Hasieratu eskaera* botoia zapaltzen da, eta interfaze grafikoaren eremu guztiak hutsik gelditzen dira eta prest eskaera sartzeko.

Diseinatzen ari garen servletak ez du zailtasun handirik izango, bere lan bakarra eskaera biltegiratzea izango baita `txt` motako fitxategi batean. Beraz, nahikoa zaigu bi metodo programatzea: `init()` eta `doPost()`.

`init()` metodoarekin hasierako eragiketak gauzatuko ditugu, servletari igortzen zaizkion metodoak egikaritu baino lehen. Metodo horretan, `getInitParameter()` metodoa egikarituko da, servletaren hasierako parametroak eskuratzeko. Azpimarratu behar dugu ezen `InitParameter` horiek ez datozela HTML orritik, `servlet.properties` [26] fitxategitik baizik. Gure kasuan emaitzen direktorioa zein izango den adierazten du parametroaren balioak. `init()` metodoak kode birdefinitu hau izango du:

```
    public void init(ServletConfig config)
    throws ServletException
{
    super.init(config);
      emaitzenDir = getInitParameter("emaitzenDir");
    if (emaitzenDir == null) {
        Enumeration initParametroak = getInitParameterNames();
        System.err.println("Hasierako parametroak : ");
        while (initParametroak.hasMoreElements()) {
           System.err.println(initParametroak.nextElement());
        }
        System.err.println("Errorea parametroetan !!!!!");
        throw new UnavailableException (this,
           "Ez didazu direktoriorik eman emaitzak gordetzeko!");
    }
}
```

`doPost()` metodoaren eginbeharra eskaeraren emaitzak irteerako fitxategian idaztea izango da, `POST` metodoari erantzunez. Horretarako, irteerako fitxategia irekitzen du lehendabizi, eta ondoren eskaeraren parametroen izenak eta balioak eskuratzen ditu, fitxategian idazten hasteko. Fitxategian, `<HASIERA>` eta `<BUKAERA>` etiketak izango ditu erabiltzailearen eskaera bakoitzak, ondo bereizteko bezeroaren eskaera bakoitza. Esan dugun guztia gauzatzeko honako kode hau idatzi beharko dugu `doPost()` metodoan:

---

26. Gogoratu `servlet.properties` fitxategian parametroak eta balioak pasatzen dizkiola programatzaileak servletari, formatu honi jarraituz:

`servlet.<servletIzena>.initArgs= <izena1 = balioa1>, <izena2 = balioa2>,.....`

Gure kasuan honako hau idatzi beharko dugu irteerako fitxategia eraikitzeko:

```
# erosketa servlet
servlet.eskaeraServlet.code=eskaeraServlet
servlet.eskaeraServlet.initArgs=\emaitzenDir=/tmp
```

```
     fitxGorde.println("<HASIERA>");
     Enumeration balioak = req.getParameterNames();
        while(balioak.hasMoreElements()) {
          String izena = (String)balioak.nextElement();
          String balioa = req.getParameterValues(izena)[0];
             if(izena.compareTo("submit") != 0) {
                fitxGorde.println(izena + ": " + balioa);
                }
           }
     fitxGorde.println("<BUKAERA>");
```

Kodean ikus daitekeenez, `HttpServletRequest` motako objektuak `getParameterNames()` eta `getParameterValues()` metodak erabiltzen ditu, parametroen izenak eta balioak eskuratzeko. Hori egin ondoren, gelditzen den eragiketa bakarra parametroen balioak eta izenak fitxategian biltegiratzea da. Servleta behin baino gehiagotan egikaritu ondoren, honela gelditzen litzateke irteerako fitxategia:

```
<HASIERA>
ordainketa: Txartela
kopurua: 10
oharra: Baieztatu nire helbide berria !!!!!
eskaera: EskaerarenBalioak
CD: Ruper Ordorika | Hautsi da anphora | Euskal Herria | 14.95 E
bezeroa: Xabier Zubizarreta
<BUKAERA>
<HASIERA>
ordainketa: Urtean
kopurua: 15
oharra: Gabonetan ez bidali ordainagiririk
eskaera: EskaerarenBalioak
CD: Cesaria Evora | Cesaria Evora | Cabo Verde | 16.95 E
bezeroa: Pello Zubizarreta Arano
<BUKAERA>
```

Idazketa-prozesu guztia ondo joan bada, HTML orri bat sortzen du `doPost()` metodoak, eta bezeroari bidaltzen dio erosketa ondo gauzatu dela adieraziz. Hona hemen `doPost()` metodoaren kode osoa nola gelditzen litzatekeen azkenean:

```
     public void doPost(HttpServletRequest req, HttpServletResponse
res)
       throws ServletException, IOException
    {
        // lehenengo, finkatu erantzunaren "content type" goiburua
       res.setContentType("text/html");

       // Bezeroari testu bat itzultzeko antolatu
      // erantzunaren PrintWriter .
        PrintWriter bezeroariBidal = res.getWriter();
```

```
        try {

             // Ireki fitxategia eskaeraren informazioa gordetzeko
            String eskaeraGorde =
req.getParameterValues("eskaera")[0];
  FileWriter emaitzenFitxategia = new FileWriter(eskaeraGorde +
".txt", true);
            PrintWriter fitxGorde = new
PrintWriter(emaitzenFitxategia);

     // Bezeroaren formularioko datuak eskuratu eta
     //biltegiratu txt motako fitxategian
            fitxGorde.println("<HASIERA>");
            Enumeration balioak = req.getParameterNames();
            while(balioak.hasMoreElements()) {
                String izena = (String)balioak.nextElement();
             String balioa = req.getParameterValues(izena)[0];
                if(izena.compareTo("submit") != 0) {
                    fitxGorde.println(izena + ": " + balioa);
                }
            }
            fitxGorde.println("<BUKAERA>");

           //Emaitzen fitxategia itxi.
            emaitzenFitxategia.close();

        // Bezeroari erantzun ondo joan bada eskaeraren prozesu
           guztia
           bezeroariBidal.println("<html>");
           bezeroariBidal.println("<title>ESKERRIK ASKO!</title>");
            bezeroariBidal.println("EROSKETA ONDO JOAN DA");
           bezeroariBidal.println("</html>");

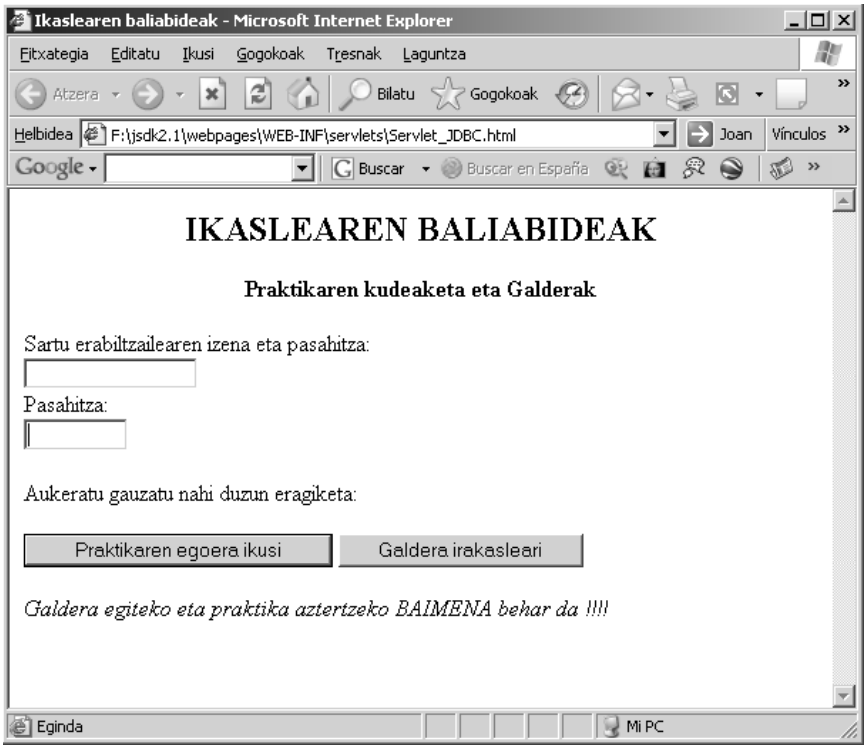
        } catch(IOException e) {
            e.printStackTrace();
            bezeroariBidal.println(
             "Arazoak sortu dira eskaera gordetzeko orduan. "
             + "Saiatu berriro.");
        }
        // Erantzuna gauzatuta. Itxi fitxategia
      bezeroariBidal.close();
    }   }
```

Servleten aurkezpen hau bukatzeko, aplikazio erreal bat diseinatu eta programatuko dugu. Aurreko adibideetan ikusi duguna laburbilduko du aplikazioak, alde batetik, eta, bestetik, datu-baseak erabiliko ditu informazioa biltegiratzeko eta erabiltzailearen pasahitza aztertzeko.

Oraingoan domeinuz aldatuko dugu eta irakaskuntzaren arloan murgilduko gara. Diseinatu eta inplementatu nahi dugun aplikazioarekin irakasgai bateko

ikasleek aukera izango dute, Internet bidez, irakasgaiaren bi arlo arakatzeko: tutoretzak eta irakasgaiko praktikak.

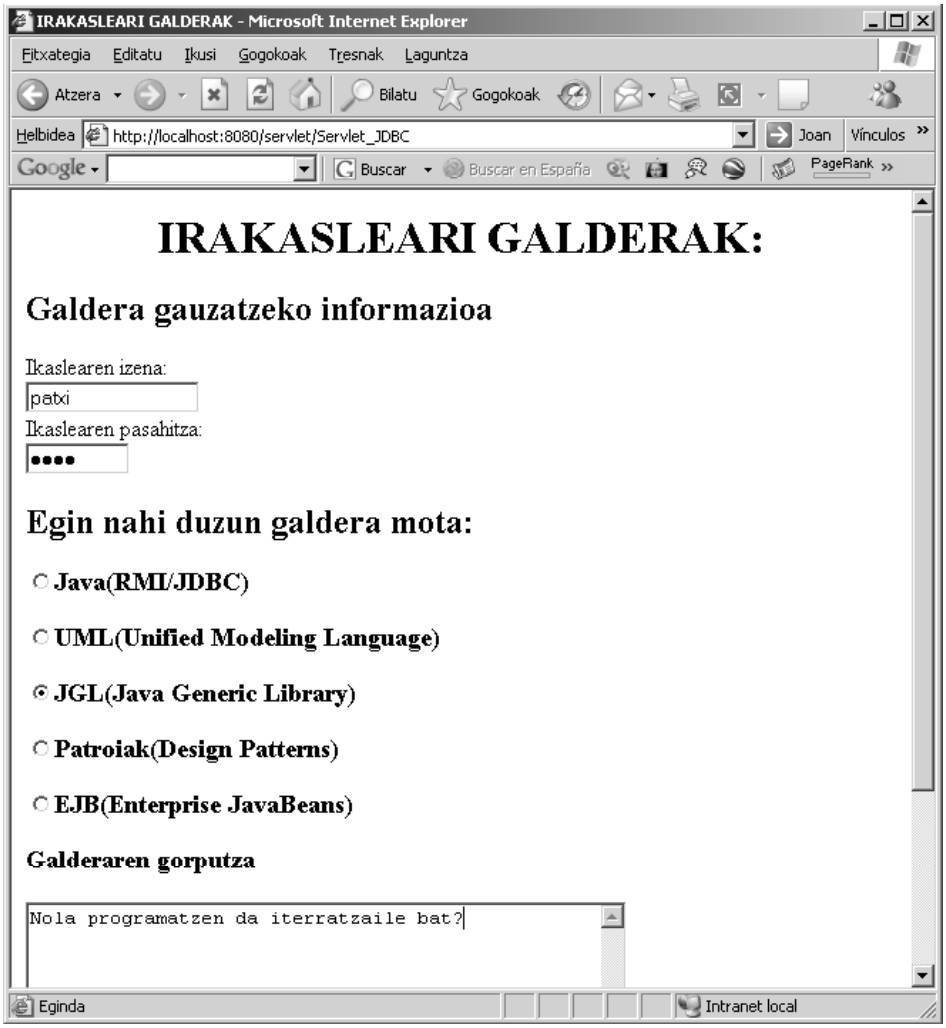
Ikasleak tutoretza birtuala gauzatu dezake etxetik, Internet bidez, galderak eginda irakasleari. Irakasleak behin galdera aztertuta, berriz erantzungo dio teknologia bera erabilita. Aldi berean, kontsultak gauzatu ditzake ikasleak Internet bidez, irakasgaiaren praktiken egoera ikusteko. Bai batean eta bai bestean, erabiltzaileari izena eta pasahitza eskatuko zaizkio, baimena eduki dezan aplikazioa egikaritzeko.



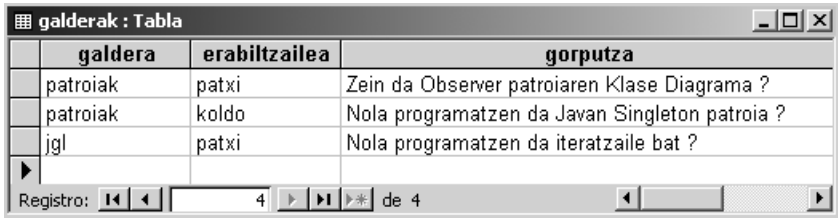
3.12. irudia. Adibidearen hasierako interfaze grafikoa.

3.12. irudian ikusten den hasierako interfaze grafikoa izango du aplikazioak. Lehenengo eta behin, bere izena eta pasahitza sartzen ditu erabiltzaileak, eta gauzatu nahi duen eragiketa aukeratzen du, botoi bat zapalduta.

Demagun ikasleak tutoretza bat eskatu nahi duela, eta horretarako galdera bat bidali nahi dio irakasleari. Hori egiteko beste leiho bat irekitzen dio aplikazioak **3.13. irudia**n ikusten den itxurarekin. Galdera zehazteko, ikasleak lehendabizi zein izango den galdera mota (UML/JGL/Patroiak/EJB) aukeratu behar du. Galdera mota jakinda, irakasleak errazago antzemango dio ikasleak jarraian egingo dion galderaren nondik norakoari. Galdera biltegiratuta gelditukо da datu-basearen taula batean, **3.14. irudia**n ikusten den legez.

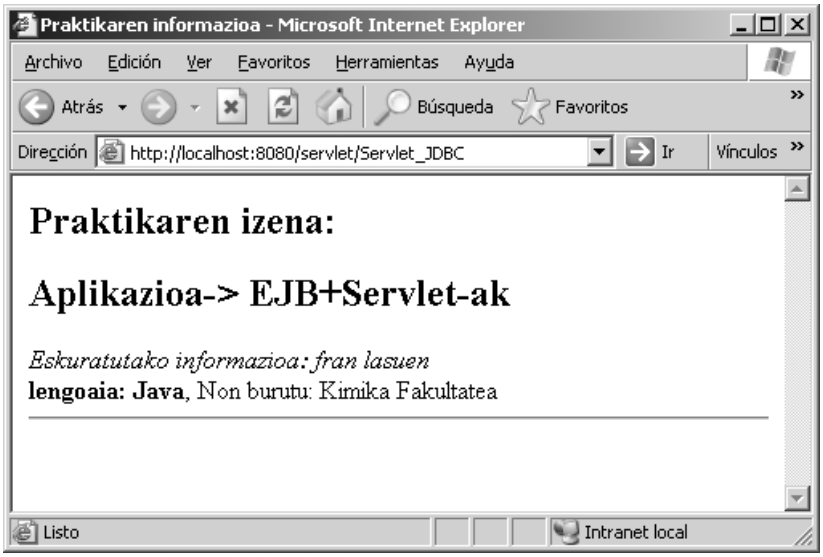


3.13. irudia. Ikaslearen galderaren interfaze grafikoa.



3.14. irudia. Datu-basearen egoera ikasleen galderak biltegiratu ondoren.

Tutoretza gauzatu beharrean, irakasgaiako praktika aztertu nahi badu erabiltzaileak, lehen bezala, izena eta pasahitza sartu beharko ditu lehendabizi. Baimena jasotzen badu, praktikaren informazioa leiho batean agertuko zaio. Praktikaren informazioa datu-basearen taula bat atzituta eskuratuko da. **3.15. irudia**n, agertuko den leihoa ikusten da, eta **3.16. irudia**n, kontsultatzen den taula agertzen da.



3.15. irudia. Ikaslearen praktikaren interfaze grafikoa.



3.16. irudia. Ikaslearen praktikaren informazioa biltegiratzen duen taularen diseinua.

Aplikazioaren betekizunak eta diseinua aztertu ondoren, servlet bat inplementatzea pentsatu dugu, aitzinean aipatu dugun funtzionalitate osoa jasotzeko. Servletak berak, beharrezkoak zaizkigun HTML orriak eraikiko ditu, eta bezeroari bidaliko dizkio. Datu-basera egin behar diren atzipen guztiak datuak kontsultatzeko eta txertatzeko, servletean bertan gauzatuko dira. Eskuratutako informazioa HTML orrietan bidaliko dio servletak bezeroari.

Aipatu ditugun betekizun guztiak programatuta, honela gelditu da servleta:

```
import java.io.*;
import java.net.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class Servlet_JDBC extends HttpServlet {
  String DBurl = "jdbc:odbc:Baliabideak";
  String erabiltzaileGet = "";
  String erabiltzailePost = "";
  String pasahitzaGet = "";
  String pasahitzaPost = "";
  Connection con;
  DatabaseMetaData metaData;

  // Metodo honek hasieraketak eta datu-basearekin konexioa
  // finkatzen ditu
  public void init( ServletConfig conf ) throws ServletException {
    SQLWarning w;

    super.init( conf );
    try {
      Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
      con = DriverManager.getConnection(
DBurl,erabiltzailePost,pasahitzaPost );
      if( (w = con.getWarnings()) != null ) {
        while( w != null ) {
          log( "SQLWarning: "+w.getSQLState()+'\t'+
               w.getMessage()+'\t'+w.getErrorCode()+'\t' );
          w = w.getNextWarning();
        }
      }
    } catch( ClassNotFoundException e ) {
      throw new ServletException( "init" );
    } catch( SQLException e ) {
      try {
        con = DriverManager.getConnection(
DBurl,erabiltzaileGet,pasahitzaGet );
      } catch( SQLException ee ) {
        ee.printStackTrace();
        while( e != null ) {
          log( "SQLException: "+e.getSQLState()+'\t'+
               e.getMessage()+'\t'+e.getErrorCode() );
          e = e.getNextException();
        }
        throw new ServletException( "init" );
      }
    }

  public void service( HttpServletRequest req,HttpServletResponse res )
  throws ServletException,IOException {
    String erabiltzailea = req.getParameter( "erabiltzailea" );
    String onartuta;
    String ekintza = req.getParameter( "ekintza" );
```

```
    try {
      onartuta = baimena( req );
      if( ekintza.equals( "Praktikaren egoera ikusi" )
           && onartuta.equals( "GET") ) {
       irakurriPraktika(req, res);
                }
      else if( ekintza.equals( "Bidali" )
        && onartuta.equals( "POST" ) ) {
        bidaliGaldera( req,res );
        }
      else if ( ekintza.equals("Galdera irakasleari" ) ) {
        if ( erabiltzailea == null )
          erabiltzailea = " ";
        PrintWriter out = new PrintWriter( res.getOutputStream() );
        out.println( "<HTML>" );
        out.println( "<HEAD><TITLE>IRAKASLEARI GALDERAK
</TITLE></HEAD>" );
        out.println( "<BODY>" );
     out.println( "<CENTER><H1>IRAKASLEARI GALDERAK:
</H1></CENTER>" );
        out.println( "<FORM
ACTION=http://localhost:8080/servlet/Servlet_JDBC METHOD=POST>" );
        out.println( "<H2>Galdera gauzatzeko informazioa</H2>" );
        out.println( "<P>Ikaslearen izena: <BR>" );
        out.println( "<INPUT NAME=erabiltzailea TYPE=text
VALUE='"+erabiltzailea+"' SIZE=16 MAXLENGTH=16><BR>" );
        out.println( "Ikaslearen pasahitza: <BR>" );
        out.println( "<INPUT NAME=gakoa TYPE=password SIZE=8
MAXLENGTH=8></P>" );

       out.println("<H2> Egin nahi duzun galdera mota: </H2>");
   out.println("<H3> <input type=radio name=galdera
value=java>Java(RMI/JDBC)</H3>");
   out.println("<H3><input type=radio name=galdera
value=uml>UML(Unified Modeling Language)</H2>");
   out.println("<H3><input type=radio name=galdera
value=jgl>JGL(Java Generic Library)</H3>");
   out.println("<H3><input type=radio name=galdera
value=patroiak>Patroiak(Design Patterns)</H3>");
   out.println("<H3><input type=radio name=galdera
value=ejb>EJB(Enterprise JavaBeans)</H3>");

        out.println( "<H3>Galderaren gorputza</H3>" );
        out.println( "<P><TEXTAREA NAME=gorputza ROWS=10
COLS=50></TEXTAREA></P>" );
        out.println( "<P><INPUT NAME=ekintza TYPE=submit
VALUE='Bidali'></FORM>" );
        out.println( "</BODY></HTML>" );
        out.flush();
        }
```

```
      else {
        PrintWriter out = new PrintWriter( res.getOutputStream() );
        out.println( "<html>" );
     out.println( "<head><title>Atzipena debekatuta</title></head>" );
        out.println( "<body>" );
        out.println( "Atzipen errore bat eman da sisteman:<br>" );
        out.println( "Sartutako erabiltzailea edo gakoa ez dira
egokiak <br>" );
        out.println( "ez duzu atzipenik funtzionalitate honetara." );
        out.println( "</body></html>" );
        out.flush();
        }
    } catch( SQLException e ) {
      while( e != null ) {
        log( "SQLException: "+e.getSQLState()+'\t' +
             e.getMessage()+'\t' +e.getErrorCode()+'\t' );
        e = e.getNextException();
        }
      }
    }


  // Konexioa ixten da datu-basearekin
  public void destroy() {
    try {
      con.close();
    } catch( SQLException e ) {
      while( e != null ) {
        log( "SQLException: "+e.getSQLState()+'\t' +
             e.getMessage()+'\t' +e.getErrorCode()+'\t' );
        e = e.getNextException();
        }
    } catch( Exception e ) {
      e.printStackTrace();
      }
    }


  // Metodo honetan kontsulta gauzatzen da,
  // praktikaren egoera aztertzeko.
  // Emaitza HTML orrian eraikitzen da eta bezeroari bidaltzen zaio

  public void irakurriPraktika( HttpServletRequest
req,HttpServletResponse res )
  throws IOException,SQLException {
    Statement stmt = con.createStatement();
    String kontsulta="";
    ResultSet rs;

    res.setStatus( res.SC_OK );
```

```
    res.setContentType( "text/html" );
    kontsulta = "SELECT praktika.izenaPra,praktika.lengoaia,";
    kontsulta +=
"praktika.tresna,erabiltzaileak.izena,erabiltzaileak.nongoa ";
    kontsulta += "FROM praktika,erabiltzaileak WHERE
praktika.iden=erabiltzaileak.erabiltzailea";
    rs = stmt.executeQuery( kontsulta );

    PrintWriter out = new PrintWriter( res.getOutputStream() );
    out.println( "<HTML>" );
 out.println( "<HEAD><TITLE>Praktikaren informazioa</TITLE></HEAD>" );
    out.println( "<BODY>" );
    out.println( "<H2>Praktikaren izena:</H2>" );

    while( rs.next() ) {
      out.println( "<H2>" );
      out.println( rs.getString(1) );
      out.println( "</H2><p>" );
      out.println( "<I>Eskuratutako informazioa:
"+rs.getString(4)+"</I><BR> " );
      out.println( "<B>lengoaia: "+rs.getString(2)+"</B>, Non burutu:
"+rs.getString(5) );
      out.println( "<HR>" );
      }
    out.println( "</BODY></HTML>" );
    out.flush();
    rs.close();
    stmt.close();
    }


    // Ikaslearen galdera <galderak> taulan biltegiratzen da
public void bidaliGaldera( HttpServletRequest req,HttpServletResponse
res )
  throws IOException,SQLException {
    Statement stmt = con.createStatement();
    String consulta = "";
    String erabiltzailea= req.getParameter( "erabiltzailea" );

    PrintWriter out = new PrintWriter( res.getOutputStream() );
    res.setStatus( res.SC_OK );
    res.setContentType( "text/html" );
    out.println( "<HTML>" );
    out.println( "<HEAD><TITLE>Burututa eragiketa</TITLE></HEAD>" );
    out.println( "<BODY>" );

    consulta = "INSERT INTO galderak VALUES( '";
    consulta += req.getParameter( "galdera"
)+"','"+erabiltzailea+"','";
    consulta += req.getParameter("gorputza")+"')";
    int result = stmt.executeUpdate( consulta );
```

```
    if( result != 0 ) {
      out.println( "Zure galdera ONDO prozesatu da." );
      }
    else {
      out.println( "Akatsak zure galdera prozesatzen.<BR>" );
      out.println( "Kontsultatu Datu-basearen administratzailea, edo
kontsultatu<BR>" );
      out.println( " servletaren <I>log</I> fitxategia" );
      }
    out.println( "</BODY></HTML>" );
    out.flush();
    stmt.close();
    }


  // Servletaren informazioa itzultzen du
  public String getServletInfo() {
    return "Servlet_JDBC. Tutoretzak eta Praktikak";
    }

   // Metodo honek erabiltzaileak zein baimen mota duen aztertzen du
  public String baimena( HttpServletRequest req ) throws SQLException
{
    Statement stmt = con.createStatement();
    String kontsulta;
    ResultSet rs;
    String ondo = "";
    String erabiltzailea = req.getParameter( "erabiltzailea" );
    String gakoa = req.getParameter( "gakoa" );
    String baimena="";

    kontsulta = "SELECT onartu FROM erabiltzaileak WHERE erabiltzailea
= '" + erabiltzailea;
    kontsulta += "' AND gakoa = '"+gakoa+"'";
    rs = stmt.executeQuery( kontsulta );

    while( rs.next() ) {
      ondo = rs.getString(1);
      }
    rs.close();
    stmt.close();

    if( ondo.equals( "" ) ) {
      baimena = "ATZIPENA DEBEKATUTA";
      }
    else {
      // Baimena praktikak irakurtzeko bakarrik
      if ( ondo.equals( "E" ) ) {
        baimena = "GET";
```

```
        // Baimena galderak bidaltzeko irakasleari
        }
      else if ( ondo.equals( "B" ) ) {
        baimena = "POST";
        }
      }
    return baimena;
    }
        }
```

Aurreko adibidean ikusi dugun legez, servletak `init()` metodoa birdefinitu egiten du, hasierako eragiketak egikaritzeko. Datu-base bateko taulak atzituko dituenez, JDBCren hasieraketa guztiak bertan gauzatuko ditu aplikazioak. Horrela, behin hasieraketa gauzatuta, ez da behin eta berriz egikaritu beharko, servleta, datu-basearen taula batera atzitzen denean.

Hasieraketa egin ondoren, bezeroaren eskaeren zain gelditzen da `service()` metodoa. Eskaera bat jasotzen denean, erabiltzailea eta ekintza identifikatzen ditu metodoak, eta `baimena()` metodoa egikarituta egiaztatzen da erabiltzailea eta haren gakoa datu-basearen taula batean daudela.

Erabiltzailearen profila aztertu ondoren, `baimena()` metodoak `String` balio bat itzultzen du adieraziz erabiltzailearen baimen mota. Aplikazioak baimen mota bi onartzen ditu: baimena praktikak irakurtzeko bakarrik, eta baimena irakasleari galderak bidaltzeko. Ikasle bakoitzak duen baimen mota `erabiltzaileak` (**3.17. irudia**) taularen `onartu` atributuan adierazten da.

erabiltzaileak : Tabla

izena	nongoa	erabiltzailea	gakoa	onartu
patxi etxebarrieta	Informatika Fakultatea	patxi	1111	B
koldo gil	Informatika Fakultatea	koldo	2222	B
fran lasuen	Kimika Fakultatea	fran	3333	E

Registro: 4 de 4

3.17. irudia. Biltegiratutako erabiltzailearen informazioa.

Erabiltzailea onartua izan dela sisteman ziurtatu ondoren, zein ekintza gauzatuko den erabakitzen du `service()` metodoak `ekintza` parametroa aztertuta. Praktika irakurri behar badu, `irakurriPraktika()` metodoa egikaritzen du servletak. Praktikaren informazioa datu-basearen taulatik eskuratzen da eta HTML orri batean bidaltzen dio bezeroari.

Erabiltzaileak irakasgaiaren irakasleari galdera bat egin nahi badio, HTML orri bat eraikitzen du servletak eta bezeroari bidaltzen dio. Galdera gauzatuta, *`Bidali`* botoia zapaltzen du erabiltzaileak, eta servletak jasotzen du galderaren informazioa `POST` metodo gisa formulario batean.

Servletak, `POST` erako eskaera bat jaso duenez, `service()` metodoa egikaritzen du berriz, eta oraingoan, `bidaliGaldera()` metodoari deitzen dio egikaritzeko, ekintza mota `Bidali` izan baita. Erabiltzailearen galderaren ezaugarriak `galderak` taulan txertatzen ditu `bidaliGaldera()` metodoak eta HTML (**3.18. irudia**) orri bat bidaltzen dio erabiltzaileari ohartaraziz txertaketa ondo gauzatu dela.



3.18. irudia. Servletaren erantzuna.

3.2.2. JSP (Java Server Pages) orriak

3.2.2.1. Sarrera

JSP orri bat automatikoki eraikitako servlet bat da. Softwarearen arkitekturaren ikuspegitik servlet baten goi-mailako abstrakziotzat har dezakegu. Aurreko orrietan ikusi dugun bezala, servlet bat programatzeko Java lengoaia erabiltzen da eta nahitaez, prestakuntza eta ezagutza onak izan behar ditu programatzaileak, servleta ondo programatzeko. Askotan, web orriak diseinatzen dituztenak ez dira programatzaileak izaten, ez dute programatzaileak izan behar. Erabiltzailearen ikuspuntutik web orriaren aurkezpenak du garrantzia nahiz eta aurkezpen horren atzetik hainbat eta hainbat iturburu-kode egon erabiltzaileak sartzen duen informazioa tratatzeko.

JSP orri bat martxan jartzeko, JSP edukiontzi bat behar dugu zerbitzarian. Bezeroak lehendabizi eskatzen duenean JSP orri bat, servlet bat eraikitzen du edukiontziak, JSP orriaren informazioa prozesatzeko. Servleta sortu ondoren, konpilatu egiten du edukiontziak, eta servletaren `.class` fitxategia eraikitzen du.

Hasierako prozesu horri guztiari *itzulpen-fasea* deitu ohi izan zaio JSP orriaren bizi-zikloan. Normala izaten da fase horrek denbora pixka bat behar izatea egikaritzeko, hori dela-eta, bezeroaren lehen eskaeraren prozesua motelago joaten da. Batzuetan hasierako prozesu hori azkartzeko, JSP orria aurrekonpilatu egiten da, eraginik izan ez dezan bezeroaren eskaeretan.

Itzulpen-fasea bukatu eta gero, eraikitako servleta prest gelditzen da bezeroaren eskaerak prozesatzeko. Bigarren fase horri, *eskaeraren prozesaketa-fasea* deitu ohi izan zaio.

JSP orria ez bada aldatzen, bezeroaren ondorengo eskaera guztiak zuzenean prozesatuko ditu edukiontziak eskaeraren prozesaketa-fasea egikarituta. Aldaketak egiten badizkiogu JSP orriari betekizun berriak jasotzeko edo garapenean gertatu diren erroreak konpontzeko, berriz igaro beharko da itzulpen-fasetik bezeroaren eskaera prozesatu baino lehen.

3.2.2.2. JSP orriaren izaera

JSP orri batean, hiru osagai mota aurki daitezke: aginduak, etiketak eta scripting osagaiak.

Aginduak erabilita, JSP orriari buruzko informazioa sortzen dugu. JSP edukiontziak erabiliko du aginduaren informazioa, eraikiko duen servletaren konfigurazio orokorra egituratzeko. Aginduaren sintaxia honako hau izango da:

```
<%@ agindua { atributua = balioa }  %>
```

Hiru agindu mota erabil daitezke: orriarekiko agindua (`page`), txertaketa-agindua (`include`) eta etiketen liburutegiaren agindua (`taglib`).

*Orriarekiko agindua*k honako atributu hauek eduki ditzake: `language`, `extend`, `import`, `session`, `buffer`, `autoflush`, `isThreadState`, `info`, `errorPage`, `isErrorPage` eta `contenttype`. Hona hemen adibide bat:

```
<%@ page language = "java" import = "java.util.* "  %>
```

Agindu horrekin, Java lengoaian idatzitako pakete bat txertatu nahi dugu JSP orrian.

Txertaketa-aginduak fitxategi bat txertatzen du JSP orrian itzulpen-fasean, eta *etiketen liburutegiaren aginduak* URL (Uniform Resource Locator) bat definitzen du adierazteko non dauden JSP orrian erabiliko diren bezeroaren etiketa pertsonalizatuak.

Etiketak, JSP orri baten bigarren osagai mota dira eta oso erabiliak dira aplikazioaren aurkezpena eta logika bereizteko. Etiketak erabiliz erraz txertatzen dira JavaBean osagaiak eta JavaBean osagai baten ezaugarriak, hau da, domeinuaren logika aplikazioaren aurkezpenean, JSP orrian alegia. Etiketekin applet bati ere dei diezaiokegu eta bezeroaren eskaera berriz helbideratu dezakegu HTML orri batera, beste JSP orri batera edo servlet batera.

Domeinuaren logika tratatzeko hiru etiketa mota daude eta hona hemen bakoitzaren sintaxia eta adibide bat:

- `<jsp:useBean id = "javaBeanInstantziaIzena" scope = { page, request, session, application} class= "paketeKlasea" />`

  Etiketa horrekin, JavaBean bat instantziatzen dugu eta beraren hedapena zein izango den adierazten dugu.

- `<jsp:setProperty name = "javaBeanInstantziaIzena property = "ezaugarriIzena" value = "balioa" />`

  Etiketa horrekin, balio bat esleitzen zaio JavaBean baten ezaugarri bati. JavaBean osoari esleitu nahi badizkiogu balio lehenetsiak, nahikoa da: `property = "*"` balioa, `property` atributuan jartzea.

- `< jsp:getProperty name = "javaBeanInstantziaIzena property = "ezaugarriIzena" / >`

  Etiketa hau, aurrekoaren ahaidea da eta JavaBeanaren ezaugarri baten balioa eskuratzeko balioko du.

Lehenago esan dugu etiketa batekin applet bat egikaritu dezakegula edo bezeroaren eskaera berriz helbideratu. Bata eta bestea egiteko: `<jsp:plugin>` eta `<jsp:forward>` etiketak erabiliko ditugu.

`<jsp:plugin>` etiketa erabilita applet bat kargatu eta egikaritu egiten du bezeroak nabigatzailean beharrezkoa den HTML kodea eraikita.

`<jsp:forward>` etiketak bezeroaren eskaera berriz helbideratu egiten du beste norabait. Hona hemen etiketaren sintaxia:

```
<jsp:forward page = "URLHelbidea" />
```

JSP orri baten hirugarren eta azken osagai mota *scripting osagaiak* dira. Osagai mota honekin Java kodea txerta dezakegu JSP orri batean eta bezeroak orria eskatzen duenean egikaritzeko eskaeraren prozesaketa-fasean alegia, idatzi dugun kodea egikaritu egingo da kodetu dugun ordenan. Scripting osagaiak hiru eratakoak izan daitezke: scriptlet bat, adierazpen bat edo erazagupen bat.

Scriptlet bat `<%` eta `%>` artean kokatzen dugun Java kodea izango da. Kode zati hori itzulpen-fasean eraikiko den servletaren `_jspService()` metodoan txertatuko da. Gogoratu ezen metodo horrek interpretatu egiten duela bezeroarengandik jasotako eskaera, bidalitako eskaera, HTTP metodotzat harturik.

Adierazpen batekin, balioak itzultzen dizkiogu bezeroari HTML kodean. Adierazpen baten sintaxia honako hau da: `<% = adierazpena %>` eta hona hemen adibide bat:

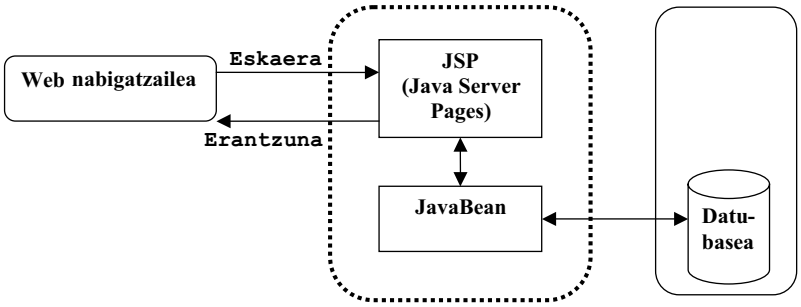
```
Proba kopurua : <% =zenbigarri.getprobaKopurua() %> proba
```

Adierazpen bat erabilita `zenbigarri` osagaiaren `getprobaKopurua()` metodoa egikaritzen da eta itzultzen duen balioa HTML orrian txertatzen da bezeroari erantzun gisa bidaltzeko.

Erazagupen batekin aldagaiak eta metodoak erazagutzen dira. Metodo eta aldagai horiek JSP orria inplementatzen duen klasean txertatuko dira. Hona hemen erazagupen baten sintaxia: `<%! erazagupena %>` eta hona hemen adibide bat:

```
<%! int kopurua = 0; %>
<%! private String lortuKopuru ()
       {  return Kopuru orokorra : + kopurua; }
%>
```

Web aplikazio bat eraikitzean JSP teknologia erabilita bi eredu mota erabili ohi dira: lehenengo eredu motan[27], JSP orriak zuzenean prozesatzen ditu bezeroaren eskaerak eta bidali behar dituen erantzunak. Eredu horretan, banaketa bat gertatzen da aplikazioaren aurkezpenaren eta edukiaren artean. Aplikazioaren aurkezpena JSP orria erabilita gauzatzen da eta datuen kudeaketa JavaBean bat erabilita egituratzen da. Datuak atzitu behar dituenean, JavaBeana erabiltzen du JSP orriak bitartekari-lanak gauzatzeko. Horregatik software-arkitektura horri bi mailako arkitektura deitu ohi izan zaio. **3.19. irudia**k bi mailako arkitekturaren osagai nagusien arteko erlazioak islatzen ditu. Arkitektura mota hori aplikazio txikietan erabiltzen da. Aplikazioa handia eta konplexua bihurtzen denean, ez da komeni arkitektura hori erabiltzea.



3.19. irudia. Bi mailako arkitektura (Model 1).

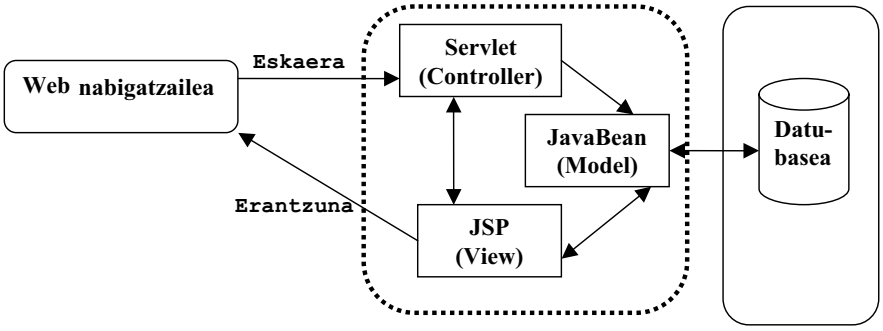
Aplikazio konplexuetan bi mailako arkitektura erabiltzen bada, JSP orria scripting osagaiez betetzen hasten da eta hainbat Java kode pilatzen hasten da JSP orrian. Horrela, orriaren irakurgarritasuna zaila bihurtzen da eta programatzaileari nekagarria egingo zaio aplikazioa hedatzea eta mantentzea. Arazoa areagotu egiten da JSP orrien kudeaketa web orrien diseinatzaileen esku badago eta horixe izaten da ohiko egoera proiektu handietan, partaide bakoitzaren lana hainbat roletan banatuta egoten baita.

Bigarren eredu motak, lehenengoaren gabeziak konpontzen ditu. Horretarako arkitektura zertxobait nahasten da baina osagai bakoitzaren ardurak banatuta mantentzen dira. Horrela eraikitako softwarea askoz ere eramangarriagoa eta hedagarriagoa izango da eta, aldi berean, bere mantenua asko erraztuko da. Software-arkitektura mota horri hiru mailako arkitektura deitu ohi zaio. **3.20. irudia**k

27. Sun etxearen dokumentazio eta espezifikazioetan lehenengo eredu mota `Model 1` izendatzen da eta bigarren eredu mota, `Model 2`.

azaltzen du non kokatzen diren hiru mailako arkitekturaren osagai nagusiak eta osagai bakoitza zeinekin eta nola jartzen den harremanetan.

Irudian ikusten den legez, arkitektura mota horrek servletak eta JSP orriak erabiltzen ditu batera. JSP orria, aurkezpen mailaz arduratzen da batez ere bezeroari erantzuna bidali behar zaionean. Servletak, bere aldetik, kontrolatzaile lana gauzatuko du eta bera arduratuko da bezeroaren eskaera prozesatzeaz eta beharrezkoak diren JavaBean eta objektuak eraikitzeaz. Horrela, domeinuaren logika ez da JSP orrian egongo eta prozesuaren logika osoa servletaren ardurapean egongo da banaketa funtzionala arkitektura osoari aplikatuta. Horrela antolatutako hiru mailako arkitekturak MVC (Model-View-Control/Eredu-Bista-Kontrol) diseinu-patroi ezagunari jarraitzen dio orpoz orpo.



3.20. irudia. Hiru mailako arkitektura (Model 2).

Web aplikazioaren egitura hiru arlotan banatzen da: *domeinuaren logika*[28], *aurkezpena* eta bezeroaren *eskaeraren prozesaketa*.

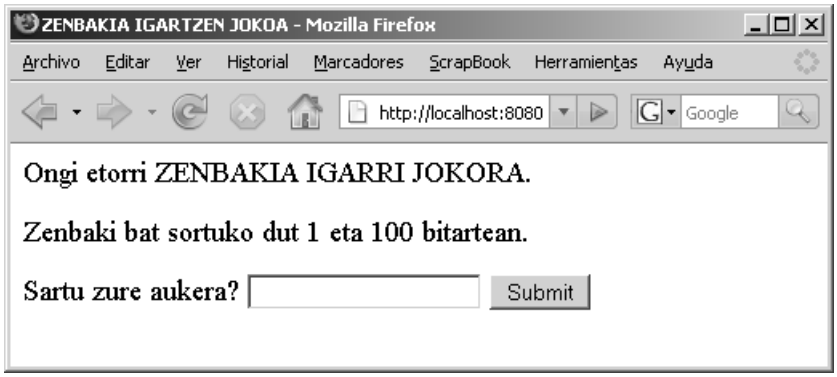
Aplikazioaren datuak kudeatzen ditu domeinuaren logikak eta aplikazioaren zatirik egonkorrena izango da. Datuak nola ikusi eta bezeroari nola aurkeztuko zaizkion, aurkezpen-arloaren ardura izango da eta bezeroaren eskaeraren prozesaketak domeinuaren logika eta aurkezpena lotuko ditu. **3.20. irudia**n ikusten den legez, hiru mailako arkitekturak ederki asko betetzen ditu aipatutako hiru zatiak.

Web aplikazioa horrela antolatzeak abantailak eskaintzen ditu etorkizunean softwarea eguneratu, aldatu eta mantendu nahi denean. Gogoratu ezen aplikazioaren datuen egitura eta logika egonkorrenak bihurtzen direla denboran zehar eta gutxi aldatuko direla. Ezin dugu gauza bera esan aurkezpen mailaz, zeina askotan bezeroaren eraginez edota teknologiaren bilakaerak bultzatuta, aplikazioaren bizizikloan behin eta berriz aldatu behar izaten baita; izan ere, aplikazioaren zatirik ezegonkorrena baita.

28. *Business logic* ingelesezko softwarearen ingeniaritzaren literaturan.

Aplikazioa ondo egituratu badugu, eta hiru mailako arkitektura erabiltzea bide egokia izan daiteke helburu hori lortzeko, ez da zaila izango aipatutako aurkezpen mailan eguneraketak gauzatzea aplikazioaren zati nagusiak ukitu gabe. Horra hor, hiru mailako arkitekturaren eta MVC (Model-View-Control/Eredu-Bista-Kontrol) diseinu-patroiaren balioa softwarearen eraikuntzan.

JSP orrien sarrera hau bukatzeko eta egin dugun azalpen teorikoa argitzeko bi adibide aztertuko ditugu hurrengo orrialdeetan. Lehenengo adibidean bi mailako arkitekturan (**3.19. irudia**) oinarritutako web aplikazio bat eraikiko dugu; bigarrenean, aldiz, **3.20. irudia**n ikusten den hiru mailako arkitektura izango dugu aztergai. Has gaitezen, bada, bi mailako arkitekturaren adibidearen nondik norakoak aztertzen.

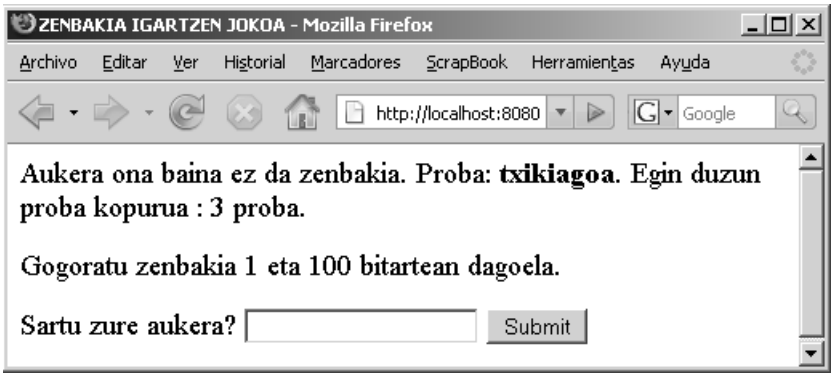


3.21. irudia. Adibidearen hasierako interfaze grafikoa.

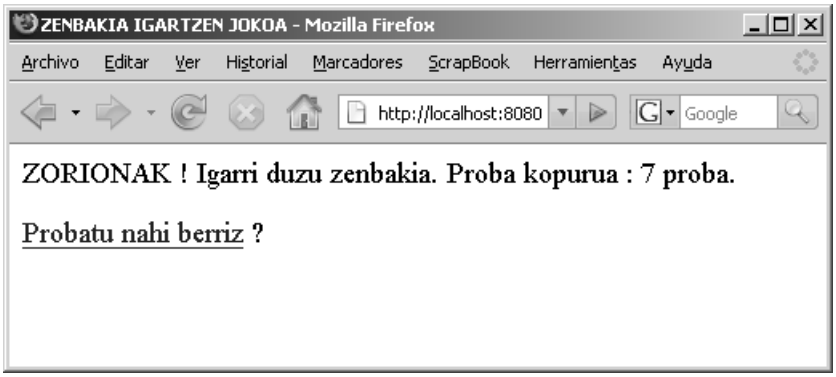
Adibideak JSP orri bat (**3.21. irudia**) aurkeztuko dio aplikazioaren hasieran erabiltzaileari, adieraziz joko bat proposatzen diola eta zenbaki bat sartu behar duela.

Zenbakiak 1etik 100era bitartean egon behar du eta zenbaki osoa izan behar du. Erabiltzaileak sartzen duen zenbakia alderatu egingo du aplikazio informatikoak berak sortu duenarekin eta mezu (**3.22. irudia**) bat eraikiko du HTML formatuan adieraziz ordenagailuak sortu duen zenbakia handiagoa ala txikiagoa den, eta zenbat proba gauzatu dituen zenbakia igarri nahian.

Zenbakia igarri eta gero, azken mezua (**3.23. irudia**) agertuko zaio erabiltzaileari zorionak emanez eta ahalegin kopurua adieraziz.



3.22. irudia. Erabiltzailea informatzeko interfaze grafikoa.



3.23. irudia. Adibidearen azken interfaze grafikoa.

Adibidearen betekizunak eta interfaze grafikoak aurkeztuta, has gaitezen web aplikazioa diseinatzen bi mailako arkitektura erabilita. Arkitekturak adierazten duen bezala, JSP orria izango da eskaerak eta erantzunak kudeatuko dituena, hau da, aurkezpen maila eta bezeroaren eskaeraren prozesaketa, bere ardura izango dira. Bestalde, datuen egitura eta logika kudeatzeko, JavaBean bat erabiliko da. Hona hemen nola programatu dugun aipatutako JavaBeana:

```
package zenbaki;

import java.util.*;

public class ZenbakiaIgarriBean {

  int erantzuna;
  boolean arrakasta;
  String aholkua;
  int probaKopurua;

  public ZenbakiaIgarriBean() {
```

```
    hasieratu();
  }

  public void setProba(String proba) {
    probaKopurua++;

    int balioa;
    try {
      balioa = Integer.parseInt(proba);
    }
    catch (NumberFormatException e) {
      balioa = -1;
    }

    if (balioa == erantzuna) {
      arrakasta = true;
    }
    else if (balioa == -1) {
      aholkua = "hurrengoan zenbaki batekin, mesedez";
    }
    else if (balioa < erantzuna) {
      aholkua = "handiagoa";
    }
    else if (balioa > erantzuna) {
      aholkua = "txikiagoa";
    }
  }

  public boolean getarrakasta() {
    return arrakasta;
  }

  public String getaholkua() {
    return "" + aholkua;
  }

  public int getprobaKopurua() {
    return probaKopurua;
  }

  public void hasieratu() {
    erantzuna = Math.abs(new Random().nextInt() % 100) + 1;
    arrakasta = false;
    probaKopurua = 0;
  }   }
```

JavaBeana instantziatu eta gero, metodo eraikitzaileak aldagaiak hasieratzen ditu: a) igarri behar den zenbakia (`erantzuna` aldagaia) sortuz; b) `arrakasta` aldagai boolearrari `false` balioa esleituz, eta, c) proba-kontagailua (`probakopurua` aldagaia) zero balioarekin hasieratuz. Hori egin ondoren, JavaBeana prest dago

mezuak jasotzeko eta bere metodoak egikaritzeko. Gelditzen zaigun gauza bakarra da JSP orriari adieraztea JavaBeana erabiliko dugula orria egikaritzeko. Horretarako, JSP orrian agindu eta etiketa egokiak txertatu behar ditugu orriak jakin dezan non bilatu behar duen JavaBeana eta nola izango den identifikatua orrian zehar:

```
<%@ page import = "zenbaki.ZenbakiaIgarriBean" %>

<jsp:useBean id="zenbigarri" class="zenbaki.ZenbakiaIgarriBean"
scope="session"/>
<jsp:setProperty name="zenbigarri" property="*"/>
```

JSP orriaren eta JavaBeanaren kateatzea zehaztu ondoren, JSP orriaren semantika nagusia programatzen da, aipatutako interfaze grafikoak eraikitzeko:

```
<html>
<head><title>ZENBAKIA IGARTZEN JOKOA</title></head>
<body bgcolor="white">
<font size=4>

<% if (zenbigarri.getarrakasta()) { %>

  ZORIONAK !  Igarri duzu zenbakia.
  Proba kopurua : <%= zenbigarri.getprobaKopurua() %> proba.<p>

  <% zenbigarri.hasieratu(); %>

  <a href="zenbaki.jsp">Probatu nahi berriz</a>  ?

<% } else if (zenbigarri.getprobaKopurua() == 0) { %>

  Ongi etorri ZENBAKIA IGARRI JOKORA.<p>

  Zenbaki bat sortuko dut 1 eta 100 bitartean.<p>

  <form method=get>
  Sartu zure aukera? <input type=text name=proba>
  <input type=submit value="Submit">
  </form>

<% } else { %>

  Aukera ona baina ez da zenbakia.  Proba: <b><%=
zenbigarri.getaholkua() %></b>.

  Egin duzun proba kopurua : <%= zenbigarri.getprobaKopurua() %>
proba.<p>

  Gogoratu zenbakia 1 eta 100 bitartean dagoela.<p>

  <form method=get>
  Sartu zure aukera? <input type=text name=proba>
```

```
  <input type=submit value="Submit">
  </form>

<% } %>

</font>
</body>
</html>
```

Konturatu ezen orri guztia baldin habiaratu handi bat besterik ez dela. Baldin bakoitzaren adarkatzea erabakitzeko, hasieran eraikitako JavaBeanaren metodoak egikaritzen dira. Hala, `getarrakasta()` metodoak `true` itzuliko du erabiltzaileak sartu duen zenbakia eta ordenagailuak sortu duena bat badatoz. Hori horrela bada, JSP edukiontziak erantzuna itzuliko dio bezeroari HTML formatuan, zorionak emanez eta ahalegin kopurua adieraziz:

```
  <% if (zenbigarri.getarrakasta()) { %>
  ZORIONAK !  Igarri duzu zenbakia.
  Proba kopurua : <%= zenbigarri.getprobaKopurua() %> proba.<p>
  <% zenbigarri.hasieratu(); %>
  <a href="zenbaki.jsp">Probatu nahi berriz</a>  ?
```

Erabiltzaileak arrakastarik izan ez badu, baldin guztien azken adarkatzea egikarituko da eta erantzuna itzuliko dio JSP edukiontziak bezeroari HTML formatuan, adieraziz, ordenagailuak sortu duen zenbakia txikiagoa ala handiagoa den:

```
   Aukera ona baina ez da zenbakia.  Proba: <b><%=
zenbigarri.getaholkua() %></b>.
  Egin duzun proba kopurua : <%= zenbigarri.getprobaKopurua() %>
proba.<p>

  Gogoratu zenbakia 1 eta 100 bitartean dagoela.<p>
  <form method=get>
  Sartu zure aukera? <input type=text name=proba>
  <input type=submit value="Submit">
  </form>
```

Jokoaren hasieran bagaude zero balioa itzuliko du `getprobaKopurua()` metodoak, eta JSP edukiontziak, jokoaren aurkezpenaren informazioa itzuliko dio bezeroari erantzun gisa:

```
if (zenbigarri.getprobaKopurua() == 0) { %>
  Ongi etorri ZENBAKIA IGARRI JOKORA.<p>

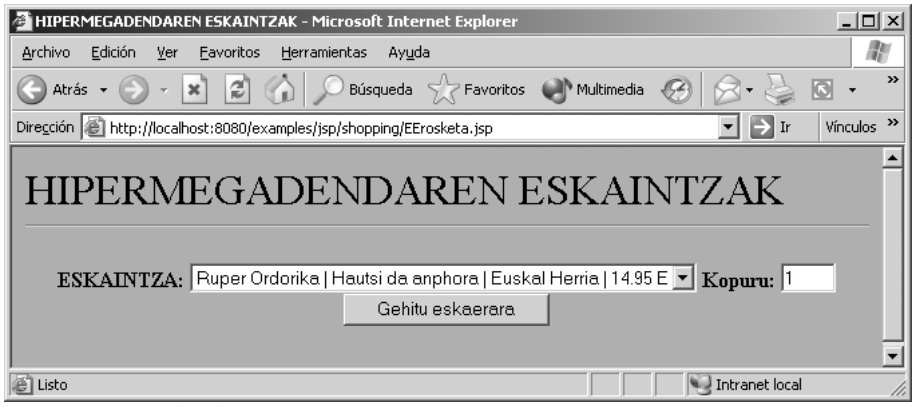
  Zenbaki bat sortuko dut 1 eta 100 bitartean.<p>

  <form method=get>
  Sartu zure aukera? <input type=text name=proba>
  <input type=submit value="Submit">
  </form>
```

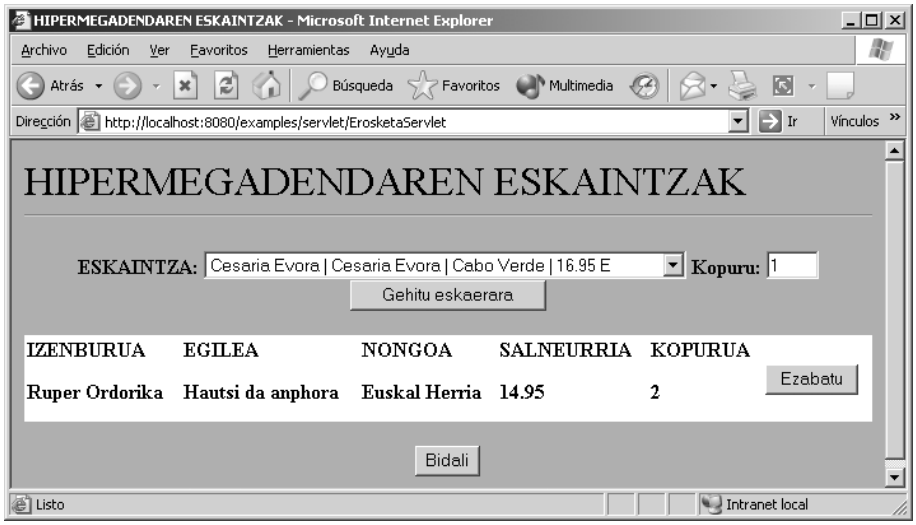
JSP orriaren kodea aztertzen badugu, hainbat scripting osagai txertatu ditugu scriptlet edota adierazpen gisa. Aurreko orrialdeetan esan dugun legez, Java kode hori guztia, itzulpen-fasean eraikiko den servletaren `_jspService()` metodoan txertatuko da automatikoki eta gardena izango da programatzailearentzat. Hona hemen nola txertatzen duen JSP edukiontziak aipatutako Java kodea:

```
if(_jspx_specialzenbigarri == true) { }
out.print(_jspx_html_data[2]);
JspRuntimeLibrary.introspect(zenbigarri, request);
out.print(_jspx_html_data[3]);
if (zenbigarri.getarrakasta()) {
out.print(_jspx_html_data[4]);
out.print( zenbigarri.getprobaKopurua() );
out.print(_jspx_html_data[5]);
zenbigarri.hasieratu();
out.print(_jspx_html_data[6]);
} else if (zenbigarri.getprobaKopurua() == 0) {
out.print(_jspx_html_data[7]);
} else {
out.print(_jspx_html_data[8]);
out.print( zenbigarri.getaholkua() );
out.print(_jspx_html_data[9]);
out.print( zenbigarri.getprobaKopurua() );
out.print(_jspx_html_data[10]);
}
out.print(_jspx_html_data[11]);
        } catch (Throwable t) {
            if (out.getBufferSize() != 0)
                out.clear();
            throw new JspException("Unknown exception: ", t);
        } finally {
            out.flush();
            _jspxFactory.releasePageContext(pageContext);
```

Bi mailako arkitekturan, erraz diseinatu eta programatzen da web aplikazio bat, batez ere, ez denean handia eta konplexua, zeren JSP orriak egiten baitu lan nagusia. JSP orria arduratzen da aplikazioaren aurkezpenaz eta kontrola kudeatzeaz. Baina, aldi berean, azaldu dugun adibideak argi uzten ditu bi mailako arkitekturaren mugak eta ezintasunak. JSP orriaren irakurgarritasuna zeharo aldebresten da scripting osagaiz betetzen eta handitzen hasten denean. Programatzaileek eta web orrien diseinatzaileek zailtasunak izaten dituzte softwarea hedatzeko eta mantentzeko. Aplikazioak jokaera hori hartzen duenean, askoz ere eraginkorragoa izango da hiru mailako arkitektura erabiltzea, bigarren adibidean ikusiko dugun legez.



3.24. irudia. Adibidearen hasierako pantaila.



3.25. irudia. Eskaera eguneratuaren interfaze grafikoa.

Bigarren adibideak (**3.24. irudia**) salerosketa elektroniko bat inplementatuko du web aplikazio bat erabilita. Erosleari, hau da, aplikazioaren bezeroari, lehenengo eta behin, eskaintzak agertzen zaizkio pantailan eta aukera izango du aldiko, eskaintza bat aukeratzeko eta bere kopurua adierazteko. Hori egin eta gero, aukeratutako produktua eskaerari gehitzen zaio. Dena ondo joan bada, eskaeraren egoera[29] eguneratuta (**3.25. irudia**) azalduko da pantailan. Halaber, edozein unetan aukera izango du erosleak eskaerako produktu bat ezabatzeko.

29. Aukeratutako produktua lehendik aukeratua izan bada, ez da beste lerro bat sortzen eskaeran, kopurua eguneratzen da eta eskaera bere horretan geratzen da.

Eskaera amaitu eta gero, *Bidali* botoia zapalduko du erosleak erosketa-prozesua amaitzeko. Dena ondo joan bada, beste pantaila bat agertuko zaio ordaindu behar duena adierazteko.

Honaino, web aplikazioaren betekizunak zeintzuk izango diren; has gaitezen aplikazioa programatzen. Aitzinean esan dugun bezala, aplikazioa programatzeko hiru mailako arkitektura estandarra erabiliko dugu, eta horretarako aurkezpen maila, JSP orriak erabilita kudeatuko dugu. Aplikazioaren logika servlet batean kapsulatuko dugu, eta, azkenez, datuen egitura gordetzeko eta kudeatzeko, JavaBean bat eraikiko dugu.

Beraz, argi eta garbi ikusten da gure hiru mailako arkitekturak esplizituki bereizten dituela aplikazioaren hiru zati funtzionalak. Horrela, nahiz eta aplikazioaren programazioa luzatu eta zertxobait zaildu hasiera batean, etorkizunean, hedapena eta mantenua erraz kudeatuko dira, aplikazioaren alde funtzional bakoitza erabat kapsulatuta baitago.

Has gaitezen, horrenbestez, aplikazioaren aurkezpen maila aztertzen. Aplikazioaren hasierako JSP orria (`Eerosketa.jsp`), betekizunetan esan dugun legez, produktuak aukeratzeko erabiliko du aplikazioaren bezeroak. Horretarako, zerrenda (`select`) bat programatuko dugu hainbat aukerarekin (`option`). Hona hemen nola gelditu litzatekeen JSP orria funtzionalitate guztiak programatuta:

```
<%@ page session="true" %>
<html>
<head>
  <title>HIPERMEGADENDAREN ESKAINTZAK</title>
</head>
<body bgcolor="#33CCFF">
  <font face="Times New Roman,Times" size="+3">
    HIPERMEGADENDAREN ESKAINTZAK
  </font>
  <hr><p>
  <center>
  <form name="shoppingForm"
    action="/examples/servlet/ErosketaServlet"
    method="POST">
  <b>ESKAINTZA: </b>
  <select name=CD>
  <option>Ruper Ordorika | Hautsi da anphora | Euskal Herria | 14.95
E</option>
    <option>EKM| Gogoz eta asmoz | Euskal Herria| 16.95 E</option>
    <option>Kaira | Tounami Diabate| Mali | 16.95 E</option>
    <option>Shakuhachi| Kohachiro Miyata | Japonia | 13.95
E</option>
    <option>Dance the Devil Away | Outback | Australia | 14.95
E</option>
```

```
    <option>Record of Changes | Samulnori | Korea | 12.95
E</option>
    <option>Djelika | Tounami Diabate | Mali | 14.95 E</option>
    <option>Rapture | Nusrat Fateh Ali Khan | Pakistan | 12.95
E</option>
    <option>Cesaria Evora | Cesaria Evora | Cabo Verde | 16.95
E</option>
    <option>Ibuki | Kodo | Japonia | 13.95 E</option>
  </select>
  <b>Kopuru: </b><input type="text" name="kopuru" SIZE="3" value=1>
  <input type="hidden" name="ekintza" value="GEHITU">
  <input type="submit" name="Submit" value="Gehitu eskaerara">
  </form>
  </center>
  <p>
  <jsp:include page="Erosketa.jsp" flush="true" />
</body>
</html>
```

Konturatu ezen orriak `POST` mezu bat bidaltzen diola aplikazioaren logika kontrolatuko duen servletari (`ErosketaServlet`). Halaber, orriaren bukaeran `jsp:include` etiketa bat programatu dugu beste JSP orri bat txertatzeko hasierako orrian. Bigarren JSP orri (`Erosketa.jsp`) horrekin, eskaeraren egoera bistaratuko da une oro pantaila nagusian. Eskaeraren egoera, geroago ikusiko dugun legez, servletak eguneratuko du bektore bat erabilita, eta bigarren JSP orri horrek, sesio-aldagai bat erabilita, bektorearen balioak eskuratuko ditu eta hasierako orrian[30] txertatuko ditu. Hori egiteko honelako agindu eta scriptlet-ak erabiltzen dira:

```
<%@ page session="true" import="java.util.*, shopping.CD" %>
<%
  Vector EskaZerrenda = (Vector) session.getValue("eerosketa");
  if (EskaZerrenda != null && (EskaZerrenda.size() >0)) {
%>
```

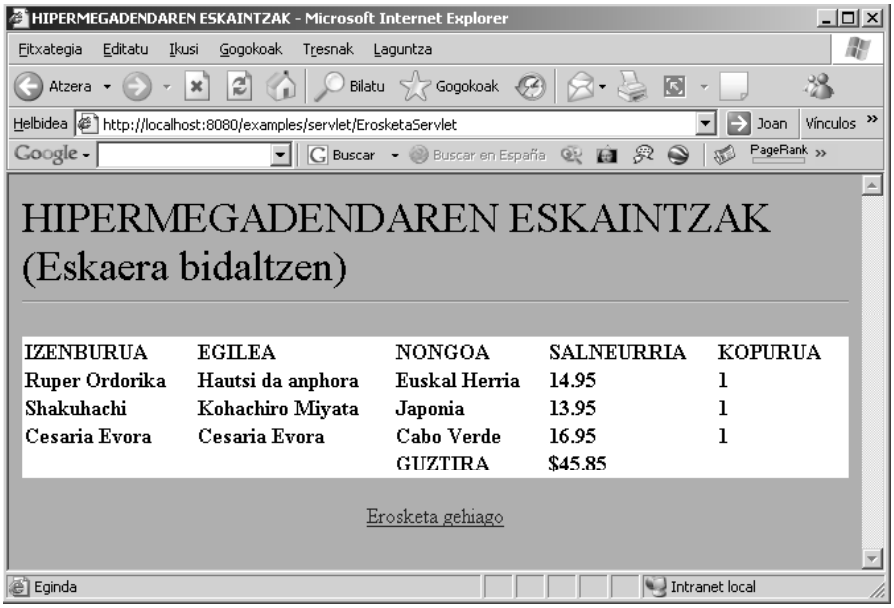
Eskaeraren balioak bektorean jaso eta gero, goiburuak jarri eta HTML kodea eraikitzen da scripting osagaiak erabilita. Hona hemen `Erosketa` JSP orriaren kode osoa:

```
<%@ page session="true" import="java.util.*, shopping.CD" %>
<%
  Vector EskaZerrenda = (Vector) session.getValue("eerosketa");
  if (EskaZerrenda != null && (EskaZerrenda.size() >0)) {
%>

  <center>
  <table border="0" cellpadding="0" width="100%" bgcolor="#FFFFFF">
```

30. Berez bigarren JSP orrian jasotzen dira eguneratutako balioak baina bigarren JSP orri hori hasierako JSP orrian txertatuta dagoenez, erabiltzaileak hasierako orrian ikusiko ditu eskaeraren balioak.

```
    <tr>
      <td><b>IZENBURUA</b></td>
      <td><b>EGILEA</b></td>
      <td><b>NONGOA</b></td>
      <td><b>SALNEURRIA</b></td>
      <td><b>KOPURUA</b></td>
      <td></td>
    </tr>
    <%
      for (int index=0; index< EskaZerrenda.size();index++) {
        CD eskaeraBat = (CD) EskaZerrenda.elementAt(index);
    %>
    <tr>
      <td><b><%= eskaeraBat.getIzena() %></b></td>
      <td><b><%= eskaeraBat.getEgilea() %></b></td>
      <td><b><%= eskaeraBat.getNongoa() %></b></td>
      <td><b><%= eskaeraBat.getSalneurria() %></b></td>
      <td><b><%= eskaeraBat.getKopurua() %></b></td>
      <td>
        <form name="deleteForm"
              action="/examples/servlet/EroskетaServlet"
              method="POST">
          <input type="submit" value="Ezabatu">
          <input type="hidden" name= "kenindex" value='<%= index %>'>
          <input type="hidden" name="ekintza"  value="KENDU">
         </form>
      </td>
    </tr>
    <% } %>
  </table>
  <p>
  <form name="checkoutForm"
    action="/examples/servlet/EroskетaServlet"
    method="POST">
    <input type="hidden" name="ekintza" value="EROSI">
    <input type="submit" name="Erosketa" value="Bidali">
  </form>
  </center>
<% } %>
```



3.26. irudia. Adibidearen azken interfaze grafikoa.

Aurkezpen maila amaitzeko, hirugarren eta azken JSP orria (`Erosi.jsp`) aurkeztuko dugu. Orri horrekin (**3.26. irudia**) eskaera amaitutzat emango du bezeroak eta aurreko orrian egin dugun bezala, lehenengo eta behin, sesio-aldagai bat erabilita eskaeraren balioak eskuratuko ditugu bektore batean, eta ondoren, scripting osagaiak erabilita, eskaeraren balioak txertatzen dira eraikitzen ari garen HTML kodean. Ordaindu beharreko kopuru orokorra, servletak kalkulatuko du eta JSP orriak `getAttribute()` metodoa erabiliko du eskuratzeko. Hona hemen nola gelditu litzatekeen JSP orria:

```
<%@ page session="true" import="java.util.*, shopping.CD" %>
<html>
<head>
  <title>HIPERMEGADENDAREN ESKAINTZAK</title>
</head>
<body bgcolor="#33CCFF">
  <font face="Times New Roman,Times" size=+3>
    HIPERMEGADENDAREN ESKAINTZAK (Eskaera bidaltzen)
  </font>
  <hr><p>
  <center>
  <table border="0" cellpadding="0" width="100%" bgcolor="#FFFFFF">
  <tr>
    <td><b>IZENBURUA</b></td>
    <td><b>EGILEA</b></td>
    <td><b>NONGOA</b></td>
    <td><b>SALNEURRIA</b></td>
```

```
    <td><b>KOPURUA</b></td>
    <td></td>
  </tr>
  <%
    Vector EskaZerrenda = (Vector)
    session.getValue("eerosketa");
    String kopuru = (String) request.getAttribute("kopurua");
    for (int i=0; i< EskaZerrenda.size();i++) {
      CD eskaeraBat = (CD) EskaZerrenda.elementAt(i);
  %>
  <tr>
    <td><b><%= eskaeraBat.getIzena() %></b></td>
    <td><b><%= eskaeraBat.getEgilea() %></b></td>
    <td><b><%= eskaeraBat.getNongoa() %></b></td>
    <td><b><%= eskaeraBat.getSalneurria() %></b></td>
    <td><b><%= eskaeraBat.getKopurua() %></b></td>
          </tr>
        <%
          }
    session.invalidate();
  %>
  <tr>
    <td>     </td>
    <td>     </td>
    <td><b>GUZTIRA</b></td>
    <td><b>$<%= kopuru %></b></td>
    <td>     </td>
  </tr>
  </table>
  <p>
  <a href="/examples/jsp/shopping/EErosketa.jsp">Erosketa gehiago</a>
  </center>
</body>
</html>
```

Hiru mailako arkitekturaren bigarren osagaia web aplikazioaren kontrolaz arduratuko da. Bigarren adibide honetan diseinu-erabaki sendo bat hartu dugu: argi eta garbi bereiztea aurkezpen maila eta kontrol maila aplikazioan. Horrela, JSP orriak aurkezpen mailaz arduratuko dira soilik, eta ahal denik eta Java kode gutxien txertatuko dugu irakurgarritasuna eta orrien mantenua ez zailtzeko.

Hortakotz, aplikazioaren kontrola kudeatzeko servlet (`ErosketaServlet`) bat erabiliko dugu. Servletak, JSP orrietan agertzen diren botoien ekintzak egikarituko ditu eta, horretarako, `doPost()` metodoan, `getParameter()` metodoa erabiliko du. Horrenbestez, ez gara asko harrituko servletaren semantika nagusia baldin habiaratu erraldoi bat dela jakitean:

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import shopping.CD;

public class ErosketaServlet extends HttpServlet {
  public void init(ServletConfig conf) throws ServletException  {
    super.init(conf);
  }
  public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
    HttpSession session = req.getSession(false);
if (session == null) {
res.sendRedirect("http://localhost:8080/errorea.html");
}
Vector EskaZerrenda=
(Vector)session.getValue("eerosketa");
String ekintza = req.getParameter("ekintza");
if (!ekintza.equals("EROSI")) {
if (ekintza.equals("KENDU")) {
String del = req.getParameter("kenindex");
int d = (new Integer(del)).intValue();
EskaZerrenda.removeElementAt(d);
} else if (ekintza.equals("GEHITU")) {
// cd-a lehenago erosita?
boolean bilatuta=false;
CD CDbat = getCD(req);
if (EskaZerrenda==null) {
// gehitu lehen cd-a saskira
EskaZerrenda = new Vector(); //lehen eskaera
EskaZerrenda.addElement(CDbat);
} else {  // ez da lehenengo erosketa
for (int i=0; i< EskaZerrenda.size(); i++) {
CD cd = (CD) EskaZerrenda.elementAt(i);
if (cd.getIzena().equals(CDbat.getIzena())) {
cd.setKopurua(cd.getKopurua()+CDbat.getKopurua());
EskaZerrenda.setElementAt(cd,i);
bilatuta = true;
} //if aginduaren amaiera
} // for amaiera
if (!bilatuta)
EskaZerrenda.addElement(CDbat);
}
}
session.putValue("eerosketa", EskaZerrenda);
String url="/jsp/shopping/EErosketa.jsp";
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(url);
rd.forward(req, res);
```

```
} else if (ekintza.equals("EROSI"))  {
float guztira =0;
for (int i=0; i< EskaZerrenda.size();i++) {
CD EskaeraBat = (CD) EskaZerrenda.elementAt(i);
float salneurri= EskaeraBat.getSalneurria();
int kopuru= EskaeraBat.getKopurua();
guztira += (salneurri * kopuru);
}
guztira += 0.005;
String kopuru = new Float(guztira).toString();
int n = kopuru.indexOf('.');
kopuru = kopuru.substring(0,n+3);
req.setAttribute("kopurua",kopuru);
String url="/jsp/shopping/Erosi.jsp";
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(url);
rd.forward(req,res);
                                                    }

  }
  private CD getCD(HttpServletRequest req) {
    // scriptlet batean neketsua eta konplexua zeharo
    String nireCd = req.getParameter("CD");
    String kop= req.getParameter("kopuru");
    StringTokenizer t = new StringTokenizer(nireCd,"|");
    String izena= t.nextToken();
    String egilea = t.nextToken();
    String nongoa= t.nextToken();
    String salneurria= t.nextToken();
    salneurria= salneurria.replace('E',  ').trim();
    CD cd = new CD();
    cd.setIzena(izena);
    cd.setEgilea(egilea);
    cd.setNongoa(nongoa);
    cd.setSalneurria((new Float(salneurria)).floatValue());
    cd.setKopurua((new Integer(kop)).intValue());
    return cd;
  }
}
```

Servletak `ekintza` parametroaren balioa aztertu eta gero, gauzatu beharreko prozesua egikaritzen du. Adibidez, *`Gehitu eskaerara`* botoia zapaldu izan badu erabiltzaileak hasierako JSP orrian, `ekintza` parametroak "`GEHITU`" `String`-balioa izango du eta honako kode hau egikarituko da servletean:

```
boolean bilatuta=false;
CD CDbat = getCD(req);
if (EskaZerrenda==null) {
// gehitu lehen cd-a saskira
    EskaZerrenda = new Vector(); //lehen eskaera
    EskaZerrenda.addElement(CDbat);
```

```
} else { // ez da lehenengo erosketa
for (int i=0; i< EskaZerrenda.size(); i++) {
     CD cd = (CD) EskaZerrenda.elementAt(i);
if (cd.getIzena().equals(CDbat.getIzena())) {
    cd.setKopurua(cd.getKopurua()+CDbat.getKopurua());
    EskaZerrenda.setElementAt(cd,i);
    bilatuta = true;
} //if aginduaren amaiera
} // for amaiera
if (!bilatuta)
    EskaZerrenda.addElement(CDbat);
}
   }
session.putValue("eerosketa", EskaZerrenda);
String url="/jsp/shopping/EErosketa.jsp";
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(url);
rd.forward(req, res);
```

Botoia aurreneko aldiz zapaldu bada, eskaeraren zerrenda gordetzeko bektorea eraikitzen da lehenengo eta behin, eta jarraian, aukeratutako produktua gehitzen zaio bektoreari. Hurrengo atzipenetan, egiaztatu egiten da aukeratutako produktua bektorean biltegiratu ote den. Biltegiratuta badago lehendik, kopuruak gehitzen dira eta produktua eguneratu egiten da kopuru berriarekin berriz biltegiratuz. Produktua berria bada, bektorean gehitzen dugu eta kito.

Produktua kudeatzeko, `CD` JavaBean klasea erabiltzen du web aplikazioak. Horrela, produktuaren edozein aldaketa erabat zentralizatua izango dugu JavaBean horretan eta ez dugu besterik ukitu beharko aplikazioan, produktuaren diseinuaren eguneraketa bat gauzatzeko, klase horrek kudeatzen baititu produktuaren atributu eta metodo atzitzaileak (`get()` eta `set()`). Hona hemen zein izango litzatekeen JavaBean horren iturburu-kodea:

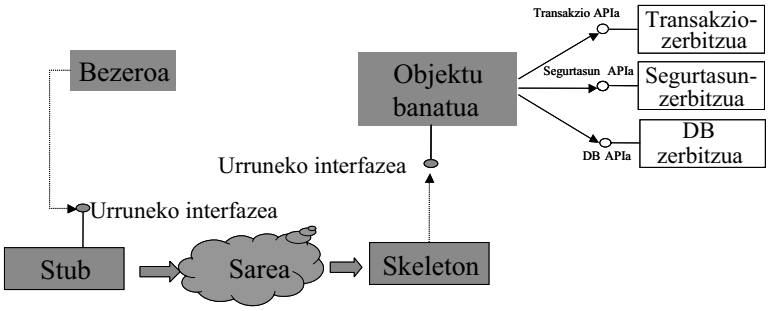
```
package shopping;
public class CD {
String izena;
String egilea;
String nongoa;
float salneurria;
int kopurua;
public CD() {
izena="";
egilea="";
nongoa="";
salneurria=0;
kopurua=0;
}
public void setIzena(String izen) {
```

```
       izena=izen;}
public String getIzena() {
         return izena;}
public void setEgilea(String egile) {
         egilea=egile;}
public String getEgilea() {
        return egilea;}
public void setNongoa(String nongo) {
        nongoa=nongo;}
public String getNongoa() {
         return nongoa;}
public void setSalneurria(float sal) {
          salneurria=sal;}
public float getSalneurria() {
         return salneurria;}
public void setKopurua(int kop) {
         kopurua=kop;}
public int getKopurua() {
         return kopurua;}
            }
```

3.3. J2EE TEKNOLOGIA

3.3.1. Sarrera

J2EE teknologia erabili badugu proiektu informatiko bat garatzeko, aplikazio informatikoaren EJB osagaiak banatuta egongo dira sarean zehar Javaren makina birtualetan kokatuta. Ingurune horretan, bezeroak objektu bat atzitu nahi badu, RMI APIaren ezaugarriak (**3.27. irudia**) ezagutu beharko ditu, objektuaren metodoak egikaritzeko. Laburki, honako prozesu hau gauzatzen da: bezeroak urruneko objektura atzitzeko urruneko interfaze bat erabili behar du lehendabizi. Urruneko interfaze horrekin, bezeroak bere aldean dagoen stub osagaiari deitzen dio, horrek sareko komunikazioa gauzatu dezan. Horrela, bezeroarentzat erabat gardena izango da komunikazio-prozesu guztia programatzea, bera arduratuko da objektuaz eta objektu horrekin egin dezakeenaz. Bezeroarentzat stub-a urruneko objektuaren proxy-a izango da. Komunikazioa ondo gauzatzeko, stub-ak kudeatzen duen deia saretik joango da urruneko zerbitzarira eta han errepikatu egingo da prozesua, hau da, urruneko objektuaren zerbitzariaren aldeko proxy (skeleton) bat erabiliko da sareko komunikazioaren arlo guztiak kudeatzeko.



3.27. irudia. RMI baliabidearen lan-fluxua.

Komunikazioa gauzatu ondoren, bezeroak urruneko objektuari egin dion deia, zerbitzarian dagoen objektu banatuari delegatzen dio skeleton objektuak. Egin beharrekoak eginda, skeleton objektuari pasatzen dio kontrola objektu banatuak eta skeleton objektuak, stub objektuari. Bukatzeko, stub objektuak bezeroari itzultzen dio berriz kontrola.

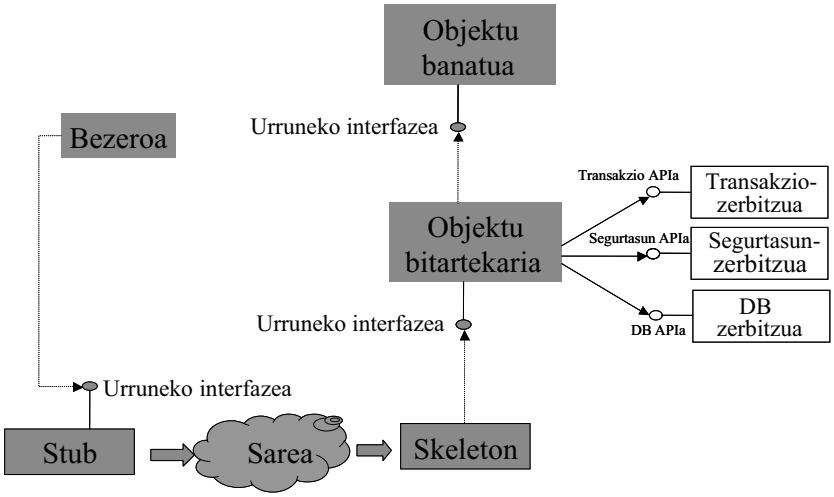
Aurkeztu dugun eskema operatiboak ondo funtzionatzen du objektu banatuak identifikatzeko eta egikaritzeko, hau da, bezeroak ez dizkionean eragiketa konplexuak eskatzen urruneko objektuari. Tamalez, hori gutxitan gertatzen da, eta gehienetan zerbitzu gehigarriak behar izaten dira zerbitzarian, besteak beste, transakzio- eta segurtasun-arazoak kudeatu nahi izaten dira.

Tradizionalki zerbitzu gehigarri horiek objektu banatu mailan gauzatu ohi dira eta domeinuaren logika "zikindua" gelditzen zen "logika gehigarri" horrekin. Argitu dezagun hori, adibide txiki baten bidez. Demagun, banku bateko aplikazio informatikoaren objektu banatu batek diru kopuru bat mugitu nahi duela kontu batetik beste batera. Objektu horrek, honelako eragiketak gauzatu beharko lituzke:

```
mugiKontu1Kontu2(Kontua kontu1, Kontua kontu2, int kopurua) {
      //1: Deitu erdiko mailako APIari segurtasuna kudeatzeko
      //2: Deitu erdiko mailako APIari  transakzioa hasieratzeko
//3: Deitu erdiko mailako APIari datu-basetik lerroa eskuratzeko
      //4 : Gauzatu kenketa kontu batean(Kontu1) eta gehitu
      //    diru kopurua beste kontuan(Kontu2)
//5:Deitu erdiko mailako APIari lerroa biltegiratzeko datu-basean
      //6: Deitu erdiko mailako APIari  transakzioa bukatzeko   }
```

Adibidean ikusten den legez, domeinuaren logika laugarren lerroan gauzatzen da bakarrik, beste eragiketa guztiak nahitaez txertatu behar izan ditugu hor, objektu banatuak kontrolatu behar baitu hiru mailako arkitekturaren erdiko mailaren funtzionalitatea. Argi ikusten da ezen domeinuaren logika gainkargatua dagoela objektu banatuan, eta, eragiketa arrunt bat gauzatzeko, hainbat iturburu-kode lerro idatzi behar dugu. Estrategia horrekin iturburu-kodea astun bihurtzen da eta zaildu egiten da bere mantenua. Hornitzaileak erdiko mailako APIa eguneratzen badu

aldaketak txertatuz APIan, aplikazio informatiko osoa berriz idatzi egin beharko dugu API berria erabiltzeko.



3.28. irudia. Urruneko deiak egikaritzeko J2EE teknologiaren estrategia.

J2EE teknologiaren diseinatzaileek, aipatutako eragozpenak saihesteko, beste estrategia bat[31] bat proposatu zuten: objektu bitartekariak (**3.28. irudia**) txertatzea objektu banatuaren eta skeleton objektuen artean. Horrela, objektu banatua domeinuaren logikaz arduratuko da eta segurtasuna, iraunkortasuna eta transakzio-arazoak objektu bitartekariaren esku gelditukо dira. Bistan da mantenua eta kodearen kudeaketa erabat errazten direla egoera berri honetan. Adibidez, aipatutako banku-eragiketa honela gelditukо litzateke eszenatoki berri honetan:

```
mugiKontu1Kontu2(Kontua kontu1, Kontua kontu2, int kopurua) {
      //1 : Gauzatu kenketa kontu batean(Kontu1) eta gehitu
      //     diru kopurua beste kontuan(Kontu2)
      }
```

Objektu bitartekariak bezeroaren deia jasotzen du eta beharrezkoak diren erdiko mailako eragiketak gauzatzen ditu. Eragiketa horiek erabat gardenak dira objektu banatuarentzat eta, horrela, objektu banatua, domeinuaren logikak eskatzen dizkion eragiketez arduratzen da bakarrik.

3.3.2. EJB osagaiak garatzen J2EE teknologia erabilita

Aplikazio informatikoaren osagaiak eraikitzea EJB beanak erabilita ez da eragiketa arrunta orain arte ikusi dugun bezala, hau da, ez da soilik Java fitxategi

31. Estrategia hori, osagaietan oinarritutako teknologia gehienek ere aplikatzen dute, besteak beste, CORBA, Microsoft .NET eta Component Model teknologiek.

batekin konpontzen. Jakina fitxategi bat beharko duguna beana gordetzeko, baina objektu bitartekarien semantika ere adierazi egin beharko da beste fitxategi batzuk erabilita. Fitxategi horiek EJB edukiontziak kudeatuko ditu zerbitzariarekin batera. EJB edukiontzia gure laguntzaile ikusezina izango da softwarearen eraikuntzan. Isil-isilik, baina era eraginkorrean, bitartekari-lanak gauzatuko ditu. Huraxe arduratuko da bezeroaren deiei ongi etorria emateaz eta bean egokia bilatzeaz deia egikaritzeko.

Beraz, prozesu guztiak bi adar izango ditu elkarri lotuta: batean, domeinuaren logika programatuko dugu bean mota bat programatuz. Bean mota horrek `javax.ejeb.EnterpriseBean` interfazea inplementatzen du baina ez zuzenean. Lehenengo, hautaketa egiten da zein motatako beana eraiki nahi dugun zehaztuta eta gero inplementatu egiten da eraiki nahi dugun bean mota horren interfazea (**3.30. irudia**). Adibidez, sesio-bean[32] bat eraiki nahi badugu honako hau jarri beharko genuke klasearen hasieran:

```
public KaixoBean inplements javax.ejb.SessionBean { ........}
```

Bezeroaren deia kudeatzeko, programatzaileak informazio zehatzez hornitu behar du EJB edukiontzia. Informazio zehatz horretan adieraziko denak aipatutako objektu bitartekariaren lana eratzeko eta antolatzeko balio izango dio edukiontziari. **3.29. irudia**k ulertzen lagunduko digu azaldu nahi duguna. Aitzinean aipatu dugun legez, bezeroak urruneko objektuaren metodo bat egikaritu nahi duenean urruneko interfaze bat erabiltzen du deia bideratzeko saretik zehar. Stub eta skeleton objektuek deia egokitu egiten dute sarean zehar garraiatzeko marshall eta demarshall eragiketak gauzatuta. EJB edukiontzira heltzen denean, ez da zuzenzuzenean bean egoki batera bideratzen, beanak gauzatu dezan metodoak eskatzen dion domeinuaren logika zehatza. Horren ordez, lehenengo, deia aztertzen du edukiontziak eta beharrezko zerbitzuak (segurtasuna, iraunkortasuna, osagaien bizi-zikloa, transakzioa, baliabideen kudeaketa...) egikaritzen ditu. Eragiketa hori gauzatzeko, `EJBObject` motako objektu bat erabiliko du edukiontziak, aipatutako objektu bitartekariaren lana gauzatuko duena. Bezeroaren deia zerbitzarian kudeatzeko, interfaze bat sortu behar du programatzaileak, `EJBObject` interfazea hedatzen duena. Fitxategi horretan beanak bezeroei eskaintzen dizkien metodoen signaturak adierazten dira eta metodo bakoitzak urruneko dei bat egikaritu behar duenez, `RemoteException` salbuespena altxa dezake. Aipatutako EJB objektuaren hainbat metodo, EJB edukiontziak inplementatuko ditu, zuzenean eta era gardenean programatzailearentzat.

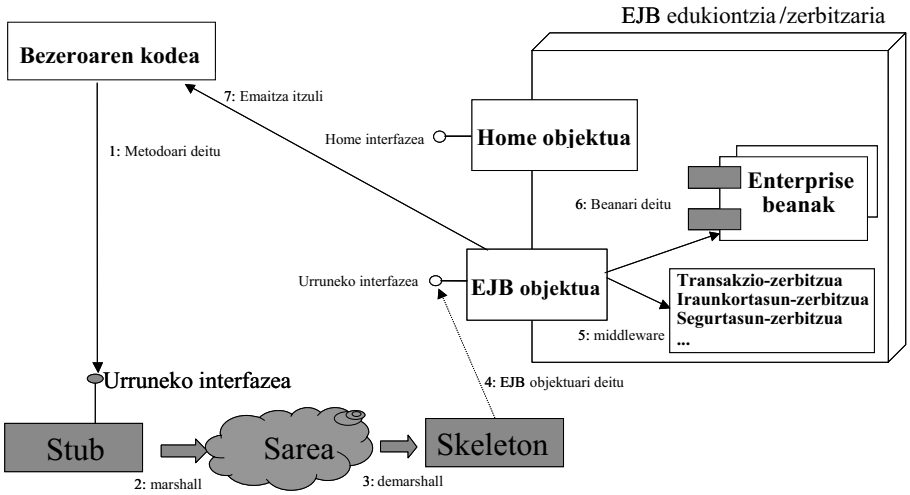
32. Entitate-bean bat eraiki nahi badugu honako interfaze hau inplementatu behar da: `javax.ejb.EntityBean`.

Aldiz, inplementatu nahi dugun beana mezuak bideratutako beana bada, beste interfaze hau inplementatzen da: `javax.ejb.MessageDrivenBean`

Errepara ezazu irakurle ezen `EJBObject` interfazearen metodoen inplementazioa EJB edukiontziaren esku gelditzen dela eta programatzaileak ez duela arduratu behar horretaz.

Bezeroaren deiaren zerbitzuak prozesatu ondoren, EJB objektu bat erabilita, EJB bean egokiari deitzen zaio bezeroaren deia egikaritzeko. Deia egikarituta, emaitza, bezeroari itzultzen dio edukiontziak aipatutako prozesua errepikatuta.

Gardentasuna areagotzeko J2EE teknologian, bezeroak ez du jakin behar non dauden kokatuta zehatz-mehatz EJB objektuak. Bestalde, eta orain arte ikusi dugunez, EJB objektuak erabilita jartzen da bezeroa harremanetan beanekin. Nola daki, orduan, non dauden EJB objektuak? Nola erreferentziatu EJB objektu bat, bere kudeaketa EJB edukiontziaren esku badago?



3.29. irudia. EJB edukiontziaren arkitektura orokorra.

Lehenengo eta behin, esan beharra dago bezeroak ezin duela EJB objektu bat instantziatu zuzenean, beste makina batean egon baitaiteke. Hori dela-eta, aipatutako kokapen-gardentasuna mantentzeko, EJB objektuak instantziatuko dituen faktoria-objektu bat behar dugu objektuak eraikitzeko, bilatzeko eta suntsitzeko.

EJB espezifikazioan, faktoria-objektu horiei, Home objektuak deitzen zaie. Home objektuak eraikitzeko beste interfaze bat sortu behar du programatzaileak `EJBHome` interfazea (**3.30. irudia**) hedatzen duena. `EJBHome` interfazeak edozein Home objektuk behar dituen metodoak definitzen ditu objektuak eraikitzeko eta suntsitzeko. Programatzaileak eraikitako interfaze hori, EJB edukiontziak inplementatzen du automatikoki eta programatzailearentzat erabat gardena da nola gauzatzen den prozesua. Aurreko interfazean bezala, interfaze honen metodoek,

besteak beste, `RemoteException` salbuespena altxa dezakete, zeren Home objektuak, EJB objektuak bezala, makina birtualen artean mugi baitaitezke.

J2EE teknologia erabilita diseinatzen ari garen sistema informatikoa erabat banatua denean, nahitaezkoa izaten da urruneko interfazeak erabiltzea, beanak bilatzeko urruneko zerbitzarian. Horrek, asko luzatzen du bilaketa-prozesua, zazpi eragiketa gauzatuko baitira:

1. Bezeroak stub lokal bati deitzen dio.
2. Stub objektuak informazioa sarera egokitzen du marshall eragiketa bat gauzatuta.
3. Informazioa sarean zehar garraiatzen da skeleton-a aurkitu arte.
4. Skeleton objektuak, informazioa, berriro egokitzen du, demarshall eragiketa bat osatuta. Horrela, informazioa ulergarria izango da EJB edukiontzi/zerbitzariarentzat.
5. Skeleton objektuak EJB objektuari deitzen dio bezeroaren deia prozesatzeko.
6. EJB objektuak, bezeroaren deiaren domeinuaren logika prozesatu baino lehen beharrezkoak diren iraunkortasun-, segurtasun- edo transakzio-zerbitzuak egikaritzen ditu.
7. Zerbitzuak egikaritu eta gero, Enterprise bean egokiaren instantziari dei egiten dio EJB objektuak, eskatu duen metodoa egikari dezan bezeroak.

Metodoa egikarituta, sortzen diren emaitzek atzekoz aurrera bete beharko dituzte aipatutako zazpi urratsak.

Bide luze hori nahitaezkoa izango da horrela programatzea sistema erabat banatua denean, baina batzuetan, abagunea izaten da sistemaren eraginkortasuna hobetzeko objektu lokalak inplementatuta. Objektu lokalek interfaze lokal bat inplementatuko dute urruneko interfaze baten ordez. Horrela, bezeroaren deiak bete beharko duen bidea askoz ere laburragoa izango da:

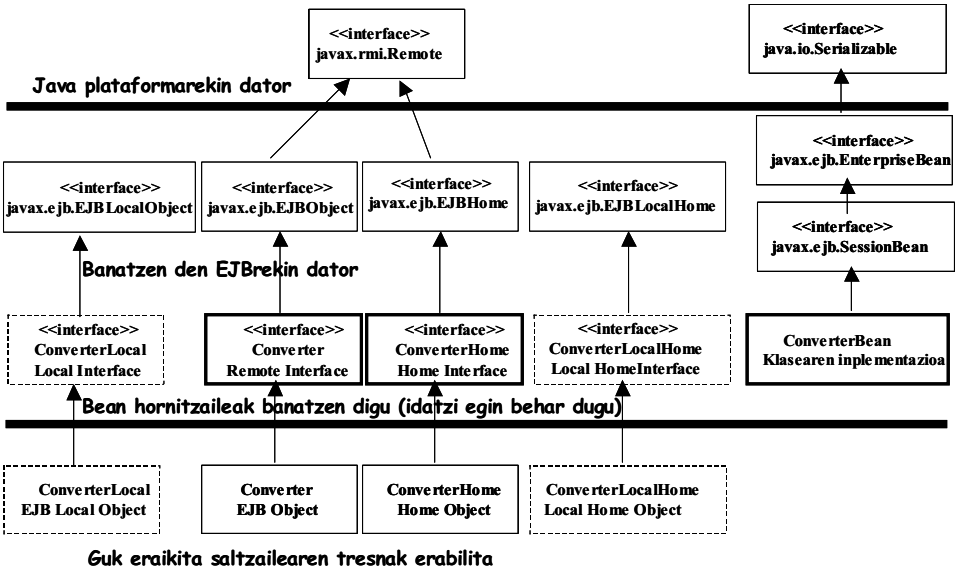
1. Bezeroak objektu lokal bati deitzen dio.
2. Objektu lokalak, bezeroaren deiaren domeinuaren logika prozesatu baino lehen, beharrezkoak diren iraunkortasun-, segurtasun- edo transakzio-zerbitzuak egikaritzen ditu.
3. Enterprise beanaren instantziak bezeroak eskatu duen metodoa egikaritzen du eta kontrola pasatzen dio objektu lokalari. Bukatzeko, objektu lokalak bezeroari itzultzen dio kontrola.

Begi-bistakoa da bide horrekin beanaren eraginkortasuna nabarmen hobetuko dela.

Programatzaileak bezeroaren deia bide horretatik kudeatu nahi badu, interfaze bat eraiki behar du, `EJBLocalObject` interfazea hedatuko duena. Interfaze hori, `EJBObject` interfazea hedatzen duen interfazearen bertsio lokala izango da, eta, beraz, urruneko interfazearen egitura bera izango du. Baina, objektuak lokalak direnez, interfaze berriaren metodoek ez dute `RemoteException` salbuespena altxatuko.

EJB objektuekin bezala, objektu lokalak instantziatuko dituen faktoria-objektu bat behar dugu objektu lokalak eraikitzeko, bilatzeko eta suntsitzeko. EJB espezifikazioan, faktoria-objektu horiei, `LocalHome` objektuak deitzen zaie. Programatzaileak, oraingoan, `LocalHome` objektuak eraikitzeko, beste interfaze bat sortu behar du, zeinak `EJBLocalHome` interfazea (**3.30. irudia**) hedatuko duen. Lehen bezala, objektu lokalekin ari garenez, interfaze berriaren metodoek ez dute zertan `RemoteException` salbuespena altxaturik.

Esanak esan, argi ikusten da EJB osagai bat diseinatzea ez dela Enterprise bean klasea programatzea soilik (**3.30. irudia**); izan ere, nahitaez, EJB edukiontziak informazio gehiago behar du beana atzitzeko. Batzuetan nahikoa izango da azpiegitura prestatzea bezeroaren urruneko deiak prozesatzeko, horretarako, nahikoa izango zaigu beste bi fitxategi (`EJBObject` eta `EJBHome` interfazeak) gehiago diseinatzea. Beste batzuetan, oraindik, beste bi fitxategi (`EJBLocalObject` eta `EJBLocalHome` interfazeak) diseinatu beharko ditugu objektu lokalak tratatzeko.



3.30. irudia. EJB osagaiaren azpiegitura teknologikoa.

Atal hau bukatzeko, adibide oso bat garatuko dugu, aipatutako fitxategi guztiak eraikiko dituena. Adibidearen bertsio partzial bat Sun erakundearen J2EE 1.3 tutoretza-eskuliburuan aurki dezake irakurleak. Bertsioak urruneko bezeroak hartzen ditu kontuan soilik, jaramonik egin gabe objektu lokalei. Aztertuko dugun adibideak diru-truke arrunt bat gauzatuko du. Bezeroari zuzenean deitzen bazaio agindu-lerro bat erabilita (**3.31. irudia**), bezeroak suposatzen du 100 zenbakia sartu dugula parametro gisa eta kopuru hori dela trukatu nahi duguna. Aitzitik, JSP orria erabiltzen badugu bezeroari deitzeko, trukatu nahi dugun balioa erabiltzaileak testu-eremu batean adierazten du eta emaitza JSP orrian (**3.32. irudia**) bertan agertzen da.

3.30. irudiak, eraiki behar ditugun objektu guztien egitura hierarkikoa islatzen du. Bertan, eskuineko zutabean diseinatu eta programatu behar dugun beana nola eratu adierazten da. EJB edukiontziak behar dituen fitxategiak bezeroaren urruneko deiak eta era lokalean gauzatuko diren deiak tratatzeko, irudiaren erdierdian agertzen dira.

Azter dezagun, bada, bestelako itzulingururik gabe, aipatutako fitxategi bakoitzaren edukia. Lehenengo eta behin, osagaiaren muinarekin hasiko gara: aplikazioak izango duen sesio-bean motako beanarekin, hain zuzen. Beanak, esan dugun bezala, `SessionBean` interfazea inplementatuko du inplementazio hutsa itsatsiz interfazearen metodo bakoitzari. Gainera, halaber, hor inplementatuko dira, bezeroak urrunetik eskatuko dituen metodoak. Gure kasuan, aplikazioak bi metodo (`dollarToYen()` eta `yenToEuro()`) eskainiko dizkio bezeroari diru-trukea gauzatzeko. Bai metodo batek eta bai besteak, `BigDecimal` motako balio bat itzuliko diote bezeroari. Hori guztia programatuta, honako iturburu-kode hau ateratzen zaigu:

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.math.*;

public class ConverterBean implements SessionBean {

      BigDecimal yenRate = new BigDecimal("121.6000");
      BigDecimal euroRate = new BigDecimal("0.0077");

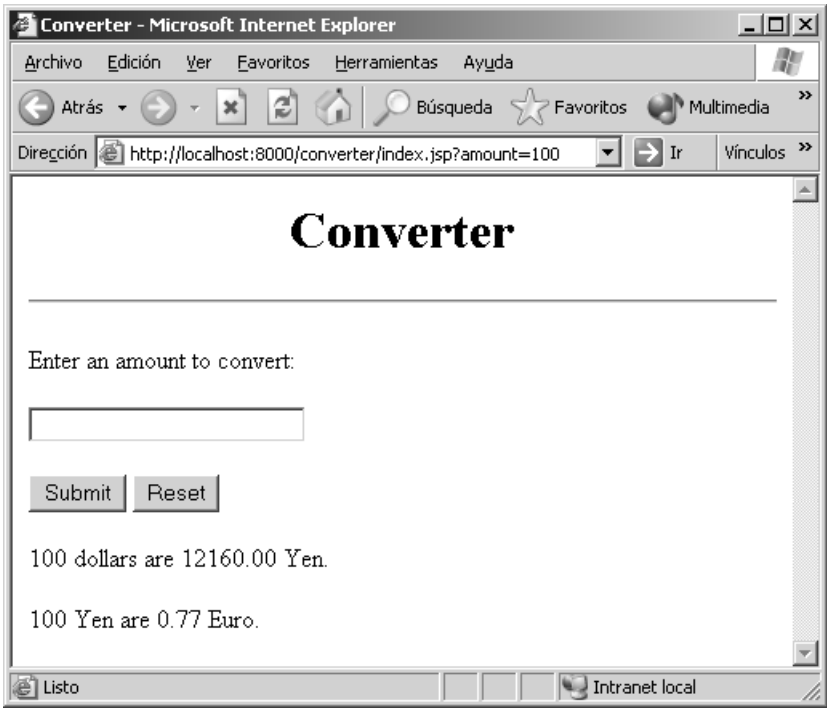
   public BigDecimal dollarToYen(BigDecimal dollars) {
      BigDecimal result = dollars.multiply(yenRate);
      return result.setScale(2,BigDecimal.ROUND_UP);}

   public BigDecimal yenToEuro(BigDecimal yen) {
      BigDecimal result = yen.multiply(euroRate);
      return result.setScale(2,BigDecimal.ROUND_UP);}
```

```
    public ConverterBean() {}
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
} // ConverterBean
```



3.31. irudia. Adibidearen irteera arrunta.



3.32. irudia. Adibidearen irteera JSP orria erabilita.

Programatu ditugun metodoak eskuragarri egon daitezen bezeroentzat, urruneko interfaze bat diseinatu behar dugu EJBObject interfazea hedatuz. Interfaze horrek, beanak ikusgai jarriko dituen metodoen signatura errepikatzen du:

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.math.*;

public interface Converter extends EJBObject {

   public BigDecimal dollarToYen(BigDecimal dollars) throws
RemoteException;
   public BigDecimal yenToEuro(BigDecimal yen) throws RemoteException;

     }
```

Konturatu ezen metodo bakoitzak `RemoteException` motako salbuespena altxa dezakeela. Bezeroaren deia, urrunekoa izan beharrean, lokala bada, `EJBLocalObject` interfazea hedatzen da, eraginkortasuna lortzeko. Gogoratu dei lokalak objektu lokalekin kudeatzen direla eta horretarako nahitaezkoa dela bigarren fitxategi hau:

```
import javax.ejb.EJBLocalObject;
import java.math.*;

public interface ConverterLocal extends EJBLocalObject {

   public BigDecimal dollarToYen(BigDecimal dollars) ;
   public BigDecimal yenToEuro(BigDecimal yen) ;   }
```

Deiak lokalak direnez, espero genuen bezala, interfaze horretan metodoek ez dute `RemoteException` salbuespena altxatzen, ez baita urruneko deirik gertatzen egoera horretan.

Berriki esan dugunez, EJB objektuak instantziatzeko faktoria-objektua behar dugu EJB objektuak eraikitzeko, bilatzeko eta suntsitzeko, hain zuzen, Home motakoak. Beraz, Home objektuak eraikitzeko interfaze bat eraikitzen da zeinak `EJBHome` interfazea hedatuko duen. Objektu lokalekin ari bagara, hedatu beharreko interfazea ez da `EJBHome` izango, `EJBLocalHome` baizik:

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface ConverterHome extends EJBHome {
    Converter create() throws RemoteException, CreateException;
      }
```

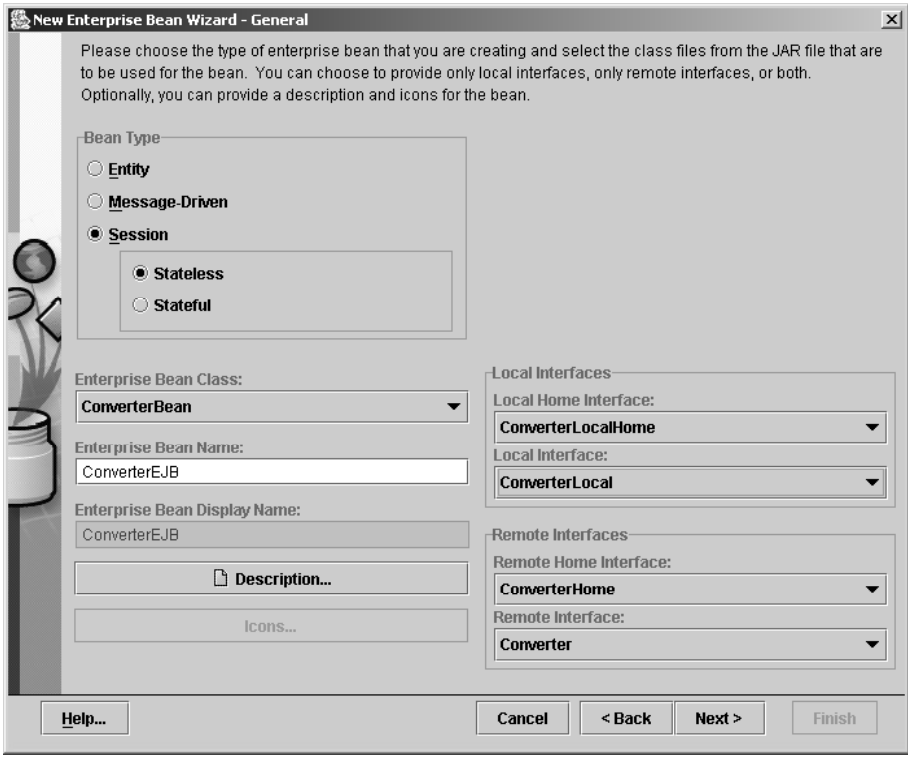
oOo

```
import javax.ejb.CreateException;
import javax.ejb.EJBLocalHome;

public interface ConverterLocalHome extends EJBLocalHome {

    ConverterLocal create() throws CreateException;
      }
```

Hemen ere gauza bera gertatuko da `RemoteException` salbuespenarekin, hau da, era lokalean ari garenez, ez dugu adierazi behar. Gogoratu ezen interfaze horiek EJB zerbitzariak inplementatzen dituela eta erabat gardenak direla programatzailearentzat. Gure adibidean, Home motako objektuek `create()` metodoa dute soilik eta haien eginkizuna `Converter` motako (`ConverterLocal` motakoa era lokalean ari garenean) EJB objektu bat eraikitzea da zerbitzarian.



3.33. irudia. Beaneren arkitektura orokorra.

EJB edukiontziaren azpiegitura antolatu ondoren, aplikazioaren zerbitzariaren aldea utzi eta bezeroarengana joango gara. Bertan, bezeroak, urruneko beanaren metodoei dei egin baino lehen, bizpahiru eragiketa gauzatu beharko ditu. Lehenengo eta behin, `Home` interfazea aurkitu beharko du, aipatutako `create()`

metodoa egikaritzeko. Metodoa egikaritzeko, `ConverterHome` motako objektu bat aurkitu eta instantziatu beharko du. Objektuaren bilaketa eta instantziazioa lau urratsetan gauzatzen da:

1. Lehenengo, hasierako izenen testuinguru bat eraikitzen da erabiliko diren izenen loturak gauzatzeko.

   ```
   Context initial = new InitialContext();
   ```

   Testuinguruak kudeatzeko, `InitialContext` motako objektu bat eraikitzen da, `Context` motako interfazea inplementatzeko. Horrenbestez, sarrera-puntu bat lortuko dugu izenen auzia konpontzeko. Gogoratu egikaritzen diren eragiketa guztiek beti izan behar dutela erreferentzia bat testuinguru batera. `Context` interfazea JNDI APIaren zati bat da, izenak kudeatzeko erabiltzen dena.

2. Bigarrenez, aplikazioaren izenen testuinguruaren ingurunea eskuratzen da. Horretarako, izenen testuinguruaren ingurunea kudeatzeko, aurretik eraikitako `Context` motako objektuaren `lookup()` metodoa egikaritzen dugu:

   ```
   Context myEnv = (Context)initial.lookup("java:comp/env");
   ```

   `lookup()` metodoaren parametroak, kudeatuko dugun izenen testuinguruaren ingurunea zehazten dio aplikazioaren bezeroari.

3. Ingurunea zehaztu ondoren, `ejb/SimpleConverter` izenarekin etiketatuta dagoen objektua bilatzen du programatzaileak, hau da, izenaren eta objektuaren arteko lotura bilatzen du. Horretarako, `lookup()` metodoa egikaritzen da berriz:

   ```
   Object objref = myEnv.lookup("ejb/SimpleConverter");
   ```

   Ondo badoaz gauzak, `ejb/SimpleConverter` izenak bat egingo du Enterprise bean baten erreferentzia batekin. Gure kasuan, `ConverterHome` motako objektu baten erreferentziarekin lotuko da izena. Konturatu ezen Enterprise beanaren izenak `java:com/env/ejb` azpitestuinguruan duela bere egoitza.

3. Bukatzeko, eskuratu dugun objektu-erreferentzia mugatu eta finkatu egin behar dugu `ConverterHome` motako objektu gisa:

   ```
   ConverterHome home =
     (ConverterHome)PortableRemoteObject.narrow(objref,
             ConverterHome.class);
   ```

Goiko lau urratsak egikarituta, programatzailea prest dago bean-instantzia eraikitzeko. Gogoratu, atzipena, ez dela zuzen-zuzenean egiten, EJB objektuak erabilita baizik. Hori da, beraz, bete beharreko hurrengo urratsa:

```
Converter currencyConverter = home.create();
```

Aplikazioaren bezeroak, `ConverterHome` motako objektuaren `create()` metodoa egikarituta, beana instantziatzen du, eta `Converter` motako `EJBObject` objektua uzten du eskuragarri eta ikusgai. Gogoratu, bezeroak, `EJBObject` motako objektuei egiten dizkiela deiak eta ez zuzenean beanaren instantziei. Gero zerbitzarian EJB edukiontziak delegatuko dio deia beanari.

Kateaketa eta lotura guztiak gauzatu ondoren, `Converter` motako objektuaren metodoei dei egiten die bezeroak, domeinuaren urruneko metodoak egikaritzeko. Azaldutako guztia klase batean idatziko bagenu honako iturburukode hau izango genuke:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import java.math.BigDecimal;

public class ConverterClient {

   public static void main(String[] args) {
       try {
                   Context initial = new InitialContext();

           Context myEnv =
(Context)initial.lookup("java:comp/env");
           Object objref = myEnv.lookup("ejb/SimpleConverter");

           ConverterHome home =
               (ConverterHome)PortableRemoteObject.narrow(objref,
                                              ConverterHome.class);

           Converter currencyConverter = home.create();

           BigDecimal param = new BigDecimal ("100.00");
           BigDecimal amount = currencyConverter.dollarToYen(param);
           System.out.println(amount);
           amount = currencyConverter.yenToEuro(param);
           System.out.println(amount);

           System.exit(0);

       } catch (Exception ex) {
           System.err.println("Caught an unexpected exception!");
           ex.printStackTrace();   }} }
```

Beanari dei egiteko, web maila bat txertatzen badugu bezeroaren eta edukiontziaren artean JSP orri bat erabilita, antzeko bideari jarraitzen zaio urruneko beana atzitzeko:

```
<%@ page import="Converter,ConverterHome,javax.ejb.*, java.math.*,
javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException" %>
<%!
   private Converter converter = null;

   public void jspInit() {
      try {
         InitialContext ic = new InitialContext();
         Object objRef = ic.lookup("java:comp/env/ejb/TheConverter");
         ConverterHome home =
(ConverterHome)PortableRemoteObject.narrow(objRef,
ConverterHome.class);
         converter = home.create();
      } catch (RemoteException ex) {
            System.out.println("Couldn't create converter bean."+
ex.getMessage());
      } catch (CreateException ex) {
            System.out.println("Couldn't create converter bean."+
ex.getMessage());
      } catch (NamingException ex) {
            System.out.println("Unable to lookup home: "+
"TheConverter "+ ex.getMessage());
      }
   }
public void jspDestroy() {
         converter = null;
   }
%>
<html>
<head>
    <title>Converter</title>
</head>

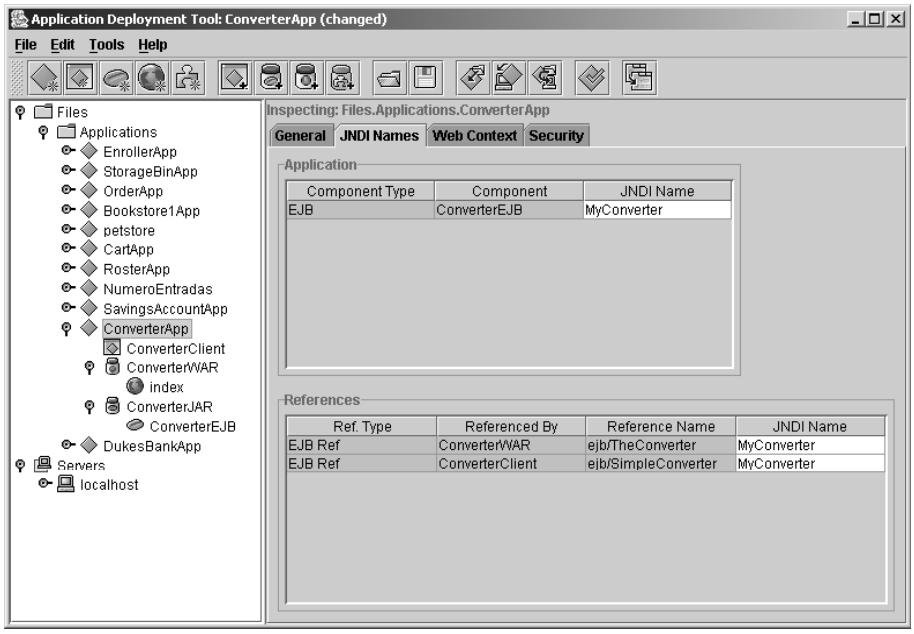
<body bgcolor="white">
<h1><b><center>Converter</center></b></h1>
<hr>
<p>Enter an amount to convert:</p>
<form method="get">
<input type="text" name="amount" size="25">
<br>
<p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

<%
    String amount = request.getParameter("amount");
    if ( amount != null && amount.length() > 0 ) {
       BigDecimal d = new BigDecimal (amount);
```

```
%>
   <p>
   <%= amount %> dollars are  <%= converter.dollarToYen(d) %>  Yen.
   <p>
   <%= amount %> Yen are <%= converter.yenToEuro(d) %>  Euro.
<%
    }
%>

</body>
</html>
```

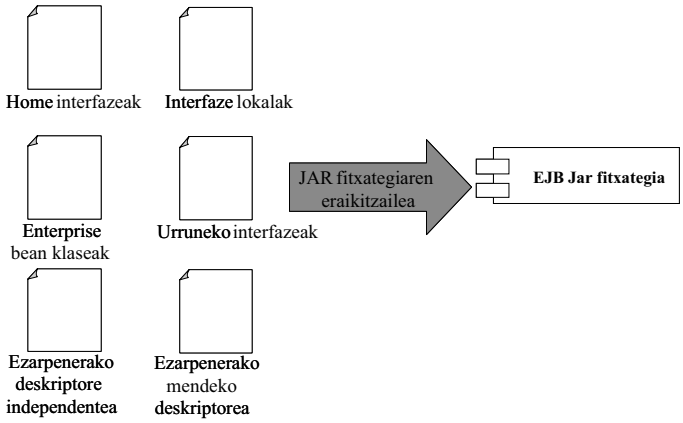
Aldaketa bakarra izenen testuinguruarekin (**3.34. irudia**) gertatzen da, hau da, bezero arruntak, `ConverterHome` motako objektu bat erreferentziatzen duenean `ejb/SimpleConverter` izena erabiltzen du. Bestalde, aplikazioaren bezero arrunta erabili beharrekoan web bezeroaren bidea aukeratzen badugu beana atzitzeko, `ConverterHome` motako objektu bat erreferentziatzen denean honako izen hau erabiltzen da: `ejb/TheConverter`. Nola liteke sinonimia hori? Nola eta non uztartzen dira izen desberdinak eta beana?



3.34. irudia. Izenen testuinguruak.

3.34. irudiak korapiloa askatzen lagunduko digu. Malgutasuna bultzatu nahian, posible da aplikazioa garatzen ari garenean, kodean egiten diren erreferentziak mapeatzea Enterprise beanaren JNDI izenarekin. Estrategia horrek zeharkako bideak eraikitzen ditu bezeroaren eta beanaren artean. Horrela,

erreferentzia-izen desberdina erabil dezake bezero mota bakoitzak osagai bera atzitzeko. Baina orduan, nola gauzatzen da lotura? `JNDI Name` eremuan izen bera jarrita.



3.35. irudia. JAR fitxategiaren osagaiak.

Orain arte garatu ditugun programetan nahikoa izaten zen iturburu-kodea konpilatzea programa egikaritzeko. EJB osagaiekin kontua zertxobait korapilatzen da. Hemen ez da nahikoa bakarrik `.class` fitxategiak izatea aplikazioa egikaritzeko, EJB `.jar` (**3.35. irudia**) motako fitxategi bat eraikitzen da zeinak aipatutako `.class` fitxategiak eta ezarpenerako deskriptoreak izango dituen. EJB `.jar` fitxategia eraiki ondoren, hedatu egin behar du edukiontziak, EJB osagaiak eskuragarri egon daitezen bezeroentzat. Ezarpenerako deskriptoreek, era erabat deklaratiboan eta XML erabilita, beanaren atributuak eta bezeroen ezaugarriak deskribatzen dituzte. Hona hemen, Sun enpresaren `deploytool` tresnak eraikitzen duen ezarpenerako deskriptorea, adibidearen Enterprise beanaren atributuak adierazteko:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
  <display-name>ConverterJAR</display-name>
  <enterprise-beans>
    <session>
      <display-name>ConverterEJB</display-name>
      <ejb-name>ConverterEJB</ejb-name>
      <home>ConverterHome</home>
      <remote>Converter</remote>
      <ejb-class>ConverterBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
```

```
      <security-identity>
        <description></description>
        <use-caller-identity></use-caller-identity>
      </security-identity>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <method-permission>
      <unchecked />
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Home</method-intf>
        <method-name>remove</method-name>
        <method-params>
          <method-param>java.lang.Object</method-param>
        </method-params>
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>yenToEuro</method-name>
        <method-params>
          <method-param>java.math.BigDecimal</method-param>
        </method-params>
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>getHandle</method-name>
        <method-params />
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Home</method-intf>
        <method-name>remove</method-name>
        <method-params>
          <method-param>javax.ejb.Handle</method-param>
        </method-params>
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Home</method-intf>
        <method-name>getHomeHandle</method-name>
        <method-params />
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>getPrimaryKey</method-name>
        <method-params />
```

```
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Home</method-intf>
        <method-name>getEJBMetaData</method-name>
        <method-params />
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>remove</method-name>
        <method-params />
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Home</method-intf>
        <method-name>create</method-name>
        <method-params />
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>isIdentical</method-name>
        <method-params>
          <method-param>javax.ejb.EJBObject</method-param>
        </method-params>
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>getEJBHome</method-name>
        <method-params />
      </method>
      <method>
        <ejb-name>ConverterEJB</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>dollarToYen</method-name>
        <method-params>
          <method-param>java.math.BigDecimal</method-param>
        </method-params>
      </method>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```

Deskriptorearen lehenengo lerroetan adierazten da EJB edukiontziak beharbeharrezkoa duen informazioa, besteak beste, `Home` eta urruneko interfazeak. Konturatu, halaber, adibideko Enterprise beana egoera gabeko (`Stateless`) sesiobeana dela.

Jarraian agertzen den informazio gehiena automatikoki eraikitzen du tresnak eta, bertan, metodoen informazioa islatzen da, hala nola metodoaren izena eta parametroen izaera. Konturatu, halaber, bertan daudela bezeroak erabiliko dituen `yenToEuro()` eta `dollarToYen()` metodoak.

Era berean, tresnak, beste bi deskriptore eraikitzen ditu bezero arruntaren eta web bezeroaren ezaugarriak jasotzeko:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application-client PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE
Application Client 1.3//EN' 'http://java.sun.com/dtd/application-
client_1_3.dtd'>

<application-client>
  <display-name>ConverterClient</display-name>
  <ejb-ref>
    <ejb-ref-name>ejb/SimpleConverter</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>ConverterHome</home>
    <remote>Converter</remote>
  </ejb-ref>
</application-client>
```

ooo

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN' 'http://java.sun.com/dtd/web-app_2_3.dtd'>

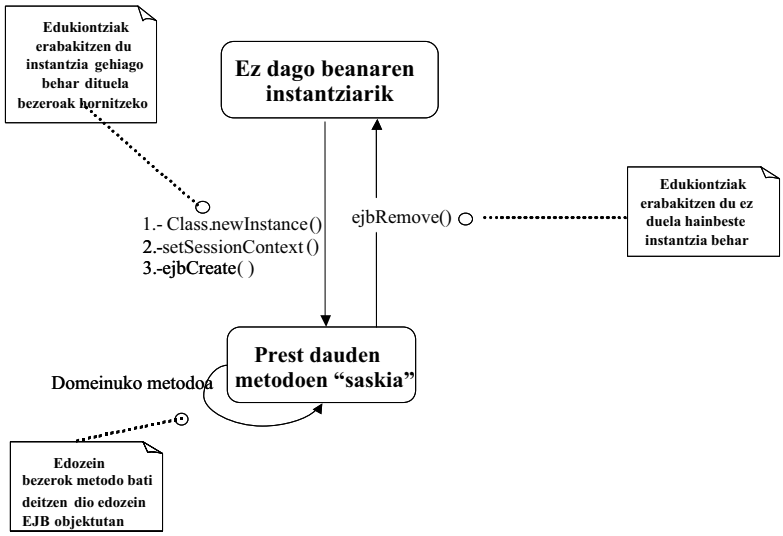
<web-app>
  <display-name>ConverterWAR</display-name>
  <servlet>
    <servletname>index</servletname>
    <display-name>index</display-name>
    <jsp-file>/index.jsp</jsp-file>
  </servlet>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <ejb-ref>
    <ejb-ref-name>ejb/TheConverter</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>ConverterHome</home>
    <remote>Converter</remote>
  </ejb-ref>
</web-app>
```

3.3.3. J2EE teknologiaren bean motak

Aplikazio informatikoak garatzen direnean, aztertu beharreko informazio-sistema bi zutabe nagusitan banatzen dute analistek eta diseinatzaileek: batean, aplikazioaren prozesu nagusiak arakatzen dira eta kontuan izaten da prozesu horiek harreman estuak dituztela domeinuaren prozesu fisikoekin. Bestean, informazio-sistemaren arlo egonkorra nola dagoen antolatuta ikertzen da, hemen kontuan hartzen dira informazio-fluxuan zehar higitzen diren agiriak eta agiri horietan azaltzen den informazioa.

EJB espezifikazioaren sortzaileek aipatutako bi zutabe horiei orpoz orpo jarraitu zieten EJB beanak motetan sailkatzean. Hala, aplikazioaren prozesuak kudeatzeko EJB sesio-beanak sortu zituzten eta informazio-sistemaren iraunkortasuna ziurtatzeko eta antolatzeko EJB entitate-beanak.

Bezeroa harremanetan jartzen denean urruneko zerbitzariarekin J2EE teknologia erabilita, ohikoa izaten da komunikazioa sesio-bean batekin hastea, eta domeinuaren informazio-sistemara atzitu behar badu, entitate-bean bat eraikitzea, irakurketak eta eguneraketak gauzatzeko.



3.36. irudia. Egoera gabeko sesio-beanaren bizi-zikloa.

Sesio-beana egoera gabekoa (`stateless`) denean, komunikazioa, dei batera eta bakarrera mugatuko da, hau da, komunikazioak, domeinuaren logikaren barnean, metodo-dei bat iraungo du. Deia egikaritu ondoren, hautaketa bat egin behar du EJB edukiontziak: sesio-beana suntsitu edo berreraiki berriro. Bigarren bidea aukeratzen badu, aurreko deiaren informazio guztia ezabatu egiten du beana berreraiki baino lehen. **3.36. irudia**n, egoera gabeko sesio-beanak izango duen

bizi-zikloa jasotzen da. Irudiko egoera-diagramak bi egoera nagusi ditu, eta hiru trantsizio, egoera batetik bestera igarotzeko. Hasieran, beanik gabe gaudenez, beana instantziatu nahi duela erabakitzen du edukiontziak. Noiz eta zenbat bean instantziatzen diren edukiontziak finkatzen du, beanak eraikitzeko aplikatzen duen politikaren arabera. Instantziak eraiki behar direla erabaki ondoren, hiru urratsetan gauzatzen da eragiketa:

1. Lehenengoan, `Class` objektuaren `newInstance()` metodoa egikaritzen du edukiontziak. Une horretan, beanak lehenetsia duen eraikitzaileari deitzen zaio, hasieraketa egokia gauzatzeko.
2. Bigarren urratsean, `setSessionContext()` metodoa egikaritzen du edukiontziak.
3. Bukatzeko, `ejbCreate()` metodoa egikaritzen du edukiontziak.

Beana eraiki ondoren, bezeroen deien zain gelditzen da edukiontzia, eta, deiak heldu ahala, saskian bilatzen du beana deiari esleitzeko. Konturatu bezeroak hainbat aldiz egikaritu dezakeela bean baten metodoa, baina horrek ez du esan nahi edukiontziak beti bean-instantzia bera erabiliko duela eskaera betetzeko.

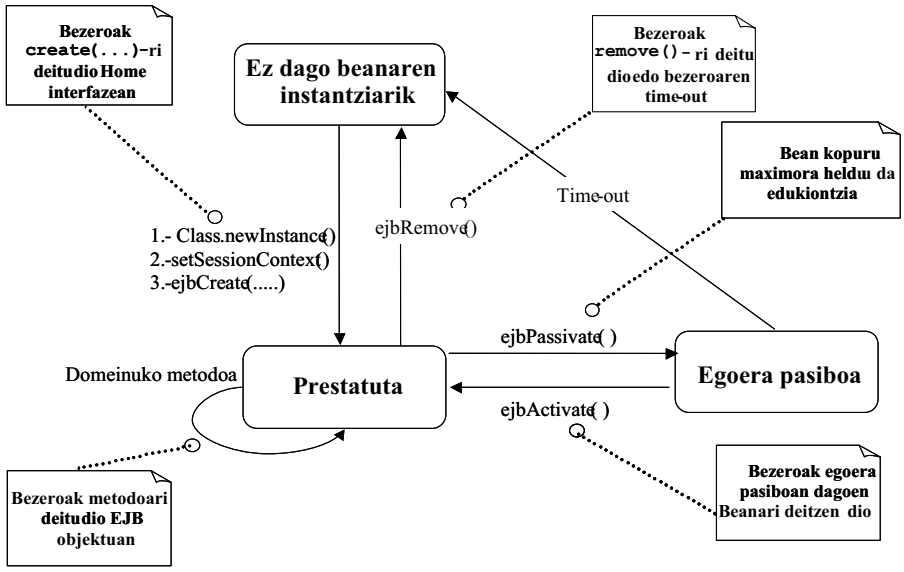
Edukiontziak erabakitzen duenean beana ezabatu egingo duela, beanaren `ejbRemove()` metodoa egikaritzen du lehenago beanak erabili dituen baliabideak askatzeko eta baliabide horiekin arazorik ez izateko.

Honaino egoera gabeko sesio-beanaren bizi-zikloari buruzkoa. Halaber, sesio-beanak egoera ere kontuan hartzen badu, hau da, bezeroaren komunikazioa ez denean murrizten dei bakarrera, beanak bizirik irauten du sesio osoan. Egoera hori da, adibidez, interesatzen zaiguna salerosketa elektroniko bat kudeatu nahi dugunean. Salerosketak irauten duen bitartean beana bizirik egotea komeni zaigu harik eta artikulu guztiak aukeratu eta eskaera bihurtu arte.

Sesio-bean mota horrek **3.37. irudia**n agertzen den bizi-zikloa edukiko du beanaren egoerak kudeatzeko. Ikusten den legez, antzekotasun handiak aurkitzen dira aurrekoarekin batera ipintzen badugu. Desberdintasun handiena `Egoera Pasiboa` egoera berriak dakar. Demagun bezeroaren dei bat jaso duela edukiontziak eta dei hori kudeatzeko, beste bean bat eraiki behar duela. Zoritxarrez, edukiontziak, beanen saskia lar beteta dauka eta zerbait egin behar du arazoa konpontzeko. Egoera gabeko sesio-beanez osatuta badago saskia, arazoa berehala konponduko du: suntsitu beanak lekua egiteko eta aurrera.

Baina, beste muturrera joanda, saskia egoeradun sesio-beanez beteta badago, zer egiten du? Bean mota horiek ezin ditu ezabatu sesioa bukatu arte gutxienez, eta bestalde, espazioa behar du bean berriarentzat. Edukiontziak, egoera berri horri aurre egiteko, biltegiratu egiten du egoeradun sesio-beana, beanaren `ejbPassivate()` metodoa egikarituta. Horrela, edukiontziak beanari informatu egiten dio biltegiratu

egingo duela eta erabiltzen ari den baliabideekin zerbait egin nahi badu orain dela une egokia.



3.37 irudia. Egoeradun sesio-beanaren bizi-zikloa.

Egoera pasiboan utzi berri dugun beanaren metodo bati deitzen dionean bezeroak, beanaren `ejbActivate()` metodoarekin berreskuratuko du edukiontziak saskira baliabideen egoera berreraikiz.

Irakurlea, honezkero zeure buruari galdezka egongo zara ea ahotan hartu dugun salerosketa elektronikoaren eskaeraren bilakaera ondo gauzatu ote zen. Egia esanda ez dakit nola gauzatu zen auzia azkenean, baina bistan da badakidala ezen aplikazioaren analistak entitate-beanak behar izan zituela aurrera ateratzeko proiektua. Ziur aski, bezeroak, eskaera osatu ondoren, edukiontziaren zerbitzariarekin mantentzen ari zen komunikazioa bukatutzat emango zuen eta aiduru geratuko zen artikuluen zain. Egoera horretan eta komunikazioa moztu baino lehen, zuhur jokatuz, euskarri egokian biltegiratuko zuen edukiontziak eskaera.

J2EE teknologiak, entitate-beanak erabilita gauzatzen du iraunkortasun hori. Hemen ere, lehen objektuekin gertatzen zen bezala, arazo txiki batekin aurkitzen gara datu-base erlazionalak erabiltzen baditugu informazioaren iraunkortasuna kudeatzeko, memorian dituen instantziak, mapeatu egin behar ditu edukiontziak datu-basearen taulen tuplekin.

Beanaren instantzia datu-basean biltegiratu nahi denean J2EE teknologia erabilita, bi estrategia aplika daitezke: beanak kudeatutako iraunkortasuna[33] aplikatu edo edukiontziak kudeatutako iraunkortasunaz[34] baliatu.

Beanak kudeatutako iraunkortasuna aplikatzea erabakitzen badu beanaren diseinatzaileak, biltegiratze-prozesu guztiaren ardura, beanaren programatzailearen esku geldituko da, beanak berak kudeatu beharko baititu eragiketa guztiak.

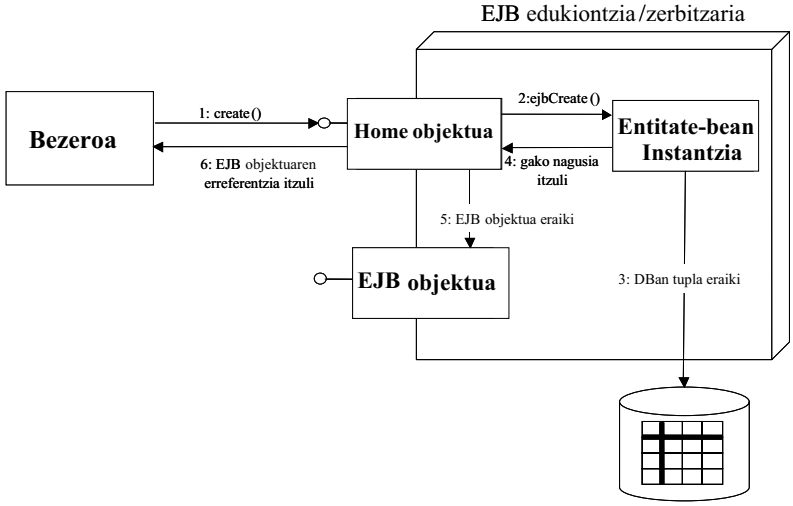
Bestalde, edukiontziak kudeatutako iraunkortasunaz baliatzen bagara, edukiontzia izango da ardura hartuko duena iraunkortasuna kudeatzeko eta programatzailearentzat erabat gardena izango da biltegiratuta dagoen informaziora atzitzea. Edukiontziak eraikiko du iraunkortasuna kudeatzeko behar duen logika guztia eta beana domeinuaren logikaz arduratuko da buru-belarri. Informazioa biltegiratu edo eskuratu behar duenean, beanak, edukiontziari eskatuko dio dagokion eragiketa gauzatzeko eta hura arduratuko da, gako nagusiak erabilita, informazioa eskuratzeaz EJB objektu erara.

Estrategia bat edo bestea erabiltzeak ez du eraginik izango entitate-bean bat eraiki nahi dugunean; batak eta besteak bide bera erabiliko dute beana eraikitzeko eta ezabatzeko. **3.38. irudia**n entitate-bean bat eraikitzeko egingo diren urratsak islatzen dira.

Aitzinean ikusi duguna sesio-beanekin, berriro errepikatuko da entitate-beanekin. Bezeroak entitate-beanaren instantziara atzitu nahi duenean, EJB objektuaz baliatuko da deien kudeaketa antolatzeko. Gogoratu ezen EJB objektuak Home objektuen bidez eraikitzen direla, eta, beanaren `ejbCreate()` metodoa erabiltzeko beraren `create()` metodoa definitu behar dugula `Home` interfazean. Horrela, lehenengo eta behin, Home objektuaren `create()` metodoari deituko dio bezeroak eta Home objektuak entitate-beanaren instantziari delegatuko dio deia `ejbCreate()` metodoa egikarituta.

Metodoa ondo egikaritu badu, entitate-beanaren instantziak eraikitako datuaren gako nagusia edukiontziari itzultzen dio. Gako nagusia erabilita, EJB objektu mota bat eraikitzen du edukiontziak eta beraren erreferentzia bezeroari itzultzen dio. Egin kontu, irakurle, ezen `create()` eta `ejbCreate()` metodoek balio desberdinak itzultzen dituztela eta kontuz programatu behar dela trantsizio hori. Adibide txiki batek esandakoa argitzen lagunduko digu. Demagun salerosketa elektronikoaren aplikazioan, bezeroaren eskaera kudeatzeko, `EskaeraBean` beana erabiltzen dugula. Beanaren instantzia bideratzeko, `Eskaera` urruneko interfazea, `EskaeraHome Home` interfazea eta `EskaeraGN` gako nagusiaren klasea diseinatu ditugu.

33. Bean-Managed Persistence (BMP).
34. Container-Managed Persistence (CMP).



3.38. irudia. Entitate-beana eraikitzeko urratsak.

Gauzak horrela, `EskaeraBean` klasean programatuko dugun `ejbCreate()` metodoak honako signatura hau izango luke:

```
public EskaeraGN ejbCreate(String eskaeraGako, String informazioa)
    throws....
```

Bestalde, `Home` interfazean pareko `create()` metodoa programatu behar dugu honako signatura hau erabilita:

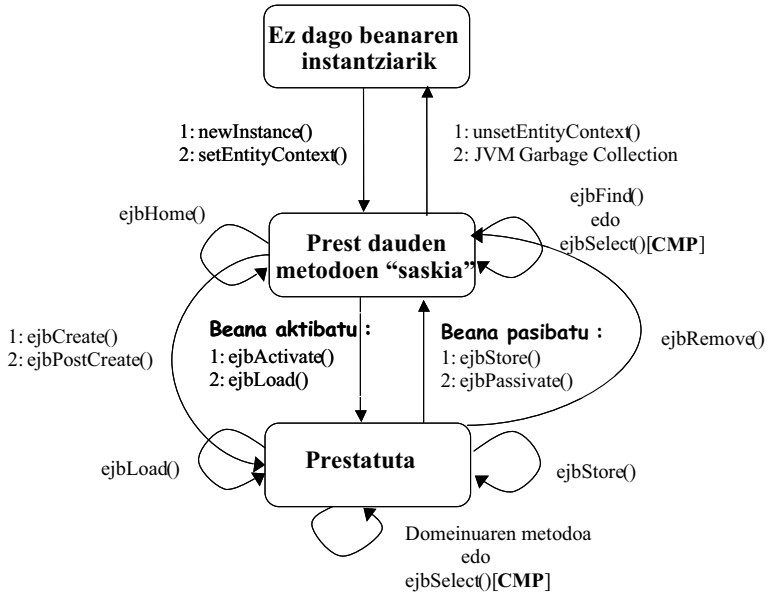
```
public Eskaera create(String eskaeraGako, String informazioa)
    throws....
```

Konturatu irakurle ezen bezeroak ikusten duena EJB objektua dela, hau da, entitate-beanaren proxy-a. Bestalde, EJB objektu hori, beanaren instantziak itzultzen dion gako nagusia erabilita eraikiko du edukiontziak. Zehatz-mehatz, gako nagusia bereganatzen du `Home` objektuak eta faktoria-lanak gauzatuz EJB objektua eraikitzen du.

Entitate-beanaren bizi-zikloak (**3.39. irudia**) aurretik ikusitako beanen bizi-zikloaren ezaugarriak errepikatzen ditu bizi-zikloaren hasierako urratsetan. Hala nola instantzia bat eraikitzeko, hemen ere, entitate-beanaren klasearen `newInstance()`[35] metodoari deitzen dio edukiontziak. Bigarren urratsean, entitate-beanaren ingurunea antolatuko du edukiontziak, testuingurua finkatuta `setEntityContext(EntityContext testuingurua)` metodoarekin. Bi eragiketak gauzatuta, prest dago beana aginduak jasotzeko saskian. Saskian dagoela, `ejbFind()` motako deiak jaso ditzake beanak datu-baseko informazioa

35. Beanak lehenetsia duen eraikitzaileari deitzen dio deiak benetan eta, horrela, beanaren instantzia bat eraikitzen da memorian.

eskuratzeko. Beste batzuetan, eragiketa orokorragoak gauzatzeko eskatuko zaio `ejbHome()` motako deiak egikarituta.



3.39. irudia. Entitate-beanaren bizi-zikloa.

Bezeroak datu berriak txertatu nahi dituenean datu-basean, lehen azaldu dugun bezala, Home objektuari `create()` metodoa egikaritzeko eskatuko dio eta, horrek, beanaren instantziaren `ejbCreate()` metodoari pasatuko dio ardura hori.

EJB objektua eraiki ondoren, beana prest dago aplikazioaren domeinuaren logikaren metodoak egikaritzeko, bezeroak behar-beharrezkoa duen EJB objektura itsatsita geldituko baita beana.

Batzuetan edukiontziak sinkronizatu beharko du entitate-beanaren instantzia baldin beste bean-instantzia batzuek datu-basearen informazioa eguneratu egiten badute. Horrela, eguneratutako informazioarekin lan egingo du bezeroak beti eta ez bertsio zaharkituekin. Hori egiteko eta buruhausteak saihesteko, `ejbLoad()` eta `ejbStore()` metodoak erabiliko ditu edukiontziak.

Prestatuta egotetik saskira itzuli nahi badugu beana, bi bide daude aukeran: lehenengoan, Home objektuaren `remove()` metodoari deitzen dio bezeroak eta bean-instantziaren `ejbRemove()` metodoari delegatzen dio edukiontziak deia. Metodoa egikaritu ondoren, eten egiten da EJB objektuaren lotura entitate-beanaren instantziarekin.

Bigarren aukeran, edukiontziak berak gauzatzen du bere kasa eragiketa. Edukiontziak erabakitzen badu, adibidez, entitate-beana erabili nahi duela beste

bezero bati zerbitzua emateko, jakin badaki, beana ezin duela ezabatu, alde bateko ituna izenpetu izan balu bezala. Lehenengo eta behin, informazioaren azken bertsioarekin lanean ari dela ziurtatu behar du eta, horretarako, lehengo `ejbStore()` metodoa egikarituko du. Hori eginda, beana pasibatu egingo du haren egoera babestuz, beanaren `ejbPassivate()` metodoa egikarituta. Beana aktibatzeko alderantzizko ibilbidea egin beharko du, hau da, beana aktibatu `ejbActivate()` metodoarekin, eta datu-basearen informazio eguneratua eskuratu, `ejbLoad()` metodoarekin. Horrela, prest dugu berriro beana, aplikazioaren domeinuaren logikaren metodoak egikaritzeko.

Beanak atzitzeko orain arte erabili dugun bidea RMI-IIOP protokoloan oinarritzen zen eta, beraz, diseinatzen genituen aplikazioak programazio sinkronoan zeuden errotuta, hau da, bezeroak dei bat egiten zuenean, dei hori une horretan prozesatzen zen. Batzuetan, ordea, tartean arazoak aurkitzen bazituen, zerbitzaria erorita zegoelako edota sareak arazoak zituelako, deia galdu egiten zen eta, gainera, bezeroak, ezin zuen ezer egin bitartean. Deietan horrelako eragozpenak saihesteko, metodoetan oinarritu beharrean bezeroaren komunikazioa zerbitzariarekiko mezuetan oinarritzen zen estrategia berri bat garatu zen. Estrategia horrek, espero genuen bezala, programazio asinkronoa hartu zuen programazio-paradigma gisa.

Egoera horretan, bezeroak ez du itxaron behar eskaera gauzatzen den ala ez. Adibide argiena Amazon.com web gune ezagunaren salerosketa elektronikoan azaltzen da. Bezeroak artikuluen eskaera osatuta ados botoia zapaltzen duenean ez du zain egon behar bere kreditu-txartela egiaztatu arte, prozesua, erabat asinkronoa bihurtzen baita une horretan.

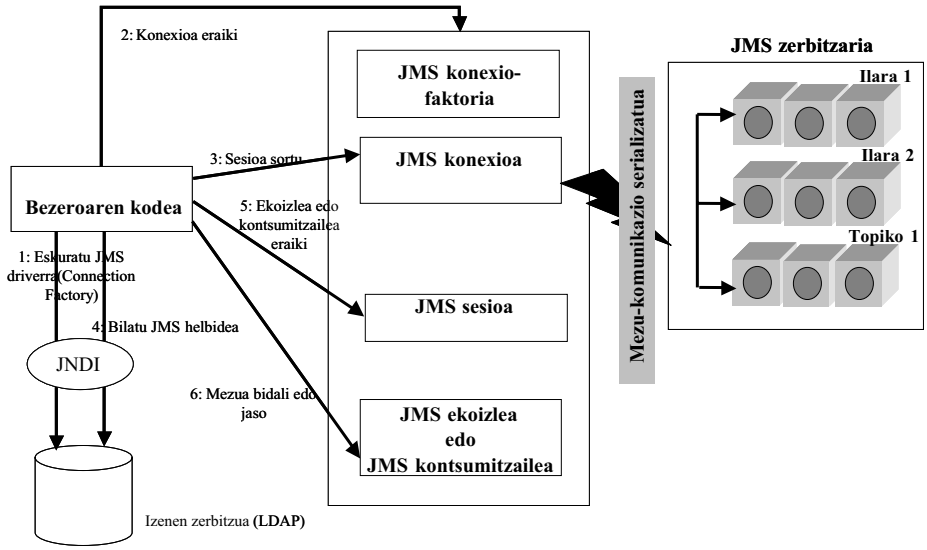
Gauza bera gertatzen da bezeroak dei bat bidaltzen dionean zerbitzariari eta zerbitzarian arazoak badaude dei hori prozesatzeko. Batzuetan, bezeroari ez zaio interesatuko deiaren erantzuna berehala jasotzea baina interesatuko zaio dei hori ez galtzea eta zerbitzariak ahal duenean prozesatzea. Horrelako egoerei aurre egiteko, mezuetara zuzendutako erdiko maila berri bat txertatu beharko da aplikazioen arkitektura orokorrean. Egun, hainbat tresna daude mezuetara zuzendutako erdiko maila hori inplementatzen dutenak, besteak beste, IBMren Mqseries, Tibcoren Rendezvous, BEAren Tuxedo/Q eta Microsoften MSMQ. Denek azpiegitura teknologiko egokia eskaintzen diote programatzaileari, mezuetan oinarritutako komunikazio asinkronoak eratzeko.

J2EE teknologiak ere, aipatutako egoerak tratatzeko, JMS (Java Message Service) APIa jartzen du programatzailearen esku, mezuak eraikitzeko, bidaltzeko eta jasotzeko. JMS APIaren arkitekturak JDBC eta JNDI APIen ikuspegi funtzional bera du, hau da, mezuen trataera erabat gardena eta independentea izatea programatzailearentzat aplikazioa programatzean.

JMSn bi estrategia erabil daitezke mezuak transmititzeko:

1. Puntutik punturako transmisioa. Transmisio mota honetan mezuak kontsumitzaile bakarra izango du. Mezu-ekoizleek JMS hornitzaileari bidaltzen dizkioten mezuak ilara batean gordetzen dira baina mezu bakoitza beti bidaltzen zaio kontsumitzaile bati.

2. Argitaratze/harpidetza mota. Bigarren transmisio mota honetan zabaldu egiten dira aukerak mezuarentzat. Mezuek ekoizle asko izan ditzakete eta aldi berean, baita kontsumitzaile ugari ere. Kontsumitzailea topikoen harpidedun bihurtzen da eta ekoizleek mezuak topiko desberdinetara bidaltzen dituzte. Mezua topiko batera heltzen denean, harpidedunen zerrenda kontsultatzen da eta mezua harpidedun bakoitzari bidaltzen zaio. Honezkero irakurle azkarrak antzemango zuen ezen azaldutakoa Observer patroi ezagunaren ikuspegi asinkronoa besterik ez dela.



3.40. irudia. JMS APIaren programazio-eredua.

JMS APIaren programazio-eredua, **3.40. irudia**n jasotzen da eta honako urrats hauek ditu:

1. JMS driverra JNDI erabilita eskuratu. Urrats honetan, JMS inplementazioaren driver partikularra atzituko du aplikazioaren bezeroak. Horretarako, eta JDBC APIarekin egiten genuen bezala, JNDI teknologia erabilita driverra eskuratzen dugu. Driverra `ConnectionFactory` motako objektua izango da. Mezuak transmititzeko erabiliko den estrategiaren arabera, faktoria-objektu hori `TopicConnectionFactory` edo `QueueConnectionFactory` motakoa izango da.

```
// Hasieratu JNDI
    Context testuingurua =
  new InitialContext(System.getProperties());
```

- Adibidea (`TopicConnectionFactory`):

```
// 1: Bilatu ConnectionFactory  JNDI erabilita
TopicConnectionFactory faktoria =
   (TopicConnectionFactory)

    testuingurua.lookup("javax.jms.TopicConnectionFactory");
```

- Adibidea (`QueueConnectionFactory`):

```
QueueConnectionFactory faktoria =
  (QueueConnectionFactory)

   testuingurua.lookup("javax.jms.QueueConnectionFactory")
```

2. Lehenengo urratsean eraiki dugun faktoria erabili JMS konexioa sortzeko. Sareko komunikazioaren behe-mailako trikimailuak kudeatuko ditu konexio horrek. JDBC konexioa ezaguna egiten bazaizu irakurle, konexio hau ere horrelakoa da.

   - Adibidea:

```
// 2: Erabili faktoria JMS konexioa(Topic) eraikitzeko,
      // berdin     Queue-rekin
        TopicConnection konexioa =
        faktoria.createTopicConnection();
```

3. JMS sesio bat eraiki faktoria gisa erabiltzeko, mezuen ekoizleentzat eta kontsumitzaileentzat. Sesioa eraikitzeko, aurreko urratsean sortu berri dugun konexioaz baliatuko gara.

   - Adibidea:

```
// 3: Konexioa erabili sesioa eraikitzeko,
// berdin     Queue-rekin
  TopicSession sesioa = konexioa.createTopicSession(
   false, Session.AUTO_ACKNOWLEDGE);
```

4. JNDI erabilita JMS helbidea bilatu. Bilatzen ari garen helbidea, mezuak bidaltzeko eta jasotzeko erabiliko dugun kanala izango da.

   - Adibidea:

```
// 4: Bilatu JMS helbidea (Topic) JNDI erabilita
// berdin     Queue-rekin
Topic topikoa = (Topic) testuingurua.lookup("gureTopikoa");
```

5. Mezu-ekoizlea edo -kontsumitzailea eraiki beharren arabera, hau da, mezuak bidaltzen badira ekoizlea eraiki beharko dugu, eta mezuak jaso egiten badira, kontsumitzailea.

• Adibidea:

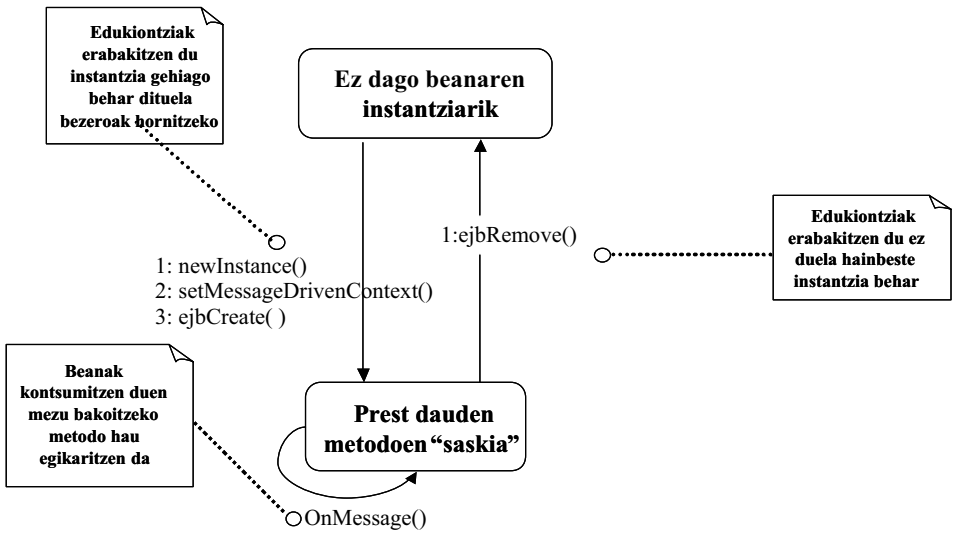
```
// 5: Eraiki mezu ekoizlea
TopicPublisher ekoizlea =
sesioa.createPublisher(topikoa);
```

6. Bukatzeko, mezua ekoitzi edo kontsumitu egiten da.

• Adibidea:

```
// 6: Mezua eraiki eta ekoitzi
TextMessage mezua = sesioa.createTextMessage();
mezua.setText("Mezua ekoizten.");
 ekoizlea.publish(mezua);
```

Aurkeztu dugun programazio-ereduak argi uzten du mezuak bideratutako beanak portaera xumea izango duela bere bizi-zikloa (**3.41. irudia**) aztertzen badugu. Bean mota hau, azken batean, EJB osagai bat izango da mezuen zain dagoena. Aurreko beste beanekin gertatzen zen bezala, hemen ere, mezua ez da zuzenean beanera joango, lehendabizi, zerbitzariak jasoko du, eta gero, bean-instantzia egokia aukeratuko du mezua egikaritzeko.



3.41. irudia. Mezuak bideratutako beanaren bizi-zikloa.

Beraz eta bean mota horren izaera kontuan izanda, beanak ez du baliorik itzuliko komunikazio asinkronoa gauzatzen baitago eta, orobat, ez du urruneko salbuespenik altxatuko. Bestalde, ez ditu urruneko interfazeak behar eta ez du inongo egoerarik gordetzen. Alde horretatik, egoera gabeko sesio-beana bezala funtzionatzen du. Gainera, aplikazioaren domeinuaren logika tratatzeko, `onMessage()` metodo bakarra izango du bean mota horrek.

Atal honi amaiera emateko, adibide bat garatuko dugu JMS APIa erabiliz. Irakurleak, nahi izanez gero, Sun erakundearen J2EE 1.4 tutoretza-eskuliburuan aurki dezake adibidea. Adibidean, aplikazioaren ekoizle-lanak gauzatzen ari den programari deitzean, hiru parametro pasatu behar dizkio bezeroak: bilatu behar duen helbidea, eratu behar duen konexio mota (`Queue` edo `Topic`) eta sortu behar duen mezu kopurua. Gauzak horrela, honako agindu hau egikarituko dugu ekoizlearen aldean:

```
> appclient –client SimpleProducer.jar jms/Queue queue 3
```

Hori egin ondoren, ekoizleak hiru eragiketa nagusi gauzatu beharko ditu mezuak eraikitzeko eta toki egokira bideratzeko:

1. Aurretik sortua ez badago, eraiki `InitialContext` motako objektua JNDI APIa erabilita.

```
try {
    Context jndiContext = new InitialContext();
} catch (NamingException e) {
   System.out.println("Could not create JNDI API " +
       "context: " + e.toString());
System.exit(1); }
```

2. Bilatu objektua, konexio-faktoria (`ConnectionFactory`) lanak gauzatuko dituena eta, konexio motaren arabera, eskuratu mezuak bidaltzeko erabiliko den helbidea (`Queue` edo `Topic`).

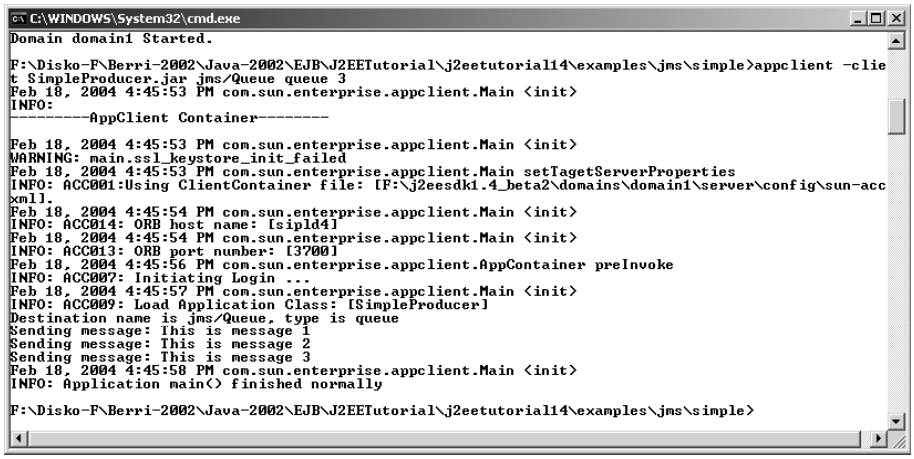
```
ConnectionFactory  connectionFactory = (ConnectionFactory)
      jndiContext.lookup("jms/QueueConnectionFactory");
   Destination dest;
   if (destType.equals("queue")) {
      dest = (Queue) jndiContext.lookup(destName);
   } else if (destType.equals("topic")) {
      dest = (Topic) jndiContext.lookup(destName);
      throw new Exception("Invalid destination type" +
          "; must be queue or topic");}
```

3. Hirugarren urrats nagusian, sei azpieragiketa kateatu gauzatzen dira jarraian: lehenengo eta behin, konexioa eraiki. Bigarrenez, sesioa sortu, eraiki berri den konexiotik eta, hirugarrenez, ekoizlea (`MessageProducer`) eta `TextMessage` motako objektuak eraiki.

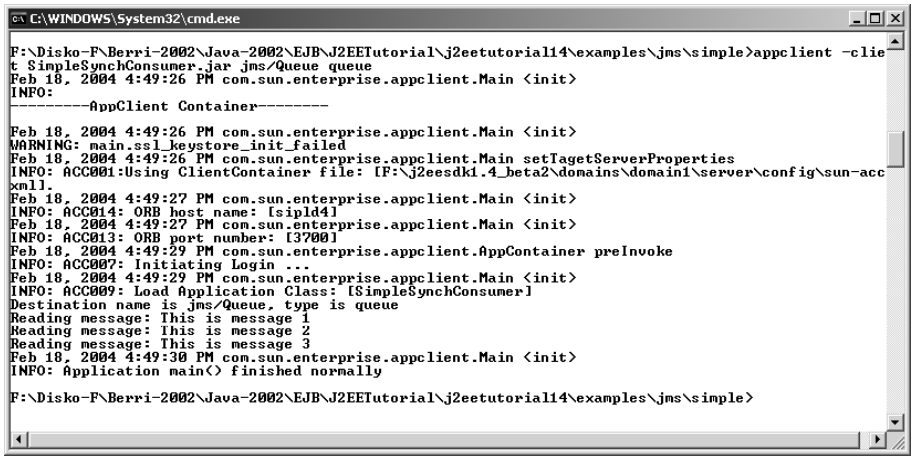
   Hiru urrats horiekin ekoizlea prest dago mezuak bidaltzeko. Horretarako, `String` motako mezu bat prestatzen dugu aldiko, eta, ekoizlearen `send()` metodoa egikarituta, bidali egiten dugu, laugarren urratsa osatuz.

   Mezu guztiak bidali ondoren, eta konexioa itxi baino lehen, mezu berezi bat bidaltzen dugu, mezuak bukatu direla adierazteko. Hori eginda, bere lana bukatutzat ematen du ekoizleak eta konexioa ixten du.

```
Connection connection = connectionFactory.createConnection();
Session session = connection.createSession(false,
       Session.AUTO_ACKNOWLEDGE);
MessageProducer producer = session.createProducer(dest);
TextMessage message = session.createTextMessage();
message.setText("This is message ");
message.getText());
producer.send(message);
producer.send(session.createMessage());
connection.close();
```



3.42. irudia. Ekoizlearen aldeko portaera irteera estandarrean.



3.43. irudia. Kontsumitzailearen aldeko portaera irteera estandarrean.

Hona hemen nola geldituko litzatekeen ekoizlearen programa nagusia aipatutako hiru urratsak batera jarri eta parametroen sarrera kudeatuko bagenu:

```
import javax.jms.*;
import javax.naming.*;

public class SimpleProducer {

    public static void main(String[] args) {
        String              destName = null;
        String              destType = null;
        Context             jndiContext = null;
        ConnectionFactory   connectionFactory = null;
        Connection          connection = null;
        Session             session = null;
        Destination         dest = null;
        MessageProducer     producer = null;
        TextMessage         message = null;
        final int           NUM_MSGS;

        if ( (args.length < 2) || (args.length > 3) ) {
       System.out.println("Program takes two or three arguments: " +
                "<dest_name> <queue|topic> " +
                "[<number-of-messages>");
            System.exit(1);  }
        destName = new String(args[0]);
        destType = new String(args[1]);
        System.out.println("Destination name is " + destName +
            ", type is " + destType);
        if (args.length == 3){
            NUM_MSGS = (new Integer(args[2])).intValue();
        } else {
            NUM_MSGS = 1;   }
        try {
            jndiContext = new InitialContext();
        } catch (NamingException e) {
            System.out.println("Could not create JNDI API " +
                "context: " + e.toString());
            System.exit(1);
        }
        try {
            connectionFactory = (ConnectionFactory)
                jndiContext.lookup("jms/QueueConnectionFactory");
            if (destType.equals("queue")) {
                dest = (Queue) jndiContext.lookup(destName);
            } else if (destType.equals("topic")) {
                dest = (Topic) jndiContext.lookup(destName);
            } else {
                throw new Exception("Invalid destination type" +
                    "; must be queue or topic");  }
```

```
} catch (Exception e) {
    System.out.println("JNDI API lookup failed: " +
        e.toString());
    e.printStackTrace();
    System.exit(1);   }
try {
    connection = connectionFactory.createConnection();
    session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);
    producer = session.createProducer(dest);
    message = session.createTextMessage();
    for (int i = 0; i < NUM_MSGS; i++) {
        message.setText("This is message " + (i + 1));
        System.out.println("Sending message: " +
            message.getText());
        producer.send(message);   }
    producer.send(session.createMessage());
} catch (JMSException e) {
    System.out.println("Exception occurred: " +
        e.toString());
} finally {
    if (connection != null) {
        try {
            connection.close();
        } catch (JMSException e) {}   }}}}
```

Ekoizlearen lana antolatuta, aplikazio informatikoaren erdi-lana osatu dugu: sistema mezuz elikatzea eta prest gelditzea norbaitek kontsumi ditzan Hori da, bada, bigarren zati honetan egingo duguna. Zorionez, lehenengo bi urratsak, orpoz orpo errepikatzen direnez, txanda-pasa eginda, lehengo hirugarren urrats nagusian kokatuko gara:

```
    try {
    connection = connectionFactory.createConnection();
    session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);
    consumer = session.createConsumer(dest);
    connection.start();
    while (true) {
        Message m = consumer.receive(1);
        if (m != null) {
            if (m instanceof TextMessage) {
                message = (TextMessage) m;
                System.out.println("Reading message: " +
                    message.getText());
            } else {
                break;  }}}
} catch (JMSException e) {
    System.out.println("Exception occurred: " +
        e.toString());
```

```
        } finally {
            if (connection != null) {
                try {
                    connection.close();
                } catch (JMSException e) {}  }}}}
```

Iturburu-kodean ikusten den legez, hemen ere, kontsumitzaileari oso gutxi dagokio: kontsumitzailea (`MessageConsumer`) eraikitzea sesioaren `createConsumer()` metodoa egikarituta; konexioa hasieratzea, `start()` metodoa egikarituta mezuak jasotzeko, eta mezuak eskuratzea, `receive()`[36] eta `getText()` metodoak erabilita.

Orain arte aurkeztu duguna balekoa da mezuak era sinkronoan kontsumitzen direnean. Gauzak zertxobait aldatzen dira mezuak kontsumitzeko era asinkronoa izaten denean. Egoera horretan, eskuarki, kontsumitzaile asinkronoa amaigabe gelditzen da egikaritze-egoeran. Hori dela-eta, kontsumitzaile asinkronoa kudeatzen duen programak irteera bat eduki beharko du prestatuta erabiltzaileak edozein unetan programaren egikaritzeari utz diezaion. Bestela, kontsumitzaile asinkronoak gauzatuko dituen urratsak, kontsumitzaile sinkronoarenak izango dira, alegia, lehenengo eta behin, konexioa eraiki; bigarrenez, sesioa sortu eraiki berria den konexiotik, eta, hirugarrenez, kontsumitzailea (`MessageConsumer`) eraiki sesioaren `createConsumer()` metodoa egikarituta:

```
Connection connection = connectionFactory.createConnection();
      Session session = connection.createSession(false,
                Session.AUTO_ACKNOWLEDGE);
      MessageConsumer consumer = session.createConsumer(dest);
```

Kontsumitzailea sortu ondoren, konexio asinkronoaren diseinuari berrikuntza bat itsatsi behar zaio, konexio sinkronoan behar ez genuena: mezu-entzulea (`MessageListener`). Mezu-entzule[37] hori, kontsumitzaile asinkronoan erregistratzen dugu, kontsumitzailearen `setMessageListener()` metodoa egikarituta:

```
TextListener listener = new TextListener();
consumer.setMessageListener(listener);
```

Baina, zer da `TextListener` motako objektu hori? Nondik dator eta nola dago eratuta? Ondoko iturburu-kodeak lagunduko digu galdera horiek erantzuten:

---

36. `receive()` metodoari, argumentuan, 0 edo hutsa jarrita deitzen bazaio, metodoa blokeatuta gelditzen da mezu bat heldu arte. Bezero bat dugunean bakarrik, ez dago arazorik baina sistemaren baliabideak ez baditugu alferrik kontsumitu nahi, denborak kontrolatutako `receive()` programatu behar dugu. Gure adibidean, `receive()` metodoa, milisegundo bateko denbora-mugarekin programatuko dugu.

37. Gogoratu antzeko estrategia aplikatzen genuela AWT edo SWING osagai bat diseinatzen genuenean. Han, aplikazio informatikoa, gelditzen zen gertaeren zain, eta hemen, mezuen zain egongo da.

```
import javax.jms.*;

      public class TextListener implements MessageListener {

    public void onMessage(Message message) {
        TextMessage msg = null;

        try {
            if (message instanceof TextMessage) {
                msg = (TextMessage) message;
                System.out.println("Reading message: " +
                    msg.getText());
            } else {
                System.out.println("Message is not a " +
                    "TextMessage");
            }
        } catch (JMSException e) {
            System.out.println("JMSException in onMessage(): " +
                e.toString());
        } catch (Throwable t) {
            System.out.println("Exception in onMessage():" +
                t.getMessage());}}}
```

`TextListener` klaseak `MessageListener` interfazea inplementatzen du, `onMessage()` metodo ezaguna inplementatuta. Gogoratu ezen `onMessage()` metodoari automatikoki deitzen zaiola, EJB edukiontzian, mezu bat heltzen zaionean kontsumitzaileari. Gure kasuan, `TextMessage` motako objektura bihurtzen du metodoak mezu heldu berria datu-bihurketa aplikatuta. Eskuratutako mezua ez bada testu-mezu bat, detektatu egiten du metodoak eta ohar bat ateratzen du irteera estandarretik.

Aurkeztutako iturburu zati guztiak elkartuko bagenitu konexio asinkronoa kudeatzen duen programan, honako iturburu-kode hau sortuko litzaiguke kontsumitzaile asinkronoa eratzeko:

```
try {
connection = connectionFactory.createConnection();
session = connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);
consumer = session.createConsumer(dest);
listener = new TextListener();
consumer.setMessageListener(listener);
connection.start();
System.out.println("To end program, type Q or q, " +
    "then <return>");
inputStreamReader = new InputStreamReader(System.in);
while (!((answer == 'q') || (answer == 'Q'))) {
    try {
        answer = (char) inputStreamReader.read();
    } catch (IOException e) {
```

```
                System.out.println("I/O exception: "
                    + e.toString());   }}
        } catch (JMSException e) {
            System.out.println("Exception occurred: " +
                e.toString());
              } finally {
            if (connection != null) {
                try {
                    connection.close();
                } catch (JMSException e) {}   }}}}
```

3.3.4. EJB egoeradun sesio-bean baten garapena

EJB osagaiak erabilita garatu bada aplikazio informatikoa, ohikoa izaten da bezeroaren urruneko deiari ongi etorria ematea zerbitzarian sesio-bean bat eraikita. Halaber, sesio-beana diseinatzean kontuan hartuko da, deia egikaritu ondoren, zerbitzarian ireki dugun sesioa itxi egin behar den ala ez. Horrek, sesio-beanaren izaera baldintzatuko du. Batzuetan, deia jaso eta egikaritu ondoren zerbitzarian, erabili dugun beana *bizirik* mantentzea interesatzen zaigu, ez baitugu sesioa bukatu nahi. Bean berak, bezeroaren dei gehiago jasoko ditu eta dei bakoitzak beanaren aurreko egoera ezagutu nahi izango du. Hori egiteko, egoeradun sesio-bean bat programatuko da zerbitzariari adierazteko ezen bezeroaren dei bat baino gehiago prozesatu nahi dugula sesio berean.

Prozesu guztia nola eratzen den zehatz-mehatz aztertzeko, adibide bat aurkeztuko dugu ondoko lerroetan. Adibidean, egoeradun sesio-bean bat diseinatuko dugu bezeroaren eskaerak prozesatzeko. Bezeroaren aldean, aplikazio informatikoaren erabiltzaileak liburu batzuk eskatuko dizkio urruneko zerbitzariari eta, horrek, eskaera jaso, gorde eta eskatzen zaizkion eguneraketak gauzatuko ditu. Eskaera ondo prozesatzen ari dela ziurtatzeko, edozein unetan, eskaeraren edukia zein den eska diezaioke bezeroak zerbitzariari.

Eskaeraren edukia, `Vector`[38] motako objektu batean biltegiratuta edukiko duenez beanak zerbitzarian, informazioa berehala eskuratu eta bezeroari bidaliko dio.

Has gaitezen, beraz, beanak behar duen EJB azpiegitura teknologikoa eraikitzen, hau da, `EJBHome` motako objektu bat eta `EJBObject` motako beste bat diseinatzen. Lehenengoak, faktoria-lanak gauzatuko ditu eta EJB objektuak instantziatzeko balioko digu. Lana ondo taxutzeko, `create()` metodoa gainkargatu egingo dugu eta aukera emango diogu bezeroari era bitara deitzeko beanaren `ejbCreate()` metodoari. Batean, nahikoa da bezeroaren izena jartzea deia egikaritzeko eta, bestean, izenari, bezeroaren identifikadorea gehituko zaio deian:

38. Gauzak ez korapilatzeko, informazioaren iraunkortasuna, `Vector` motako objektu batekin kudeatuko dugu. Jakina, horrela, bezeroaren eskaeraren informazioaren iraupena zeharo murrizten dugu. Egoera erreal batean, iraunkortasuna, JDBC APIa edota entitate-bean bat erabilita gauzatuko litzateke.

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface EskaeraHome extends EJBHome {

    Eskaera create(String bezeroIz) throws RemoteException,
CreateException;
    Eskaera create(String bezeroIz, String id) throws RemoteException,
                                                    CreateException; }
```

Faktoria eraiki ondoren, urruneko bezeroak eskura izango dituen metodoak antolatuko dira. Horrela, bezeroaren deia, bean-instantzia egokira bideratu ahal izango du EJB edukiontziak inongo arazorik gabe. Dakigun bezala, deiaren eta beanaren kateaketa EJB objektuak erabilita kudeatuko ditu edukiontziak. Beraz, interfaze bat diseinatu behar dugu, `EJBObject` interfazea hedatzen duena eta bezeroak erabiliko duena beanaren metodoetara atzitzeko:

```
import java.util.*;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Eskaera extends EJBObject {

   public void gehituLiburua(String izenburua) throws RemoteException;
   public void ezabatuLiburua(String izenburua) throws LiburuSalbuespena,
RemoteException;
   public Vector lortuEskatutakoa() throws RemoteException;  }
```

EJB azpiegitura eraikita, has gaitezen buru-belarri aplikazio informatikoak beharko duen sesio-beanaren diseinua antolatzen. Esandakoak esanda, gutxi gelditzen zaigu hemen gehitzeko. Bakarrik aipatu ezen EJB objektuak agerian uzten dituen metodoek (`gehituLiburu()` eta `ezabatuLiburu()`) `Vector` klasearen baliabideak erabiliko dituztela eskaeraren liburuak kudeatzeko, delegatuz lan guztia `Vector` klasearen metodoetan.

Halaber, dagoeneko ez zaizu arraroa egingo, irakurle, bi `ejbCreate()` metodo aurkitzea beanean, bakoitzak, `EskaeraHome` interfazearen `create()` metodo bat ordezkatzen baitu.

```
import java.util.*;
import javax.ejb.*;

public class EskaeraBean implements SessionBean {

   String bezeroIzena;
   String bezeroIdent;
   Vector eskatutakoa;
```

```
   public void ejbCreate(String bezeroIz) throws CreateException {

      if (bezeroIz == null) {
        throw new CreateException("Zerbait sartu behar duzu
izenean.");
      }
      else {
         bezeroIzena = bezeroIz;
      }

      bezeroIdent = "0";
      eskatutakoa = new Vector();
   }

   public void ejbCreate(String bezeroIz, String id) throws
CreateException {

      if (bezeroIz == null) {
        throw new CreateException("Zerbait sartu behar duzu
izenean.");
      }
      else {
         bezeroIzena = bezeroIz;
      }

      IdenEgiaztatzailea IdAzterketa = new IdenEgiaztatzailea();
      if (IdAzterketa.egiaztatu(id)) {
         bezeroIdent = id;
      }
      else {
         throw new CreateException("Gaizki identifikadorea: " + id);
      }

      eskatutakoa = new Vector();
   }

   public void gehituLiburua(String izenburua) {

      eskatutakoa.addElement(izenburua);
   }
public void ezabatuLiburua(String izenburua) throws LiburuSalbuespena
{

      boolean emaitza = eskatutakoa.removeElement(izenburua);
      if (emaitza == false) {
         throw new LiburuSalbuespena(izenburua + " ez dago
eskaeran.");
      }
   }
```

```
   public Vector lortuEskatutakoa() {
      return eskatutakoa;
   }

   public EskaeraBean() {}
   public void ejbRemove() {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void setSessionContext(SessionContext sc) {}  }

public class IdenEgiaztatzailea {

    public IdenEgiaztatzailea() {
    }

    public boolean egiaztatu(String id) {

       boolean emaitza = true;
       for (int i = 0; i < id.length(); i++) {
         if (Character.isDigit(id.charAt(i)) == false)
            emaitza = false;
       }
       return emaitza;  }}
```

Aplikazio informatikoak erabiliko duen sesio-beana diseinatuta, zerbitzarian kokatu behar dugun azpiegitura guztia eraikita dago bezeroaren deia prozesatzeko. Zerbitzaria martxan jarri eta EJB edukiontzia prest gelditzuko da bezeroaren deien zain.

Bezeroaren aldean, beana atzitu nahi badugu, lehenengo eta behin, testuingurua eraiki beharko dugu izen desberdinen loturak eratzeko eta, bigarrenez, bilatu egin beharko dugu JNDI APIa erabilita, izen zehatzarekin etiketatuta dagoen objektua. Ondo badoaz gauzak `EskaeraHome` motako objektu baten erreferentziarekin bat egingo dugu.

`EskaeraHome` motako objektuaren `create()` metodoa egikarituta, erabiliko duen sesio-beana instantziatzen du bezeroak eta `Eskaera` motako `EJBObject` objektua uzten du eskuragarri atzipenak egiteko beanaren instantziara.

Konexioa, instantziazioa eta lotura guztiak eratu ondoren, beana prest gelditzen da bezeroaren deiak prozesatzeko. Orduan, eraiki berri den EJB objektuaren metodoak egikarituko ditu bezeroak bere eskaera moldatzen joateko.

```
import java.util.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
```

```
public class EskaeraBezeroa {

   public static void main(String[] args) {
       try {
           Context initial = new InitialContext();
           Object objref =
initial.lookup("java:comp/env/ejb/SimpleEskaera");

           EskaeraHome home =
               (EskaeraHome)PortableRemoteObject.narrow(objref,
                                          EskaeraHome.class);

           Eskaera eskaeraZehatza = home.create("Irakurle
Porrokatua","123");

           eskaeraZehatza.gehituLiburua("Lagun izoztua");
  eskaeraZehatza.gehituLiburua("Historia de Navarra el Estado Vasco");
           eskaeraZehatza.gehituLiburua("El hombre duplicado");
           eskaeraZehatza.gehituLiburua("El mal de Montano");
           eskaeraZehatza.gehituLiburua("Soinujolearen semea");

           Vector liburuZerrenda = new Vector();
           liburuZerrenda = eskaeraZehatza.lortuEskatutakoa();

           Enumeration enumer = liburuZerrenda.elements();
           while (enumer.hasMoreElements()) {
              String izenburua = (String) enumer.nextElement();
              System.out.println(izenburua);
           }

           eskaeraZehatza.ezabatuLiburua("Un Rey golpe a golpe");
           eskaeraZehatza.remove();

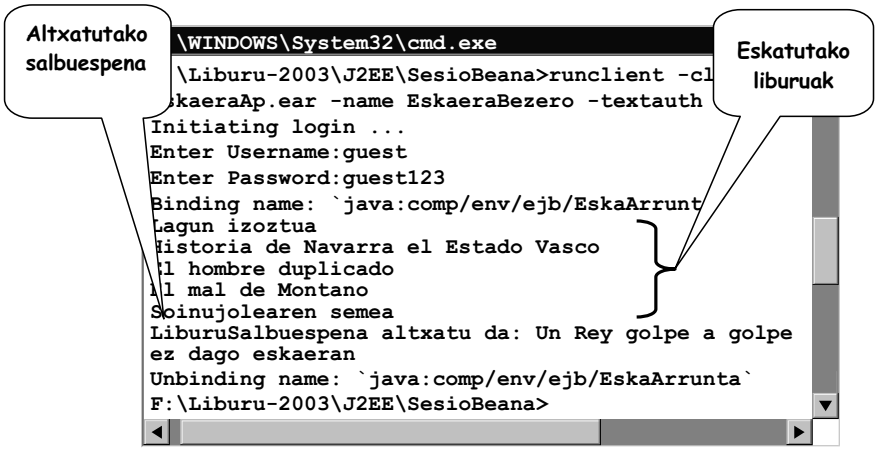
           System.exit(0);
       } catch (LiburuSalbuespena ex) {
      System.err.println("LiburuSalbuespena altxatu da: " +
ex.getMessage());
           System.exit(0);
       } catch (Exception ex) {
           System.err.println("Ezusteko salbuespena eman da!");
           ex.printStackTrace();
           System.exit(1);   }} }

public class LiburuSalbuespena extends Exception {

    public LiburuSalbuespena() {
    }

    public LiburuSalbuespena(String msg) {
        super(msg);}}
```

Bezeroaren programa egikaritzen badugu J2EEren `runclient` baliabidea egikarituta, honako irteera hau lortuko dugu:



3.44. irudia. Adibidearen irteera `runclient` baliabidea egikarituta.

3.3.5. Entitate-beanak

3.3.5.1. Beanak kudeatutako iraunkortasuna

Beanak kudeatutako iraunkortasunean, entitate-beanak berak kudeatu beharko ditu iraunkortasunaren arlo guztiak (datu-baseen konexioak sortu, datu-basea bilatu, SQL aginduak eratu, eta abar), hau da, programatzaileak berak, esplizituki programatu beharko ditu enpresaren informazio-sistemara gauzatuko diren atzipenak.

Bestela, aplikazio informatikoaren objektuak datu-base erlazionalen entitateekin mapeatzen direnean kontuan hartuko da entitate arteko erlazioak kudeatu egingo direla datu-basean eta entitate-beanetan. Aipatutako ezaugarriek eragina izango dute entitate-beanaren diseinuan, zeren azpiegitura egokia prestatu behar baita aurre egiteko egoera berri horri.

Gauzak poliki-poliki azaltzeko, adibide ezagun bat aztertuko dugu zeinak entitate-beanaren xehetasunak azaleratuko dituen. Adibideetan, salerosketekin ari garenez, lehenago sesio-beanean aztertu dugun salerosketaren adibidea hedatu eta entitate-beanetara egokituko dugu. Horretarako, eraiki nahi dugun aplikazio informatikoak Bezero-Eskaera-Artikulu (**3.45. irudia**) hirukotea erabiliko du oinarri gisa bezeroaren eskaera egituratzeko aplikazioan. **3.45. irudia**k argi erakusten duenez entitateen artean, bi erlazio mota (*1:n* eta *n:n*) erabiliko dira bezeroaren eskaeraren semantika guztia jasotzeko. Batekin (*1:n*), bezeroak *n* eskaera eduki ditzakeela adierazten da eta eskaera bat beti dela bezero bakarrarena. Bestearekin (*n:n*), bezeroaren eskaera *n* artikuluz osatua dagoela eta artikulu bat *n* eskaeratan egituratu daitekeela.

Hirukotea horrela egituratuta, betekizunen azterketa egiten dugu eta erabakitzen dugu, diseinuko entitate bakoitza kudeatzeko, entitate-bean bat eratuko dugula entitatearen ezaugarri estatiko eta dinamiko guztiak tratatzeko. Geroxeago antzemango dugu, halaber, hasierako diseinuari beste bean bat gehiago gehitu behar zaiola *n:n* erlazioaren semantika eratzeko. Azken bean hori, sesio-bean eran antolatuko dugu bere bizi-zikloa erabat baldintzatua baitago Eskaera eta Artikulu entitate-beanen portaeretara.

Eskaeraren bizi-zikloa bezeroarekin eratzen hasten denez, has gaitezen bezero horren nondik norakoak aztertzen. Lehendabizi, beanekin ari garenez, diseinatu nahi dugun beanaren `EJBHome` eta `EJBObject` interfazeak sortu behar ditugu. Lehenengoak, dakigunez, faktoria-lanak beteko dituen objektua eraikitzeko balioko digu eta, bigarrenak, urruneko bezeroak eskura izango dituen metodoak antolatzeko, beharrezkoa den `EJBObject` motako objektua eraikitzeko. Hona hemen, besterik gabe, bataren eta bestearen iturburu-kodeak:

```
import java.util.*;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface BezeroHome extends EJBHome {

    public Bezero create(String BezeroId, String Izena)
        throws RemoteException, CreateException;

    public Bezero findByPrimaryKey(String BezeroId)
        throws FinderException, RemoteException;  }

                                oOo
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.util.*;

public interface Bezero extends EJBObject {

   public ArrayList getEskaeraIdak()
      throws RemoteException;

   public String getIzena()
      throws RemoteException;

   public void setIzena(String Izena)
      throws RemoteException;  }
```

Konturatu ezen EJB osagaiak eskuragarri uzten diola urruneko bezeroari aukera, jakiteko, Bezero osagaia kontsultatuta, zeintzuk diren orain arte bezeroaren eskaerak. Informazio hori, `ArrayList` motako objektu batean bidaliko dio EJB edukiontziak urruneko bezeroari.

Entitate-beanaren J2EE azpiegitura antolatuta, azal dezagun beana bera nola egongo den egituratuta. Behin eta berriz kontuan izan behar duguna honako hau da: eraikitzen ari garen beanen iraunkortasuna, beanak berak kudeatu beharko duela une oro, horrek suposatzen duen guztiarekin. Alegia, datu-basera egingo diren atzipen guztiak erabat kontrolpean izango ditu beanak baina, aldi berean, beana diseinatzen eta programatzen duen programatzaileak iturburu-kode[39] guztia idatzi beharko du kontrol hori gauzatzeko. Mamuak uxatzeko esan, ezen, JDBC APIa menderatzen duen programatzaileak ez duela inongo arazorik izango aipatutako iturburu-kodea programatzeko.

Badakigu lehendik, `EJBHome` motako objektuaren `create()` metodo bakoitzak bere ordezkoa izango duela beanean `ejbCreate()` gisa adierazita. Aldiz, oraingoan, kontuan hartu behar dugu, entitate-bean batekin ari garela eta horrek erabat bideratuko duela metodoaren exekuzioa. Lehenengo eta behin, beanaren egoera datu-basean biltegiratuko da iraunkortasuna emanez beanaren aldagaien balioei eta, aldi berean, beanaren aldagaiak ere hasieratu egingo dira. Datu-basean txertaketa ondo egin bada, datu-basean txertatu den tuplaren gako nagusia itzultzen dio metodoak edukiontziari eta edukiontziak (**3.38. irudia**) `EJBObject` motako objektu baten erreferentzia itzultzen dio urruneko bezeroari. Azaldutako guztia batera jartzen badugu, honako iturburu-kode hau ateratzen zaigu:

```
public String ejbCreate(String BezeroId, String Izena)
 throws CreateException {

 try {
    txertatuBezero(BezeroId, Izena);
 } catch (Exception ex) {
     throw new EJBException("ejbCreate: " +
        ex.getMessage());
 }

 this.BezeroId = BezeroId;
 this.Izena = Izena;
 return BezeroId;  }
```

Egin kontu, datu-basearekin izango diren komunikazio guztiak, datuak atzitzeko, ezabatzeko, sortzeko eta eguneratzeko, beanen metodo pribatuak erabilita eratuko direla. Metodo horietariko bat programatu ondoren, besteak, errenkan programa daitezke bata bestearen atzetik, hasierako patroi bera, indarrean jarrita. Hona hemen gure adibiderako erabiliko dugun lehen metodo pribatua:

39. Geroxeago ikusiko dugunez, beanaren iraunkortasunaren kudeaketa EJB edukiontziaren esku gelditzen bada, programatzaileak ez du arduratu behar datu-basera egiten diren atzipenak programatzeaz. Nahikoa izango da ezarpenerako deskriptoreaz baliatzea, era erabat deklaratiboan, funtzionalitate horiek gauzatzeko.

```
private void txertatuBezero (String BezeroId, String Izena)
 throws SQLException {

    String txertatuAgindua =
          "insert into Bezero values ( ? , ? )";
    PreparedStatement prepStmt =
          con.prepareStatement(txertatuAgindua);

    prepStmt.setString(1, BezeroId);
    prepStmt.setString(2, Izena);

    prepStmt.executeUpdate();
    prepStmt.close();   }
```

Hasieran, egikaritu behar den SQL aginduaren txantiloi orokorra prestatzen du metodoak, `String` motako objektu batean. Gero, agindua egikaritu baino lehen, `PreparedStatement` motako objektu bat eraikitzen da eta objektuaren `setString()` metodoaz baliatuta balioak esleitzen dizkiogu txantiloi orokorraren `?` ikur bakoitzari. Bukatzeko, SQL agindua egikaritzen da, `PreparedStatement` motako objektuaren `executeUpdate()` metodoa egikarituta.

Beana eraiki ondoren, prest dago urruneko bezeroaren deiak egikaritzeko. Batzuetan, baina, egoera horretan, entitate-beanaren aldagaien balioak sinkronizatu egin behar ditu edukiontziak, memorian duguna eta datu-basean biltegiratuta duguna bat etor daitezen. Sinkronizazio-prozesua bi norabideetan ondo kudeatzeko, beanak bi metodo berezi ditu: `ejbLoad()` eta `ejbStore()`. Lehenengoak, informazioa irakurriko du datu-basetik une horretan bean-instantziak duen informazioa eguneratzeko. Bigarrenak, datu-basean biltegiratuta dagoen informazioa eguneratuko du une horretan bean-instantziak duen informazioa erabilita. Gure adibidean, `ejbLoad()` metodoak, bezeroaren informazio pertsonala eta bezero horrek dituen eskaerak eskuratuko ditu. Aldiz, `ejbStore()` metodoak, bezeroari buruz dugun informazio pertsonala eguneratuko du datu-basean beanaren instantziaren informazioa erabilita. Halaber, beana aktibatzeko eta pasibatzeko (**3.39. irudia**) bi metodo horietaz baliatuko da edukiontzia, beharrak hala agintzen dionean.

Bezero entitate-beanaren ezaugarri guztiak batera jartzen baditugu, honako iturburu-kode hau izango du beanak:

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class BezeroBean implements EntityBean {
```

```
private String BezeroId;
private String Izena;
private ArrayList EskaeraIdak;
private Connection con;
private String dbIzena = "java:comp/env/jdbc/EskaerakDB";
private EntityContext context;
private EskaeraHome EskaeraHome;

public ArrayList getEskaeraIdak() {
   return EskaeraIdak;
}

public String getIzena() {

   return Izena;
}

public void setIzena(String Izena) {

   this.Izena = Izena;
}

public String ejbCreate(String BezeroId, String Izena)
    throws CreateException {

    try {
       txertatuBezero(BezeroId, Izena);
    } catch (Exception ex) {
        throw new EJBException("ejbCreate: " +
           ex.getMessage());
    }

    this.BezeroId = BezeroId;
    this.Izena = Izena;
    return BezeroId;
}

public String ejbFindByPrimaryKey(String primaryKey)
   throws FinderException {

   boolean emaitza;

   try {
      emaitza = selectByPrimaryKey(primaryKey);
    } catch (Exception ex) {
        throw new EJBException("ejbFindByPrimaryKey: " +
           ex.getMessage());
    }

   if (emaitza) {
      return primaryKey;
```

```
      }
      else {
         throw new ObjectNotFoundException
     ("Identifikadore honekin: " + primaryKey + " ez da tuplarik
aurkitu.");
      }
   }

   public void ejbRemove() {

      try {
         ezabatuBezero(BezeroId);
       } catch (Exception ex) {
           throw new EJBException("ejbRemove: " +
              ex.getMessage());
       }
   }
public void setEntityContext(EntityContext context) {

      this.context = context;
      EskaeraIdak = new ArrayList();

      try {
         konexioaEratu();
         Context initial = new InitialContext();
         Object objref =
initial.lookup("java:comp/env/ejb/Eskaera");

         EskaeraHome =
(EskaeraHome)PortableRemoteObject.narrow(objref,
                                          EskaeraHome.class);
      } catch (Exception ex) {
          throw new EJBException("setEntityContext: " +
             ex.getMessage());
      }   }

   public void unsetEntityContext() {

      try {
         con.close();
      } catch (SQLException ex) {
          throw new EJBException("unsetEntityContext: " +
ex.getMessage());
      }
   }

   public void ejbActivate() {

      BezeroId = (String)context.getPrimaryKey();
   }
```

```
   public void ejbPassivate() {

      BezeroId = null;
   }
 public void ejbLoad() {

        try {
         kargatuBezeroa();
         kargatuEskaeraIdak();
       } catch (Exception ex) {
           throw new EJBException("ejbLoad: " +
              ex.getMessage());
       }     }

   private void kargatuEskaeraIdak() {

      EskaeraIdak.clear();

      try {
         Collection c = EskaeraHome.findByBezero(BezeroId);
         Iterator i=c.iterator();

         while (i.hasNext()) {
            Eskaera Eskaera = (Eskaera)i.next();
            String id = (String)Eskaera.getPrimaryKey();
       System.out.println("Gehitzen identifikadorea: " + id + "
zerrendara");
            EskaeraIdak.add(id);
         }

      } catch (Exception ex) {
          throw new EJBException("Salbuespena kargatuEskaeraIdak: " +
             ex.getMessage());
       }
   }

   public void ejbStore() {

        try {
         gordeBezeroa();
       } catch (Exception ex) {
           throw new EJBException("ejbStore: " +
              ex.getMessage());
       }   }

   public void ejbPostCreate(String BezeroId, String Izena) { }

/*************************************************************/
/********************* Datu-basera atzipenak *****************/
/*************************************************************/
private void konexioaEratu() throws NamingException, SQLException {
```

```
      InitialContext ic = new InitialContext();
      DataSource ds = (DataSource) ic.lookup(dbIzena);
      con =  ds.getConnection();
   }

   private void txertatuBezero (String BezeroId, String Izena)
       throws SQLException {

          String txertatuAgindua =
                "insert into Bezero values ( ? , ? )";
          PreparedStatement prepStmt =
                con.prepareStatement(txertatuAgindua);

          prepStmt.setString(1, BezeroId);
          prepStmt.setString(2, Izena);

          prepStmt.executeUpdate();
          prepStmt.close();
             }

   private boolean selectByPrimaryKey(String primaryKey)
      throws SQLException {

      String bilatuAgindua =
            "select Bezeroid " +
            "from Bezero where Bezeroid = ? ";
      PreparedStatement prepStmt =
            con.prepareStatement(bilatuAgindua);
      prepStmt.setString(1, primaryKey);

      EmaitzaSet rs = prepStmt.executeQuery();
      boolean emaitza = rs.next();
      prepStmt.close();
      return emaitza;
   }

private void ezabatuBezero(String BezeroId) throws SQLException {

      String ezabatuAgindua =
            "delete from Bezero  " +
            "where Bezeroid = ?";
      PreparedStatement prepStmt =
            con.prepareStatement(ezabatuAgindua);

      prepStmt.setString(1, BezeroId);
      prepStmt.executeUpdate();
      prepStmt.close();
   }

   private void kargatuBezeroa() throws SQLException {
```

```
      String bilatuAgindua =
            "select Izena " +
            "from Bezero where Bezeroid = ? ";
      PreparedStatement prepStmt =
            con.prepareStatement(bilatuAgindua);

      prepStmt.setString(1, BezeroId);

      EmaitzaSet rs = prepStmt.executeQuery();

      if (rs.next()) {
         Izena = rs.getString(1);
         prepStmt.close();
      }
      else {
         prepStmt.close();
         throw new NoSuchEntityException("Bezeroaren tupla : " +
BezeroId +
            " ez da aurkitzen datu-basean.");
      }
   }
     private void gordeBezeroa() throws SQLException {

      String eguneratuAgindua =
            "update Bezero set Izena =  ? " +
            "where Bezeroid = ?";
      PreparedStatement prepStmt =
            con.prepareStatement(eguneratuAgindua);

      prepStmt.setString(1, Izena);
      prepStmt.setString(2, BezeroId);
      int lerroKopuru = prepStmt.executeUpdate();
      prepStmt.close();

      if (lerroKopuru == 0) {
         throw new EJBException("Ezin da bezero txertatu datu-basean:
" +
            BezeroId);   } }
} // BezeroBean
```

Konturatu ezen entitate-beanak `EskaeraHome` motako objektua erabiliko duela bezeroaren eskaeren identifikadoreak eskuratzeko Eskaera entitate-beanetik, `ejbFindByBezero()` metodoa egikarituta. `Collection` motako edukiontzi batean itzultzen ditu metodoak bezeroaren eskaeren identifikadoreak. Hona hemen nola dagoen inplementatuta metodoa, `Eskaera` entitate-beanean:

```
public Collection ejbFindByBezero(String BezeroId)
      throws FinderException {

      Collection emaitza;

      try {
         emaitza = aukeratuBezero(BezeroId);
       } catch (Exception ex) {
           throw new EJBException("ejbFindByBezero: " +
              ex.getMessage());
       }
       return emaitza;
   }
private Collection aukeratuBezero(String BezeroId)
      throws SQLException {

      String bilatuAgindua =
            "select Eskaeraid " +
            "from Eskaera where Bezeroid = ? ";
      PreparedStatement prepStmt =
            con.prepareStatement(bilatuAgindua);

      prepStmt.setString(1, BezeroId);
      EmaitzaSet rs = prepStmt.executeQuery();
      ArrayList a = new ArrayList();

      while (rs.next()) {
         String id = rs.getString(1);
         a.add(id);
      }

      prepStmt.close();
      return a;   }
```

Bestela, `Eskaera` entitate-beanak `Bezero` entitate-beanaren egitura bera izango duenez, ez gara alferrik temoso jarriko bigarren bean horren ezaugarriak azaltzen eta bere horretan utziko dugu beanaren azalpena.

Beanak kudeatutako iraunkortasunari buruzko atal hau hankamotz gelditu ko litzateke *n:n* erlazio mota aztertuko ez bagenu. Gogoratu erlazio mota hori, gure adibidean, bezeroaren eskaeren eta artikuluen artean gauzatzen dela, eskaera batek *n* artikulu izango baititu eta artikulu bat *n* eskaeratan ager baitaiteke.

Ikusitakoak ikusita, argi geratu da hiru entitate-bean behar genituela bezeroaren, eskaeraren eta artikuluaren informazioa kudeatzeko, baina, zer gertatzen da informazioa erlazioak berak sortzen duenean? Adibidez, non kokatuko genuke bezeroak eskaera bakoitzean eskatzen duen artikulu bakoitzaren kopurua? Eta, artikulu horri eskaera horretan aplikatzen zaion beherapena, non joango litzateke? Estrategia egokiena informazio hori biltegiratzeko datu-basean, taula bat sortzea da.

Bezeroak egiten duen eskaeraren eta eskatzen dituen artikuluen artean kokatzen denez taula berria, bi entitate horien identifikadoreekin lan egingo du taulak atzipenak egiteko datu-basera. Bestalde, aipatutako bi entitate horien bizi-zikloek erabat baldintzatuta dute taula kudeatzeko diseinatu behar dugun beanaren bizi-zikloa, taulan eraiki behar dugun edozein tuplaren informazio nagusi eta garrantzitsuena beste bi entitateetatik lortzen baita. Gauzak horrela, taularen tuplak kudeatzeko erabiliko dugun beana, sesio-beana izango da eta baliabideak eskainiko ditu bezeroaren eskaeraren artikuluak txertatzeko, ezabatzeko eta atzitzeko.

Azter dezagun txertaketa bat nola antolatuko zen:

```
  public void txertatuEA(String artikuluId, String eskaeraId) {

   try {
      txertatuInformazioa(artikuluId, eskaeraId);
   } catch (Exception ex) {
       throw new EJBException("txertatuEA: " +
          ex.getMessage());    }}

  private void txertatuInformazioa(String artikuluId, String
eskaeraId)
   throws SQLException {

      String txertatuAgindua =
            "insert into eskaerArtikulu values ( ? , ? )";
      PreparedStatement prepStmt =
            con.prepareStatement(txertatuAgindua);

      prepStmt.setString(1, artikuluId);
      prepStmt.setString(2, eskaeraId);

      prepStmt.executeUpdate();
      prepStmt.close();  }
```

Begi-bistakoa da, txertaketak, antz handia duela orain arte egindakoarekin. Txertaketa mota horrek, oraintsu aipatutako arrazoiengatik, bi identifikadore beharko ditu artikulua txertatzeko eskaeran. Metodoen beste alde funtzional guztia berdin-berdin eratuko dugu artikulua txertatzeko datu-basean.

Informazioa ezabatu nahi dugunean, estrategia bera erabiliko dugu baina kontuan izan beharko dugu zer ezabatu nahi dugun. Batzuetan, eskaeraren artikulu zehatz bat ezabatuko dugu, beste batzuetan, eskaeraren artikulu guztiak edo agian, artikulu zehatz bat agertzen den eskaera guztiak. Konbinazioak asko izan daitezke eta domeinuaren kudeaketa-arauek nabarmenduko dituzte zeintzuk programatuko diren aplikazioan.

Atzipenak bideratzeko erabiliko den estrategia zertxobait aldatuko da: prozesua, beanak berak kudeatu beharrean, `Eskaera` eta `Artikulu` entitate-beanen esku utziko da ardura guztia. Horrela, eskaeraren artikuluak atzitzeko, bai

`Eskaera` beanak eta bai `Artikulu` beanak, sesio-beana erabiliko dute bitartekarilanak gauzatzeko:

```
private void eskuratuEskaeraArtikuluIdak() {

      try {
         EskaeraArtikulu eskaeraArtikulu =
eskaeraArtikuluHome.create();
         ArrayList a =
eskaeraArtikulu.eskuratuArtikuluIdak(eskaeraId);
         artikuluIdak.addAll(a);

      } catch (Exception ex) {
   throw new EJBException("eskuratuEskaeraArtikuluIdak: " +
            ex.getMessage());  } }
```

Eskaera beanak eskaeraren artikuluak kudeatzen dituen sesio-beana eraikitzen du `Home` objektuaren `create()` metodoa egikarituta eta `EskaeraArtikulu` motako `EJBObject` baten erreferentzia ematen dio entitate-beanari. Horrek, erreferentzia hori erabilita, sesio-beanaren `eskuratuArtikuluIdak()` metodoa egikaritzen du eta sesio-beanak `ArrayList` motako objektu batean bilatu dituen eskaeraren artikuluen identifikadoreak itzultzen ditu:

```
      public ArrayList eskuratuArtikuluIdak(String eskaeraId) {

       try {
          return aukeratuArtikulua(eskaeraId);
       } catch (Exception ex) {
           throw new EJBException("eskuratuArtikuluIdak: " +
              ex.getMessage());  }}

      private ArrayList aukeratuArtikulua(String eskaeraId)
            throws SQLException {

       String aukeratuAgindua =
            "select artikuluid " +
            "from eskaerArtikulu where eskaeraid = ? ";
       PreparedStatement prepStmt =
            con.prepareStatement(aukeratuAgindua);

       prepStmt.setString(1, eskaeraId);
       ResultSet rs = prepStmt.executeQuery();
       ArrayList a = new ArrayList();

       while (rs.next()) {
            String id = rs.getString(1);
            a.add(id);
      }

      prepStmt.close();
      return a;   }
```

`Artikulu` entitate-beana erabilita atzitu nahi badugu sesio-beanera antzeko eragiketak gauzatzeko, aplikatuko den estrategia bera izango da, baina eskaeraren artikuluaren ikuspegitik aztertuta.

3.3.5.2. Edukiontziak kudeatutako iraunkortasuna

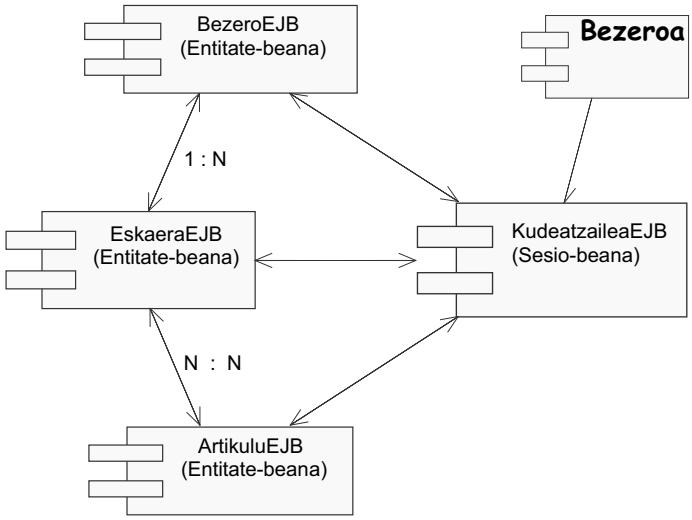
Honezkero, irakurle, pentsatzen arituko zinen, EJB edukiontzi eder bat hor izanda begirale, nola ez zitzaion inori otu zergatik ez zuen edukiontziak berak kudeatzen entitate-beanen instantziek behar zuten iraunkortasuna. Horrela, objektuen eta datu-baseen arteko mapaketaren ardura ez zen programatzailearen eginkizuna izango, edukiontziarena baizik. Egoera berri horretan, beanaren semantikaz soilik arduratuko litzateke programatzailea, alegia, domeinuaren logikaz.

Entitate-beanen diseinatzaileak ere ondorio horretara heldu ziren eta galdera horri erantzuna bilatu nahian beste iraunkortasun mota bat proposatu zuten: edukiontziak kudeatutako iraunkortasuna.

Hiru hitz horiek ederki asko sintetizatzen dute esamoldearen esanahia: edukiontziak hartuko du bere gain ardura guztia beanaren iraunkortasunaren alde guztiak kudeatzeko. Bera arduratuko da beanaren aldagaien metodo atzitzaileak (`get/set`) inplementatzeaz, baita beanak beste beanekin dituen erlazioak kudeatzeaz ere. Baina aldaketa handiena, beana datu-basearekin harremanetan jartzen denean gertatzen da, informazioa txertatzeko, ezabatzeko, eguneratzeko edo eskuratzeko. Zeregin hori, erabat gardena bihurtuko zaio programatzaileari eta ez du inongo konexiorik, bilaketarik edo SQL agindurik programatu beharko, informazioa datu-basetik atzitzeko. Edukiontzia bera arduratuko da lan horretaz.

Agian, dagoeneko norbaitek pentsatuko du aipatu dugun guztiak aldaketa handiak ekarri dizkiola urruneko bezeroari beanaren metodoak egikaritzeko orduan. Zorionez, bai urruneko bezeroentzat eta bai bezero lokalentzat, iraunkortasun mota *ezkutatuta* geldituko da eta ez dira jabetuko zein ari den aplikatzen zerbitzaria une bakoitzean.

Urruneko interfazea, interfaze lokala, Home interfazea eta Local Home interfazeak, bai iraunkortasun mota batean eta bai bestean, berdinak izango dira bezeroentzat eta batetik bestera edozein unetan jauzi egin dezake programatzaileak bataren eta bestearen eraginkortasuna neurtzeko. Abantaila horiek ditugu, EJB osagaiak erabilita softwarea eraikitzen denean.



3.45. irudia. Adibidearen domeinuko osagaiak.

Aurkeztu ditugun desberdintasunak zehatz-mehatz aztertzeko, lehenago esku artean erabili dugun adibidea berreskuratuko dugu eta **3.45. irudia**n agertzen den diseinua inplementatzen saiatuko gara. Diseinuak ez du aldaketa sakonik proposatzen, aplikazioaren bezeroak sesio bat irekitzen duenean zerbitzariarekin harremanetan jartzeko, sesio horren bizi-zikloa sesio-bean batek kontrolatuko du eta sesio-bean hori izango da urruneko bezeroaren deiak jasotzen eta egikaritzen saiatuko dena. Adibidez, artikulu bat gehitu nahi badiogu bezeroaren eskaera bati eta artikulu hori oraindik ez badago eraikita, honako bi metodo hauek erabiliko lituzke sesio-beanak eragiketa eratzeko:

```
public void eraikiArtikulu(ArtikuluZehatz elementua) {

        System.out.println("KudeatzaileaBean
eraikiArtikulu");
        try {
            LocalArtikulu Artikulu =
ArtikuluHome.create(elementua.getId(),
      elementua.getIzena(),
elementua.getKokapena(),elementua.getSalneurri());
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
}}
public void gehituArtikulu(String ArtikuluId, String
EskaeraId) {

        System.out.println("KudeatzaileaBean
gehituArtikulu");
        try {
```

```
            LocalEskaera Eskaera =
EskaeraHome.findByPrimaryKey(EskaeraId);
            LocalArtikulu Artikulu =
ArtikuluHome.findByPrimaryKey(ArtikuluId);
            Eskaera.gehituArtikulu(Artikulu);
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }  }
```

Lehenengo metodoak (`eraikiArtikulu()`) `ArtikuluBean` entitate-beana eraikitzen du `ArtikuluHome` motako faktoria-objektuaren `create()` metodoa egikarituta eta `EJBLocalObject` motako `LocalArtikulu` objektu bat uzten du eskuragarri.

Bigarren metodoan (`gehituArtikulu()`), eskaera zehatz bati esleitzen zaio artikulua. Horretarako, bai eskaeraren eta bai artikuluaren `findByPrimaryKey()` metodo berezia egikaritzen da `EJBObject` motako bi objektuak eskuratzeko. Hori egin ondoren, artikulua eskaerari gehitzen zaio `EskaeraBean` entitate-beanaren `gehituArtikulu()` metodoa egikarituta:

```
public void gehituArtikulu(LocalArtikulu Artikulu) {

        System.out.println("EskaeraBean gehituArtikulu");
        try {
            Collection Artikuluak = getArtikuluak();
            Artikuluak.add(Artikulu);
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
}}
```

Metodoak, lehenengo eta behin, eskaeraren artikuluak eskuratzen ditu `getArtikuluak()` metodoa egikarituta, eta, gero, artikulua gehitzen du, eskuratutako `Collection` motako objektuaren `add()` metodoa egikarituta.

Aztertzen ari garen sesio-beanak beste hainbat metodo izango ditu eskaeraren artikuluetara atzitzeko eta informazio desberdinak eskuratzeko, hala nola eskaeraren artikuluak, artikulu guztiak, artikulu hori eskatu duten bezeroak, eta abar. Horrela gauzak, sesio-beanean honela multzokatuko ditugu beanaren metodoak:

1. Artikuluaren metodoak domeinuaren logikan.

   Urruneko bezeroak artikuluaren informazioa arakatu nahi duenean, multzo honetan jarriko ditugun metodoak izango ditu eskuragarri atzipenak gauzatzeko.

2. Eskaeraren metodoak domeinuaren logikan.

   Multzo honetan kokatuko ditugu urruneko bezeroak eskaeraren informazioa aztertzeko behar dituen metodoak.

3. Bezeroaren metodoak domeinuaren logikan.

   Hirugarren multzo honetan, eskaeraren bezeroaren informazioa eskuratzeko behar ditugun metodoak programatuko dira.

4. Sesio-beanaren metodoak.

   Jakin badakigu sesio-beana, zerbitzuak betetzen ari zaizkiola aplikazioaren entitate-bean bakoitzari baina ezin dugu ahaztu hura ere bean bat dela eta bere metodoak izango dituela beana eraikitzeko, ezabatzeko eta era aktiboan eta pasiboan jartzeko bizi-zikloa aurrera joan ahala. Horretarako, multzo honetan, beanaren metodoak kokatuko ditugu aipatutako eragiketak gauzatzeko.

5. Beanaren metodo pribatuak.

   Beanak bere eragiketak egiteko, metodo pribatuak erabiliko ditu ezkutatzeko metodo horien portaera. Metodo horiek berarenak eta berariazkoak direnez, ez ditugu eskuragarri jarri nahi inorentzat.

Sesio-beana horrela egituratuta, hona hemen urruneko bezeroa harremanetan jarriko duen beanaren iturburu-kodea:

```
package Kudeatzailea;

import java.util.*;
import javax.ejb.*;
import javax.ejb.*;
import javax.naming.*;
import util.*;
import Eskaera.*;

public class KudeatzaileaBean implements SessionBean {

    private  LocalArtikuluHome ArtikuluHome = null;
    private  LocalEskaeraHome EskaeraHome = null;
    private  LocalBezeroHome BezeroHome = null;

    // Artikuluaren metodoak domeinuaren logikan

        public void eraikiArtikulu(ArtikuluZehatz
elementua) {

        System.out.println("KudeatzaileaBean
eraikiArtikulu");
        try {
            LocalArtikulu Artikulu =
ArtikuluHome.create(elementua.getId(),
      elementua.getIzena(),
elementua.getKokapena(),elementua.getSalneurri());
        } catch (Exception ex) {
```

```
            throw new EJBException(ex.getMessage());  }}
public void gehituArtikulu(String ArtikuluId, String
EskaeraId) {

        System.out.println("KudeatzaileaBean
gehituArtikulu");
        try {
            LocalEskaera Eskaera =
EskaeraHome.findByPrimaryKey(EskaeraId);
            LocalArtikulu Artikulu =
ArtikuluHome.findByPrimaryKey(ArtikuluId);
            Eskaera.gehituArtikulu(Artikulu);
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }

    public void ezabatuArtikulu(String ArtikuluId) {

        System.out.println("KudeatzaileaBean
ezabatuArtikulu");
        try {
            LocalArtikulu Artikulu =
ArtikuluHome.findByPrimaryKey(ArtikuluId);
            Artikulu.remove();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }
public ArtikuluZehatz eskuratuArtikulu(String
ArtikuluId) {

        System.out.println("KudeatzaileaBean
eskuratuArtikulu");
        ArtikuluZehatz ArtikuluZehatz = null;
        try {
            LocalArtikulu Artikulu =
ArtikuluHome.findByPrimaryKey(ArtikuluId);
            ArtikuluZehatz = new
ArtikuluZehatz(ArtikuluId,
                Artikulu.getIzena(),
Artikulu.getKokapena(), Artikulu.getSalneurri());
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
        return ArtikuluZehatz;
    } // eskuratuArtikulu


    public ArrayList getEskaerakoArtikuluak(String
EskaeraId) {
```

```
        System.out.println("KudeatzaileaBean
getEskaerakoArtikuluak");
        Collection Artikuluak = null;

        try {
            LocalEskaera Eskaera =
EskaeraHome.findByPrimaryKey(EskaeraId);
            Artikuluak = Eskaera.getArtikuluak();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }

        return gordeArtikuluak(Artikuluak);
    } // getEskaerakoArtikuluak
    public ArrayList getBezeroarenEskaerak(String BezeroId)
{

        System.out.println("KudeatzaileaBean
getBezeroarenEskaerak");

        ArrayList elementuaList = new ArrayList();
        Collection Eskaerak = null;

        try {
            LocalBezero Bezero =
BezeroHome.findByPrimaryKey(BezeroId);
            Eskaerak = Bezero.getEskaerak();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }

        Iterator i = Eskaerak.iterator();
        while (i.hasNext()) {
            LocalEskaera Eskaera = (LocalEskaera) i.next();
            EskaeraZehatz elementua = new
EskaeraZehatz(Eskaera.getEskaeraId(),
                Eskaera.getIzena(),
Eskaera.getEskaeraData());
            elementuaList.add(elementua);
        }

        return elementuaList;
          } // getBezeroarenEskaerak
public ArrayList getBezeroIdArtikuluak(String BezeroId) {

        System.out.println("KudeatzaileaBean
getBezeroIdArtikuluak");
        Collection Artikuluak = null;
```

```
        try {
            LocalBezero Bezero =
BezeroHome.findByPrimaryKey(BezeroId);
            Artikuluak = ArtikuluHome.findByBezero(Bezero);
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }

        return gordeArtikuluak(Artikuluak);
    } // getBezeroIdArtikuluak

    public ArrayList eskuratuArtikuluGuztiak() {

        System.out.println("KudeatzaileaBean
eskuratuArtikuluGuztiak");
        Collection Artikuluak = null;

        try {
            Artikuluak = ArtikuluHome.findAll();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }

        return gordeArtikuluak(Artikuluak);
    } // eskuratuArtikuluGuztiak
public ArrayList eskuratuArtikuluarenBezeroak(String
ArtikuluId) {

        System.out.println("KudeatzaileaBean
eskuratuArtikuluarenBezeroak");
               ArrayList elementuaList = new ArrayList();
        Collection Bezeroak = null;

        try {
            LocalArtikulu Artikulu =
ArtikuluHome.findByPrimaryKey(ArtikuluId);
            Bezeroak = Artikulu.getBezeroak();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }

        Iterator i = Bezeroak.iterator();
        while (i.hasNext()) {
            LocalBezero Bezero = (LocalBezero) i.next();
            BezeroZehatz elementua = new
BezeroZehatz(Bezero.getBezeroId(),
                Bezero.getIzena(), Bezero.getHelbidea());
            elementuaList.add(elementua);
        }
        return elementuaList;
    } // eskuratuArtikuluarenBezeroak
```

```
 // Eskaeraren metodoak domeinuaren logikan

public void eraikiBezeroarenEskaera(EskaeraZehatz
elementua, String BezeroId) {

        System.out.println("KudeatzaileaBean
eraikiBezeroarenEskaera");
        try {
            LocalBezero Bezero =
BezeroHome.findByPrimaryKey(BezeroId);
            LocalEskaera Eskaera =
EskaeraHome.create(elementua.getId(),
                elementua.getIzena(),
elementua.getEskaeraData());
            Bezero.gehituEskaera(Eskaera);
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }
public void ezabatuEskaera(String EskaeraId) {

        System.out.println("KudeatzaileaBean
ezabatuEskaera");
        try {
                  LocalEskaera Eskaera =
EskaeraHome.findByPrimaryKey(EskaeraId);
            Eskaera.remove();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }

    public EskaeraZehatz eskuratuEskaera(String
EskaeraId) {

        System.out.println("KudeatzaileaBean
eskuratuEskaera");
        EskaeraZehatz EskaeraZehatz = null;
        try {
            LocalEskaera Eskaera =
EskaeraHome.findByPrimaryKey(EskaeraId);
            EskaeraZehatz = new
EskaeraZehatz(EskaeraId,
                Eskaera.getIzena(),
Eskaera.getEskaeraData());
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
        return EskaeraZehatz;
    }
```

```
// Bezeroaren metodoak domeinuaren logikan
    public void eraikiBezero(BezeroZehatz elementua) {

        System.out.println("KudeatzaileaBean
eraikiBezero");
        try {
            LocalBezero Bezero =
BezeroHome.create(elementua.getId(),
                elementua.getIzena(),
elementua.getHelbidea());
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }

    public void ezabatuBezero(String BezeroId) {

        System.out.println("KudeatzaileaBean
ezabatuBezero");
        try {
            LocalBezero Bezero =
BezeroHome.findByPrimaryKey(BezeroId);
            Bezero.remove();
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
    }
 public BezeroZehatz getBezero(String BezeroId) {

        System.out.println("KudeatzaileaBean getBezero");
        BezeroZehatz BezeroZehatz = null;
        try {
            LocalBezero Bezero =
BezeroHome.findByPrimaryKey(BezeroId);
            BezeroZehatz = new BezeroZehatz(BezeroId,
                Bezero.getIzena(), Bezero.getHelbidea());
        } catch (Exception ex) {
            throw new EJBException(ex.getMessage());
        }
        return BezeroZehatz;
    }

       // Sesio-beanaren metodoak
    public void ejbCreate() throws CreateException {

        System.out.println("KudeatzaileaBean ejbCreate");
        try {
            ArtikuluHome = lookupArtikulu();
            EskaeraHome = lookupEskaera();
            BezeroHome = lookupBezero();
```

```
        } catch (NamingException ex) {
            throw new CreateException(ex.getMessage());
        }
    }
public void ejbActivate() {

        System.out.println("KudeatzaileaBean ejbActivate");
        try {
            ArtikuluHome = lookupArtikulu();
            EskaeraHome = lookupEskaera();
            BezeroHome = lookupBezero();
        } catch (NamingException ex) {
            throw new EJBException(ex.getMessage());
        }
    }

    public void ejbPassivate() {

        ArtikuluHome = null;
        EskaeraHome = null;
        BezeroHome = null;
    }

    public KudeatzaileaBean() {}
    public void ejbRemove() {}
    public void setSessionContext(SessionContext sc) {}

    // Beanaren metodo pribatuak
    private LocalArtikuluHome lookupArtikulu() throws
NamingException {

        Context initial = new InitialContext();
        Object objref =
initial.lookup("java:comp/env/ejb/SimpleArtikulu");
        return (LocalArtikuluHome) objref;
   }
private LocalEskaeraHome lookupEskaera() throws
NamingException {

        Context initial = new InitialContext();
        Object objref =
initial.lookup("java:comp/env/ejb/SimpleEskaera");
        return (LocalEskaeraHome) objref;
    }

    private LocalBezeroHome lookupBezero() throws
NamingException {

        Context initial = new InitialContext();
        Object objref =
```

```
initial.lookup("java:comp/env/ejb/SimpleBezero");
        return (LocalBezeroHome) objref;
   }
private ArrayList gordeArtikuluak(Collection artikuluak) {
        ArrayList elementuaList = new ArrayList();
        Iterator i = artikuluak.iterator();

        while (i.hasNext()) {
            LocalArtikulu artikulu = (LocalArtikulu)
i.next();
         ArtikuluZehatz elementua = new
ArtikuluZehatz(artikulu.getArtikuluId(),
          artikulu.getIzena(), artikulu.getKokapena(),
artikulu.getSalneurri());
            elementuaList.add(elementua);
        }

   } // KudeatzaileaBean
```

Aztertzen ari garen adibidean, entitate-bean bakoitzaren nondik norakoak zehatz-mehatz azaltzeak luze joko luke eta gainera baten ezaugarriak aztertzea nahikoa izango litzaiguke besteenak ere ezagutzeko. Hori dela-eta, entitate-bean bakarra aukeratu eta hura aztertzeari ekingo diogu jarraian.

Aukeratuko dugun entitate-beana, artikulua kudeatzeko erabiliko den `ArtikuluBean` entitate-beana izango da. Eskaeraren artikuluaren entitate-beana, lehenago sesio-beanarekin egin dugun bezala, bost zatitan egituratuko dugu. Zati bakoitzak beanaren funtzionalitate bat inplementatuko du eta helburu zehatz batekin diseinatuko da.

```
    public abstract String getArtikuluId();
    public abstract void setArtikuluId(String id);

    public abstract String getIzena();
    public abstract void setIzena(String izena);

    public abstract String getKokapena();
    public abstract void setKokapena(String kokapena)

    public abstract double getSalneurri();
    public abstract void setSalneurri(double salneurri);
```

Lehenengo multzoan, atzipen-metodoak (`get/set`) erazagutzen ditugu beanaren eremu iraunkorrak atzitzeko. Konturatu erazagutu esaten dugula eta ez definitu, programatzaileak ez baititu programatu behar metodo horiek. Eta orduan, non daude adierazita eremu horiek eta nork programatzen ditu atzipen-metodo horiek? Ohi bezala, EJB edukiontzia datorkigu laguntza ematera.

Edukiontziak, ezarpenerako deskriptorea erabilita, jakin badaki ezen beanaren informazioa deskriptorearen XML fitxategian aurkituko duela, eta informazio horretaz baliatzen da beanaren atzipen-metodoak inplementatzeko:

```
<cmp-field>
  <description>no description</description>
        <field-name>kokapena</field-name>
      </cmp-field>
      <cmp-field>
  <description>no description</description>
  <field-name>izena</field-name>
</cmp-field>
<cmp-field>
  <description>no description</description>
  <field-name>ArtikuluId</field-name>
</cmp-field>
<cmp-field>
  <description>no description</description>
  <field-name>salneurri</field-name>
</cmp-field>
<primkey-field>ArtikuluId</primkey-field>
```

Ezarpenerako deskriptorearen informaziotik ondorioztatzen du EJB edukiontziak, halaber, beanaren iraunkortasuna kudeatzeko erabili behar duen gako nagusia `ArtikuluId` eremuak hornituko diola.

Bigarren multzoan, beste atzipen metodo batzuk erazagutzen ditugu. Metodo horiekin, beanak beste beanekin dituen erlazioak kudeatzen dira. Lehen bezala, metodo horien kudeaketa, EJB edukiontziaren esku gelditzen da eta ezarpenerako deskriptorea erabiliko du metodo horiek egoki inplementatzeko. Gure adibidean, artikulua kudeatzen duen entitate-beana eskaera kudeatzen duenarekin erlazionatuta (**3.45. irudia**) dagoenez, honako metodo hauek erazagutzen ditugu:

```
public abstract Collection getEskaerak();
public abstract void setEskaerak(Collection Eskaerak);
```

Ezarpenerako deskriptoreak, bere aldetik, honako beste informazio hau edukiko du, aplikazioaren erlazioak[40] kudeatzeko:

40. Gogoratu hauek direla aplikazioak edukiko dituen erlazioak:
Bezeroa – `1:N` – Eskaera – `N:N` – Artikulua

```
    <relationships>
    <description>Erlazio nagusiak</description>
    <ejb-relation>
    <description>Eskaera_Artikulu</description>
    <ejb-relation-name></ejb-relation-name>
    <ejb-relationship-role>
<ejb-relationship-role-name>EskaeraEJB</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <relationship-role-source>
        <ejb-name>EskaeraEJB</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>Artikuluak</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>
    <ejb-relationship-role-name>ArtikuluEJB</ejb-relationship-role-
     name>
      <multiplicity>Many</multiplicity>
      <relationship-role-source>
        <ejb-name>ArtikuluEJB</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>Eskaerak</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
</ejb-relation>
<ejb-relation>
  <description>Bezero-Eskaera</description>
  <ejb-relation-name></ejb-relation-name>
  <ejb-relationship-role>
<ejb-relationship-role-name>BezeroEJB</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>BezeroEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
       <cmr-field-name>Eskaerak</cmr-field-name>
       <cmr-field-type>java.util.Collection</cmr-field-type>
     </cmr-field>
   </ejb-relationship-role>
   <ejb-relationship-role>
    <ejb-relationship-role-name>EskaeraEJB</ejb-relationship-
role-name>
      <multiplicity>Many</multiplicity>
      <cascade-delete />
```

N:N

```
        <relationship-role-source>
          <ejb-name>EskaeraEJB</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>Bezero</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>
    </ejb-relation>
  </relationships>
```

Azter dezagun poliki, `<relationships>` elementua. Elementuaren barnean habiaratuta, beste bi `<ejb-relation>` elementu daude. Lehenengoak, *n:n* erlazioa kudeatzeko balioko digu eta bigarrenak *1:n*. Erlazio bakoitzaren nondik norakoak adierazteko, `<ejb-relationship-role>` elementu bat sortzen da. Rol bakoitzean, hiru elementu izango ditugu. Lehenengokoak, rol horren kardinalitatea (`Many/One`) erlazioaren alde horretan adieraziko du. Bigarrenak, rolaren iturburua non bilatzen den argitzen du entitate-bean mailan, eta, hirugarrenak, edukiontziak kudeatutako erlazioaren eremuaren izena eta mota zeintzuk izango diren esaten du.

Hirugarren multzoan, Select metodoak aurkitzen dira. Metodo horiek beanaren barnean erabiltzen dira bakarrik eta lagungarriak izaten dira beste metodo batzuentzat. Select metodoak ez dira eskuragarri jartzen interfazeetan eta ezkutatuta daude urruneko bezeroentzat. Urruneko bezeroek beanaren metodoak erabili behar dituzte, zeharka atzitzeko metodo horietara.

Laugarren multzoa eratzeko erabiliko ditugu metodo horiek eta eskuragarri utziko dizkiogu urruneko bezeroari `EJBObject` motako objektuan. Hona hemen kateatuta, hirugarren eta laugarren multzoen iturburu-kodea. Egin kontu ezen laugarren multzoko metodoek hirugarren multzoko Select metodoak[41] erabiltzen dituztela balioak eskuratzeko.

41. `SelectBezeroak()` metodoak honako sarrera hau izango du ezarpenerako deskriptorean:

```
<query>
        <description></description>
        <query-method>
          <method-name>ejbSelectBezeroak</method-name>
          <method-params>
            <method-param>Eskaera.LocalArtikulu</method-param>
          </method-params>
        </query-method>
        <result-type-mapping>Local</result-type-mapping>
   <ejb-ql>select distinct t.Bezero from Artikulu p, in (p.Eskaerak) as t
where p = ?1</ejb-ql>
      </query>
```

```
// Select metodoak barne-eragiketak gauzatzeko

    public abstract Collection
ejbSelectBezeroak(LocalArtikulu Artikulu)
        throws FinderException;

    public abstract Collection
ejbSelectHelbideak(LocalArtikulu Artikulu)
        throws FinderException;

     // Beanaren metodoak domeinuaren logikarekin erlazionatuta

public Collection getBezeroak() throws FinderException {
LocalArtikulu Artikulu =

(Eskaera.LocalArtikulu)context.getEJBLocalObject();
         return ejbSelectBezeroak(Artikulu);  }
    public Collection getHelbideak() throws FinderException
{   LocalArtikulu Artikulu =

(Eskaera.LocalArtikulu)context.getEJBLocalObject();
         return ejbSelectHelbideak(Artikulu); }}
```

Bilaketak eta hautaketak EJB-QL (Enterprise Java Bean Query Language) erabilita adierazten dira ezarpenerako deskriptorean. Bilaketak, `find` aurrizkia idatzita adierazten dira. Horrela, jakin nahi badugu artikulua eskatu duten bezeroak nortzuk diren, honako hau idatzi beharko genuke artikuluaren `Home` interfazean:

```
public Collection findByBezero(LocalBezero Bezero)
        throws FinderException;
```

Ezarpenerako deskriptoreak, bere aldetik, honako sarrera hau eduki beharko luke metodoa ondo egikaritzeko:

```
  <query>
  <description></description>
  <query-method>
    <method-name>findByBezero</method-name>
    <method-params>
      <method-param>Eskaera.LocalBezero</method-param>
    </method-params>
  </query-method>
  <ejb-ql>select distinct object(p) from Artikulu p,
  in (p.Eskaerak) as t where t.Bezero = ?1</ejb-ql>
</query>
```

Bukatzeko, `ArtikuluBean` entitate-beanak bere bizi-zikloa eratzeko behar dituen metodoak programatzen ditugu bosgarren eta azken multzoan:

```
// Bean-instantziaren metodoak

public String ejbCreate (String id, String izena, String kokapena,
        double salneurri) throws CreateException {

        System.out.println("ArtikuluBean ejbCreate");
        setArtikuluId(id);
        setIzena(izena);
        setKokapena(kokapena);
        setSalneurri(salneurri);
        return null;
    }

    public void ejbPostCreate (String id, String izena,
String kokapena,
        double salneurri) throws CreateException { }

    public void setEntityContext(EntityContext ctx) {
        context = ctx;
    }

    public void unsetEntityContext() {
        context = null;
    }

    public void ejbRemove() {
        System.out.println("ArtikuluBean ejbRemove");
    }

    public void ejbLoad() {
        System.out.println("ArtikuluBean ejbLoad");
    }
    public void ejbStore() {
        System.out.println("ArtikuluBean ejbStore");
    }

    public void ejbPassivate() { }
    public void ejbActivate() { }
```

Konturatu ezen orain ez dugula inongo SQL agindurik programatu behar entitate-beanean, iraunkortasuna kudeatzea edukiontziaren esku geltituko baita. Adibide argiena `ejbCreate()` metodoan dugu.

3.3.5.3. Aplikazio oso baten garapena EJB beanak erabilita

Laburpen txiki bat egingo bagenio orain arte egindakoari, berehala nabarmenduko litzateke eratu dugun estrategia bean mota desberdinak aurkezteko erabat inkrementala izan dela EJB osagaien aurkezpenean. Hasierako aurkezpenari beste atal bat eransten genion eta bigarren horri, bera iturburu hartuta, beste bat gehitzen

genion. Horrela egin dugu, batez ere, esku artean erabili dugun adibidearekin[42]. Hasieran, aplikazioa, sesio-bean xume bat eratuta eraiki dugu eta berehala konturatu gara ebazpenak mugak zituela, eta nabariak gainera. Bezeroaren eskaeraren iraunkortasuna, ziurtatu gabe gelditzen zen, `Vector` motako objektu bat erabiltzen baikenuen eskaera biltegiratzeko sisteman.

Adibidearen bigarren bertsioan, gauzak nabarmen hobetu dira, entitate-beanak erabili baititugu aplikazioa kudeatzeko. Dena dela, bigarren bertsio horretan, programatzaileak JDBC APIa menderatzen bazuen, ez zegoen arazo larririk aplikazioa programatzeko baina, beanak kudeatzen zuenez iraunkortasuna, programatzaileak berak programatu behar zituen datu-basearekin gauzatzen ziren eragiketa guztiak.

Hirugarren bertsioan, iraunkortasunaren kudeaketa, edukiontziari berari utzi diogu atsegin handiz eta gu bakarrik entitate- eta sesio-beanez arduratu gara. Azken bertsio horretan, gainera, apal bada ere, sesio- eta entitate-beanak harremanetan jarri ditugu.

Prozesua biribiltzeko, honako honetan, aplikazio erreal baten osagaien lehen diseinu (**3.46. irudia**) nagusi bat aurkeztuko dugu. Aplikazioa, orain arte bezala, bezeroaren eskaeren kudeaketaz arduratuko da baina oraingoan, zertxobait hedatuko ditugu betekizunak ezaugarri berriak itsatsita. Oraingo honetan, bezeroak ez du eskaera zuzenean egingo, lehenengo, aurrekontu bat eskatuko du eta gero, gauzatuko du eskaera. Aurrekontutik eskaerara doan urratsean, aurrekontuko artikuluen informazioa eguneratu egin dezake bezeroak, alegia, artikuluak kenduta eskaeratik, artikuluak itsatsita edo artikulu batzuen deskontu-baldintzak eguneratuta.

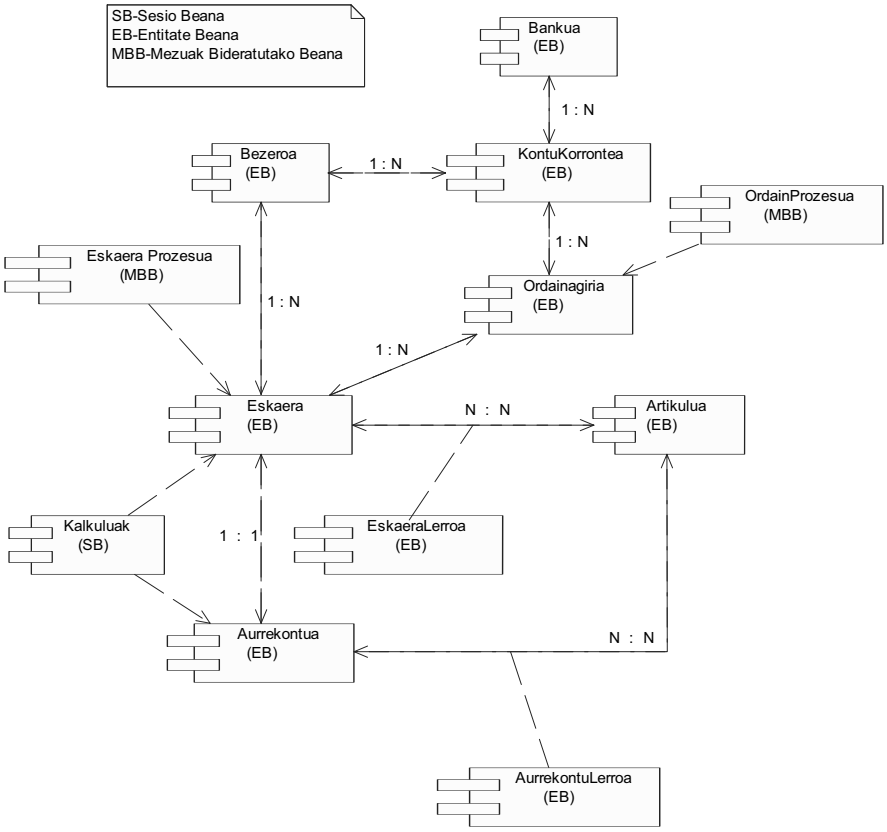
Eskaerak banatu ondoren, ordainagiriak bidaltzen zaizkio bezeroari, ordainketak gauzatzeko. Ordainketa, bezeroaren kontu-korrontea erabilita gauzatzen da.

Aipatutako prozesuaren aplikazio informatikoa eraikitzeko, EJB osgaiak erabiltzea erabaki dugu. Aplikazioaren informazio iraunkorra kudeatzeko, entitate-beanak erabiliko ditugu eta haien arteko erlazioak **3.46. irudia**n agertzen direnak izango dira. Bestalde, eskaera eta aurrekontua eraikitzen ari garenean egikaritzen diren kalkulu guztiak, salneurriak, deskontuak, zigorrak eta primak eskuratzeko eta taxutzeko, sesio-bean bat diseinatuko dugu, *bizirik* iraungo duena sesioa irekita dagoen bitartean.

Bezeroaren eskaeraren banaketa hasi baino lehen, bezeroaren adostasuna jaso behar du aplikazio informatikoak, adieraziz ados dagoela eskaeraren baldintzekin. Arazo hau kudeatzeko, mezuak bideratutako bean bat eraikitzen dugu zeina era

42. Adibideak, bezeroaren eskaerak kudeatzeko behar zuen aplikazio informatikoa eraikitzen zuen EJB osagaiak erabilita.

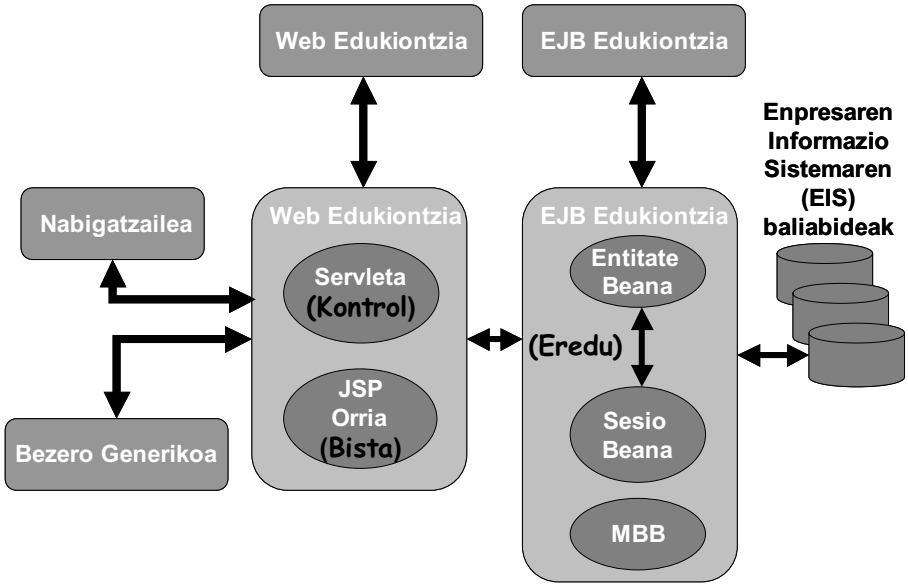
asinkronoan egikarituko den. Antzeko diseinua aplikatuko dugu bezeroaren ordainagirien kontrola eramateko eta ziurtatzeko bezeroaren kontu-korronteak diruz ondo elikatuta daudela.



3.46. irudia. Aplikazio errealaren diseinu orokorra.

Orain arte aztertu dugu zeintzuk eta zertarako diren aplikazioaren osagai nagusiak, eta erabaki dugu osagai horien kudeaketa EJB edukiontzi batek gauzatuko duela. Horrenbestez, domeinuaren logikaren zati nagusia eratuta dugu eta gelditzen zaiguna da aplikazioaren web arloa lantzea. Atal hori antolatzeko, Eredu-Bista-Kontrol (**3.47. irudia**) diseinu-patroi ezaguna erabiliko dugu.

Urruneko bezero generikoak eskaera bat luzatzen dionean aplikazio informatikoari, servlet batera bideratuko da eskaera, servletak erabaki dezan zer egin eskaera horrekin. Erabaki ondoren zer egin eskaerarekin, web edukiontzia harremanetan jartzen da EJB edukiontziarekin, eskaeraren domeinuaren logika egikaritzeko. Aplikazioaren beanak erabilita, urruneko bezeroaren deia prozesatzen du EJB edukiontziak eta emaitza web edukiontziari itzultzen dio. Horrek, gehienetan, JSP orri bat erabilita, emaitza, urruneko bezeroari bidaltzen dio deiaren bizi-zikloa itxiz.



3.47. irudia. Eredu-Bista-Kontrol diseinu-patroiaren EJB bertsioa.

3.3.6. J2EE diseinu-patroiak

J2EE diseinu-patroiak aurkezpen mailarako

Diseinu-patroiaren izena	Deskripzioa
Application Controller	Ekintzen eta bisten kudeaketa zentralizatu eta modularizatu egiten du.
Composite View	Bista konposatuak eraikitzen ditu azpibistak erabilita. Horretarako, edukiak eta diseinu-administratzaileak banatu egiten ditu.
Context Object	Egoera kapsulatzen du protokolo-independente eran aplikazioan zehar informazioa partekatzeko.
Dispatcher View	Domeinuaren prozesua baino lehen, bistaren prozesua eskatzen du.
Front Controller	Kontrolatzaile nagusi bat antolatzen du bezeroaren eskaeren kudeaketa gauzatzeko.
Intercepting Filter	Eskaeraren aurre-prozesua eta prozesu-ostea errazten ditu.
Service To Worker	Bistaren prozesua baino lehen, domeinuaren prozesua eskatzen du.
View Helper	Prozesuaren logika banandu egiten du aurkezpen mailatik, alegia, bistaren lana erraztu eta sinplifikatu egiten du.

3.48. taula. J2EE diseinu-patroiak aurkezpen mailarako.

J2EE diseinu-patroiak domeinuaren mailarako	
Diseinu-patroiaren izena	**Deskripzioa**
`Application Service`	Portaera gehitzen du zerbitzu uniforme bat eraikitzeko.
`Business Delegate`	Domeinura bideratzen den zerbitzu bat kapsulatu egiten du, adibidez, web eta EJB mailak bananduz.
`Business Object`	Domeinuaren datuak eta logika kapsulatu egiten ditu.
`Composite Entity`	Domeinuaren objektuen iraunkortasuna entitate-beanak erabilita ziurtatzen du.
`Service Locator`	Sinplifikatu egiten du bezeroen atzipena domeinuaren zerbitzuetara bilaketa zentralizatuz.
`Session Façade`	Domeinuaren objektu askoren lan-fluxua koordinatzen du objektu bakarra erabilita.
`Transfer Object`	Domeinuaren datuak *transferitzen* ditu arkitekturaren mailen artean.
`Transfer Object Assembler`	Multzokatu egiten du *Transfer Object* bat, datuiturri anitz erabilita.
`Value List Handler`	Emaitza multzo handi baten bilaketa- eta iterazio-prozesuak kudeatzen ditu.

3.49. taula. J2EE diseinu-patroiak domeinuaren mailarako.

J2EE Diseinu-patroiak integrazioaren mailarako	
Diseinu-patroiaren izena	**Deskripzioa**
`Data Acces Object(DAO)`	Datuetara bideratzen diren atzipenak kapsulatu eta era abstraktuan antolatzen ditu.
`Domain Store`	Iraunkortasun gardena eraikitzen du domeinuaren objektuentzat.
`Service Activator`	Atzipen asinkronoa antolatzen du aplikazioaren zerbitzuentzat.
`Web Service Broker`	XML eta web protokoloak erabilita zerbitzu bat edo gehiago agerian uzten ditu.

3.50. taula. J2EE diseinu-patroiak integrazioaren mailarako.

3.3.6.1. Aurkezpen maila: View Helper

Erakunde informatiko batek hiru mailako arkitektura bat eta Javaren teknologia aukeratu baditu aplikazio informatiko bat garatzeko, seguru asko, JSPren baliabideak erabilita eratuko du aurkezpen maila.

Aplikazioa ez bada oso konplexua, JSP orriak berak beteko ditu aurkezpen mailaren kontrolatzaile-lanak, alegia, bezeroak bidaltzen dizkion deiak berak jaso eta, `scriplet` gutxi batzuk erabilita, kontrolatuko ditu deiak. Baina zer gertatzen da aplikazioaren etorkizuneko bertsioek prozesu konplexuagoak eskatzen badizkiote JSP orriari? Bada, JSP orria erabat *zikinduko* dela `scriplet` kode pila bat itsatsita. Horrela, etorkizunean aldaketa bat edo eguneraketa bat gauzatzen bada, erdi erotuta bukatuko dute diseinatzaileek eta programatzaileek, JSP orriaren astuntasunak artegatuta.

Egun, nahikoa zaila denez diseinatzaile edo programatzaile onak aurkitzea, hobe izaten da halako egoeretara heldu baino lehen, irtenbide bat bilatzen hastea gauzak hainbeste ez okertzeko. Bikoteetan gertatzen den bezala, hemen ere, estrategia egokiena ardurak banatzea izaten da. Horretarako, gure hasierako JSP orriari bikotekide lirain, artez eta eraginkor bat bilatuko diogu J2EE diseinu-patroien merkatu (**3.48.**, **3.49.** eta **3.50. taulak**) zabalean.

Bilatzen ari garen bikotekideari ez diogu baldintza askorik eskatuko baina bai bat nahitaezkoa: bere gain hartzea JSP orriak prozesatu behar duen aurkezpen mailako domeinuaren logika. Lan hori egokien egin dezakeen J2EE diseinu-patroia View Helper da.

Aurkezpen mailan, aurkezpen mailaren prozesuaren logika kudeatzeko laguntza eskatzen dio View Helper patroiak bistaren rola bete behar duen osagaiari beste osagai bat eraikiz eta txertatuz.

Txerta dezagun aipatutako osagaia JSP orri batean, adibide txiki bat aztertuta. Adibideak katalogo baten produktuak eskuratu nahi ditu eta JSP orri batean utzi, erabiltzaileak ikus ditzan sailkatuta, produktuaren izenaren arabera. Osagai berri horrek `KatalogoHelper` izena edukiko du eta hura arduratuko da katalogoaren produktuak eskuratzeaz, lana kenduta JSP orriari. Produktuaren zerrenda eskuratzeko, `EmaitzaOrria` klasea erabiliko da eta `KatalogoHelper` klasearen barnean, `eskuratuProduktuak()` metodoak hartuko du `EmaitzaOrria` motako objektua eraikitzeko ardura. Hona hemen, nola gelditzen den `KatalogoHelper` klasearen iturburu-kodea:

```
public class KatalogoHelper implements java.io.Serializable {

    private KatalogoDAO dao;

    private boolean erabiliFastLane = false;
    private String searchQuery = "";
    private String kategoriaId = "";
    private String produktuId = "";
    private String elementuId = "";
    private Tokia tokia;
    private int kontatu = 2;
    private int hasiera = 0;

public KatalogoHelper(boolean erabiliFastLane) {
        this.erabiliFastLane = erabiliFastLane;
    }

    public KatalogoHelper() {
        erabiliFastLane = true;
    }
....
```

```
public EmaitzaOrria eskuratuProduktuak(String kategoriaId, int
hasiera, int kontatu, String tokiaString)
        throws KatalogoSalbuespena {
        return erabiliFastLane
    ? eskuratuProdDAOErabilita(kategoriaId, hasiera, kontatu, tokia)
    : eskuratuProdEJBErabilita(kategoriaId, hasiera,
kontatu,tokia);
    }

    private EmaitzaOrria eskuratuProdEJBErabilita(String kategoriaId,
int hasiera, int kontatu, Tokia tokia)
        throws KatalogoSalbuespena {
        return eskuratuKatalogoEJB().eskuratuProduktuak(kategoriaId,
hasiera, kontatu, tokia);
    }

    private EmaitzaOrria eskuratuProdDAOErabilita(String kategoriaId,
int hasiera, int kontatu, Tokia tokia)
        throws KatalogoSalbuespena {
        try {
            if (dao == null)
                dao = KatalogoDAOFaktoria.eskuratuDAO();
return dao.eskuratuProduktuak(kategoriaId, hasiera, kontatu, tokia);
        } catch (KatalogoDAOSysSalbuespena se) {
            throw new KatalogoSalbuespena(se.getMessage());
        }}
private KatalogoLocal eskuratuKatalogoEJB() throws
KatalogoSalbuespenan {
        try {
            ServiceLocator sl = new ServiceLocator();
            KatalogoLocalHome home
=(KatalogoLocalHome)sl.getLocalHome(JNDIIzenak.KATALOGO_EJBHOME);
            return home.create();

        } catch (javax.ejb.CreateException cx) {
        throw new KatalogoSalbuespenan("KatalogoHelper: errorea
KatalogoLocal EJB erakitzen: " + cx);
        } catch (ServiceLocatorException slx) {
         throw new KatalogoSalbuespenan("KatalogoHelper: errorea
bilatzen  Katalogo Home: " + slx);
        }
```

`eskuratuProduktuak()` metodoak bi estrategia erabil ditzake katalogoaren produktuak eskuratzeko. Batean, produktuak eskuratzeko, EJB bean lokal bat erabiliko da aurretik, `KatalogoLocalHome` motako objektuen faktoria bat eraikita. Beste estrategian, JDBC APIa erabilita, zuzenean eskuratzen ditugu katalogoaren produktuak, SQL aginduak erabilita. Horretarako, objektuen faktoria bat eraikitzen da produktuak eskuratzeko datu-basetik DAO (Data Acces Object)

J2EE diseinu-patroia[43] erabilita. Patroiak, bi klase (`KatalogoDAOFaktoria`, `CloudscapeKatalogoDAO`) eta interfaze bat (`KatalogoDAO`) erabiliko ditu atzipenak gauzatzeko:

```
public class KatalogoDAOFaktoria {

    public static KatalogoDAO eskuratuDAO() throws
KatalogoDAOSysSalbuespena {

        KatalogoDAO kategoriaDAO = null;
        try {
            InitialContext ic = new InitialContext();
            String klaseIzena= (String)
ic.lookup(JNDIzenak.KATALOGO_DAO_KLASEA);
            kategoriaDAO = (KatalogoDAO)
Class.forName(klaseIzena).newInstance();
        } catch (NamingException ne) {
 throw new KatalogoDAOSysSalbuespena("KatalogoDAOFaktoria.eskuratuDAO:
IzenSalbuespena : \n" + ne.getMessage());
        } catch (Exception se) {
            throw new
KatalogoDAOSysSalbuespena("KatalogoDAOFaktoria.eskuratuDAO:
Salbuespena DAO mota eskuratzen : \n" + se.getMessage());
        }
        return kategoriaDAO;
    }}

public interface KatalogoDAO {
……
public EmaitzaOrria eskuratuProduktuak(String kategoriaId, int
hasiera, int kontatu, Tokia l)
        throws KatalogoDAOSysSalbuespena;
……
        }
public class CloudscapeKatalogoDAO implements KatalogoDAO {
….
public static String ESKURATU_PRODUKTUAK_SQL_AGINDUA
  = "select a.produktuId, izena, desk "
  + " from (produktu a join produktu_ezaugarriak b on
a.produktuId=b.produktuId) "
  + " where tokia = ? and a.katid = ? order by izena";
……..
public EmaitzaOrria eskuratuProduktuak(String kategoriaId, int
hasiera, int kontatu, Tokia l)
    throws KatalogoDAOSysSalbuespena {
      Connection c = null;
      PreparedStatement ps = null;
      ResultSet rs = null;
```

43. J2EE diseinu-patroi hori geroxeago aztertuko dugu beste atal batean.

```
      EmaitzaOrria emaitza = null;

   try {
         c = getDataSource().getConnection();
         ps = c.prepareStatement(ESKURATU_PRODUKTUAK_SQL_AGINDUA,
                                 ResultSet.TYPE_SCROLL_INSENSITIVE,
                                 ResultSet.CONCUR_READ_ONLY);
         ps.setString(1, l.toString());
         ps.setString(2, kategoriaId);
         rs = ps.executeQuery();
         if (hasiera >= 0 && rs.absolute(hasiera+1)) {
           boolean hasNext = false;
           List elementuak = new ArrayList();
           do {
             elementuak.add(new Produktua(rs.getString(1).trim(),
                                    rs.getString(2).trim(),
                                    rs.getString(3).trim()));
           } while ((hasNext = rs.next()) && (—kontatu > 0));
           emaitza = new EmaitzaOrria(elementuak, hasiera, hasNext);
         } else {
           emaitza = EmaitzaOrria.HUTSIK_EMAITZAORRIA;
         }
         rs.close();
         ps.close();
         c.close();
         return emaitza;
       } catch (SQLException se) {
 throw new KatalogoDAOSysSalbuespena("SQLSalbuespena: " +
se.getMessage());
       }}
```

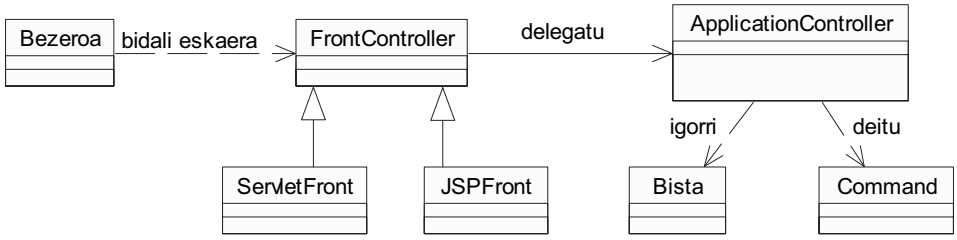
Lehenengo klasean (`KatalogoDAOFaktoria`), `KatalogoDAO` motako objektu bat itzultzen dio `KatalogoHelper` klasearen `eskuratuProdDAOErabilita()` metodoari. Horrek, `eskuratuProduktuak()` metodoa egikaritzeko eskatzen dio `KatalogoDAO` interfazea inplementatzen duen objektuari, eta gauzak ondo badoaz, `EmaitzaOrria` motako objektu bat eraikiko da katalogoaren produktuak jasotzeko.

3.3.6.2. Aurkezpen maila: Front Controller

Aplikazio informatiko bat garatzeko web osagaiak eta EJB osagaiak erabiltzea erabaki badugu, bide luze eta etenaldiz betea izango du urruneko bezeroaren deiak, erantzuna jaso baino lehen. Hasieran, zerbitzarian dagoen web osagaiak ongi etorria emango dio bezeroaren deiari eta EJB osagai egokia aukeratuko du deiak eskatzen duen domeinuaren logika tratatzeko. EJB osagai horrek, beste EJB osagai batzuei deituta edo berak zuzenean, domeinuaren logikak eskatzen duen prozesua eratzen du enpresaren informazio-sistemaren datu-baseak atzituta.

Aipatutako eszenatokia inplementatzeko estrategia bat baino gehiago dago. Guk, hasierako unea landuko dugu. Urruneko bezeroak deia luzatzen dionean zerbitzariari, adarkatze bat eduki dezake dei horren trataerak, prozesua erabat bananduz edo erabat zentralizatuz, egin beharreko ekintza guztiak kontrolpean edukitzeko. Bigarren estrategia hori inplementatzea erabakitzen badugu, Front Controller diseinu-patroiaren arkitekturak laguntza hobezina eskaintzen digu.

Front Controller diseinu-patroiak (**3.51. irudia**) bezeroaren dei guztiak zentralizatu eta bere ardurapean egikarituko ditu. Horretarako, Front Controller rola jokatuko duen osagaiak metodoak izango ditu bezeroaren deiari ongi etorria emateko, baita deiak jarraituko duen egikaritze-protokolo estandarra finkatzeko ere. Protokolo horretan, urrats aurredefinitu batzuk jarraituko ditu deiak eskatzen dituen ekintzak egikaritzeko. Horrela, urruneko bezeroaren dei bakoitza tratatzeko era estandarra izango da eta finkatuta egongo da aurretik.



3.51. irudia. `Front Controller` diseinu-patroiaren arkitektura orokorra.

Ekar dezagun azaldutako guztia iturburu-kodera eta azter ditzagun diseinu-patroiaren trikimailu guztiak:

```
import javax.servlet.*;
import javax.servlet.http.*;

public class FrontController extends HttpServlet
{
private static final String HOME = "index.jsp";
private static final String LOGIN = "servlet/ServletLogin";
// init metodoa birdefinitzen
public void init (ServletConfig config) throws ServletException {
super.init(config);
// DAO faktoria eskuratu eta aplikazioan biltegiratu
try {
DAOFaktoria daoFaktoria =
DAOFaktoria.eskuratuDAOFaktoria(DAOFaktoria.ATZIPENA);
ServletContext aplikazioa = config.getServletContext();
aplikazioa.hasieratuAtributua("DAOFaktoria", daoFaktoria);
} catch (Exception e) {
e.printStackTrace();
}
}
```

```
public void doGet(HttpServletRequest eskaera,
HttpServletResponse erantzuna)
throws ServletException, java.io.IOException {
prozesua(eskaera, erantzuna);
}
public void doPost(HttpServletRequest eskaera,
HttpServletResponse erantzuna)
throws ServletException, java.io.IOException {
prozesua(eskaera, erantzuna);
}
// Prozesua
protected void prozesua (HttpServletRequest eskaera,
HttpServletResponse erantzuna)
throws ServletException, java.io.IOException {
// Helbide fisikoa eskuratu
// EskaeraHelper objektuarentzat
String dirAplikazioa =
getServletConfig().getServletContext().getRealPath("/");
// Helper klasea erabili ekintza gauzatzeko
EskaeraHelper eskaeraHelper =
new EskaeraHelper(eskaera, dirAplikazioa);

if (eskaeraHelper.erroreaNabigatzen()) {
ikuskatu(HOME, eskaera, erantzuna);
return;
}

if (eskaeraHelper.egiaztapenaBehar()) {
// Biltegiratu URI-a
// Login servletari laguntzeko
ikuskatu(LOGIN, eskaera, erantzuna);
return;
}

// Egikaritu behar den ekintza
Ekintza jarduna = eskaeraHelper.eskuratuEkintza();

// Eskuratu aplikazioa objektua ekintza gauzatzeko
ServletContext aplikazioa =
getServletConfig().getServletContext();

// Ekintza gauzatu eta bista eskuratu
String bista = jarduna.egikaritu(eskaera, erantzuna, aplikazioa);
// Emaitza ikuskatu
ikuskatu(bista, eskaera, erantzuna);
}

protected void ikuskatu(String bista, HttpServletRequest eskaera,
HttpServletResponse erantzuna)
throws ServletException, java.io.IOException {
```

```
String bistaTestuinguru = /* eskaera.getContextPath() + "/" + */
bista;
// Eskaera bistara bideratzen da
RequestDispatcher rd =
eskaera.getRequestDispatcher(bistaTestuinguru);
rd.forward(eskaera, erantzuna);  }}
```

3.51. irudian agerian gelditzen den legez, Front Controller diseinu-patroia inplementatzeko bi estrategia erabil daitezke: lehenengoarekin, patroia inplementatzeko, JSP orri soila erabiliko genuke kontrolatzaile-lanak gauzatzeko. Kontrolatzen den lan-fluxua txikia denean, bide hau egokia izan daiteke baina aplikazioa konplexua eta handia bada, bigarren estrategia nagusitzen da zalantzarik gabe. Bigarren estrategiak, servlet bat erabiliko du diseinu-patroia inplementatzeko eta, horixe da guk aukeratu duguna patroiaren nondik norakoak arakatzeko.

Servletak `init()` metodoa birdefinitu egiten du eta, barruan, DAO (Data Access Object) faktoria bat eskuratzen du eta aplikazioan biltegiratzen du. `doGet()` eta `doPost()` metodoek bezeroaren eskaera jaso eta `prozesua()` metodora desbideratzen dute, metodo horrek bezeroaren eskaera prozesa dezan. `prozesua()` metodoak lau urrats generiko egikaritzen ditu beti: (a) eskaera prozesatzeko zein ekintza egikarituko den erabaki, (b) `Ekintza` motako objektu bat instantziatu, (c) ekintza zehatzaren prozesua egikaritzeko eskatu eta, bukatzeko, (d) emaitzak bistaratzeko agindua bidali bistari, gauzatu dezan agindua. Servlet nagusiaren lana gutxitzeko, egikarituko den ekintza erabakitzeko eta `Ekintza` motako objektua instantziatzeko, `EskaeraHelper` motako objektu laguntzailea erabiliko du programak. Hona hemen, `EskaeraHelper` klasearen egitura orokorra iturburu-kodean adierazita:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class EskaeraHelper
{
private static final String EKINTZA_PROPIETATEAK =
"/WEB-INF/Ekintza.properties";
protected HttpServletRequest eskaera;
protected String dirAplikazioa;
public EskaeraHelper(HttpServletRequest eskaera, String dirAplikazioa)
{
this.eskaera = eskaera;
this.dirAplikazioa = dirAplikazioa;
}

public Ekintza eskuratuEkintza() {
// URI-a aztertu ekintza erabakitzeko
String uri = eskaera.getRequestURI();
```

```
int posHasiera = uri.lastIndexOf("/");
int posAmaiera = uri.lastIndexOf(".");
String EkintzaGako = uri.substring(posHasiera + 1, posAmaiera);

Ekintza jarduna = null;
try {
InputStream is = new FileInputStream(dirAplikazioa +
EKINTZA_PROPIETATEAK);
Properties propietate = new Properties();
propietate.load(is);
String strKlaseEkintza = propietate.getProperty(EkintzaGako);
// Ekintza objektua instantziatu

Class klaseEkintza = Class.forName(strKlaseEkintza);
jarduna = (Ekintza) klaseEkintza.newInstance();
} catch (Exception e) {
e.printStackTrace();
}
return jarduna;
}


public boolean egiaztapenaBehar() {

// Metodo honetan egiaztapena behar duten
// baliabideak aztertuko dira
return false;
}
public boolean erroreaNabigatzen() {
// Baliabideen arteko nabigazio erlazioak egiaztatu

return false;
}}
```

`Ekintza` motako objektu bat itzultzen du `eskuratuEkintza()` metodoak propietateen fitxategi bat erabilita. Eskaera-ekintza asoziazioak edukiko ditu gordeta propietateen fitxategi horrek eta hura kontsultatuz erabakiko du programak zein ekintza gauzatuko duen. Gure kasuan, honelako hasierako balioak edukiko ditu:

```
datuak= EkintzaDatuak
erabiltzaileak= EkintzaErabiltzaileak
........
```

`Ekintza` motako objektu bakoitzak bere `egikaritu()` metodoa izango duenez, `Ekintza` interfazea honela programatuko da:

```
import javax.servlet.*;
import javax.servlet.http.*;
public interface Ekintza
```

```
{
public String egikaritu(HttpServletRequest eskaera,
HttpServletResponse erantzuna, ServletContext aplikazioa);
}
```

Servletaren eskaera eta erantzuna jasotzen ditu `egikaritu()` metodoak parametro gisa eta ekintzaren emaitza bistaratzeko erabiliko den baliabidearen (gehienetan JSP orria) informazioa itzultzen du `String` motako objektu batean. Hona hemen, `EkintzaDatuak` klaseak nola inplementatuko lukeen `Ekintza` interfazea:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class EkintzaDatuak implements Ekintza
{
public String egikaritu(HttpServletRequest eskaera,
HttpServletResponse erantzuna, ServletContext aplikazioa)
{
// Datuak eskuratzen

Bezeroa bezero = new Bezeroa();
bezero.hasieraIzena(eskaera.getParameter("izena"));
bezero. hasieraNan(eskaera.getParameter("nan"));
bezero. hasieraHelbidea(eskaera.getParameter("helbidea"));
bezero. hasieraErabiltzailea(eskaera.getParameter("erabiltzailea"));
bezero. hasieraGakoa(eskaera.getParameter("gakoa"));

// DAO Bezeroa eskuratu
DAOFaktoria DAOFaktoria = (DAOFaktoria)
aplikazioa.eskuratuAtributua("DAOFaktoria");
BezeroaDAO bezeroaDAO = DAOFaktoria.eskuratuBezeroaDAO();
//  Bezeroaren datuak eguneratu
try {
bezeroaDAO.eguneratu(bezero);
} catch (Exception e) {  e.printStackTrace();}


// Bezeroa objektua eman JSP orriari

eskaera.setAttribute("Bezeroa", bezeroa);

// Eguneratu sesioa
HttpSession sesioa = eskaera.getSession();
sesioa.setAtribute("Bezeroa", bezeroa);
return "datueguneratuak.jsp";  }}
```

Ekintza zehatz honetan, bezeroaren datu pertsonalak formulario batetik eskuratzen ditu `egikaritu()` metodoak eta `Bezeroa` motako objektu bat eraiki eta hasieratzen du. Bigarren urratsean, DAO faktoria erabilita, bezeroaren DAOa

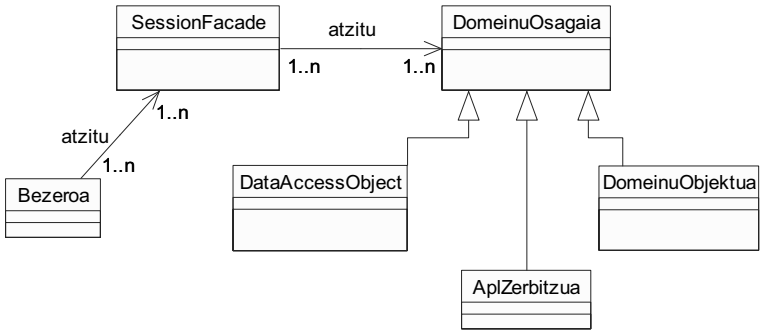
eskuratzen du eta bezeroaren datuak eguneratzen ditu. Bukatzeko, `Bezeroa` motako objektu eraiki berria JSP orriari pasatzen dio eta sesioa eguneratzen du. Dena ondo joan dela adierazteko, emaitzak bistaratzeko erabiliko den bistaren informazioa (`datueguneratuak.jsp`) itzultzen du metodoak.

```
<html>
<head>
<meta http-equiv="Content-Language" content="es">
<title>Datu Eguneratuak</title>
</head>
<body>
<jsp:useBean id="bezeroa" class="Bezeroa" scope="request" />
<p><font size="4">
Zure datuak ondo eguneratu dira:</font></p>
<p><b>Izena:</b><jsp:getProperty name="Bezeroa" property="izena" />
</p>
<p><b>NAN:</b> <jsp:getProperty name="Bezeroa" property="nan" /> </p>
<p><b>Helbidea:</b><jsp:getProperty name="Bezeroa"
property="helbidea" />
</p>
</body>
</html>
```

JSP orriak mezu bat aterako du adieraziz datuak ondo eguneratu direla eta zein den eguneratu den bezeroaren informazio pertsonala.

3.3.6.3. Domeinu eta integrazio mailak: Session Façade

Atal honetan aztertuko dugun hirugarren J2EE diseinu-patroia arras ezaguna zaigu, diseinu-patroi nagusien atalean aurkeztu baikenuen lehenengo liburuan honen gurasoa izan zitekeena. Hemen diseinu-patroiari Session Façade (**3.52. irudia**) deituko diogu eta, lehen bezala, bitartekari-lana beteko du domeinuaren osagaien eta bezeroaren artean. Arkitektura honetan, bezeroak ez ditu zuzenean bidaliko urruneko deiak EJB osagaietara, tartean, egoeradun sesio-bean bat kokatuko baitu dei horiek jasotzeko eta kudeatzeko. Horrela, osagaien artean gertatzen diren ekintza guztiak ezkutatuta gelditzen zaizkie bezeroei. Lan-fluxuaren kudeaketa, Session Façade patroiaren rola beteko duen osagaiaren esku gelditzen da baita osagaien bizi-zikloen nondik norakoak ere, osagaiak eraikitzeko, bilatzeko, eguneratzeko eta ezabatzeko. Interfazearen metodoen signatura kontrolatu eta ezagutu behar du bezeroak zerbitzuak eskatzeko. Zerbitzu horiek nola dauden egituratuta eta programatuta, Session Façade sesio-beanaren esku gelditzen da.



3.52. irudia. Session Façade diseinu-patroiaren arkitektura orokorra.

Orain arte azaldutakoa apur bat argitzeko, Sun erakundearen Pet Store[44] aplikazio paradigmatikoaren zati txiki bat aztertuko dugu. Gaia kokatzeko esan dezagun, aplikazioak etxe-abereen salmenta elektronikoa kudeatzen duela EJB osagaiak erabilita eta Eredu-Bista-Kontrol arkitektura inplementatuta. Guk bereziki, bezeroaren eskaera prozesatzeko gunean dagoen funtzionalitate zehatz bat aztertuko dugu. Bezeroak eskaerak amaitu dituenean, finantza-sailak eskaerak onartzeko, aztertu egin nahi ditu, jakiteko zein dagoen hari interesatzen zaion egoera zehatz batean. Horretarako, aplikazioaren bezeroari interfaze bat diseinatu zaio `EJBObject` interfazea hedatuko duena. Interfaze horrek, beanak ikusgai ipiniko dituen metodoen signatura errepikatzen du:

```
public interface OPCAdminFacade extends EJBObject {

    public OrdersTO getOrdersByStatus(String status)
        throws RemoteException, OPCAdminFacadeException;

    public Map getChartInfo(String request,
                            Date start,
                            Date end,
                            String requestedCategory)
    throws RemoteException, OPCAdminFacadeException; }
```

Konturatu ezen aplikazioaren bezeroaren leihoa, eskaeren egoera aztertzeko, `OPCAdminFacade` dela bakarrik. Bezeroaren deiak, nahi duen emaitza lortzeko, bakarrik jakin behar du eskaeren egoera bat aztertzeko nahikoa duela `getOrdersByStatus()` metodoa egikaritzea. Deiaren konplexutasuna egikaritzeko orduan, Session Façade rola beteko duen osagaiari pasatuko zaio, eta hura izango da bilaketak eta bilaketa horien emaitzen trataera kudeatuko dituena. Pet Store aplikazioan, `OPCAdminFacadeEJB` sesio-beanak beteko ditu aipatutako eragiketa horiek guztiak:

44. `http://java.sun.com/developer/releases/petstore/`
`http://java.sun.com/blueprints/code/`

```
public class OPCAdminFacadeEJB implements SessionBean {
……
public void ejbCreate() throws CreateException {
      try {
         ServiceLocator serviceLocator = new ServiceLocator();
         poLocalHome = (PurchaseOrderLocalHome)
serviceLocator.getLocalHome(PURCHASE_ORDER_EJB);
         ProcessManagerLocalHome processManagerLocalHome =
(ProcessManagerLocalHome)serviceLocator.getLocalHome(PROCMGR_ORDER_EJB
);
         processManagerLocal = processManagerLocalHome.create();
} catch(ServiceLocatorException se) {
          throw new EJBException(se);
      } catch (CreateException ce) {
          throw new EJBException(ce);
      }  }
     /**
  * Business method in the remote interface; Returns a collection of
  * of orders that are in the give status.
  * @param status   Represents the status of orders that are requested
  * @return <code>OrdersTO</code>
  * @throws <code>OPCAdminFacadeException</code> for all errors
     */

    public OrdersTO getOrdersByStatus(String status)
                                                 throws
OPCAdminFacadeException {
        OrdersTO.MutableOrdersTO retVal = new
OrdersTO.MutableOrdersTO();
        PurchaseOrderLocal po;
        ProcessManagerLocal mgr = getProcMgr();

        try {
            PurchaseOrderLocalHome pohome = getPO();
            Collection orders = mgr.getOrdersByStatus(status);
            Iterator it = orders.iterator();
            while((it!= null) && (it.hasNext())) {
                ManagerLocal mgrEjb = (ManagerLocal) it.next();
                po = pohome.findByPrimaryKey(mgrEjb.getOrderId());
                Date gotDate = new Date(po.getPoDate());
                String podate = (gotDate.getMonth()+1) + "/" +
                    gotDate.getDate() + "/" +
(gotDate.getYear()+1900);
    retVal.add(new OrderDetails(po.getPoId(), po.getPoUserId(),
    podate, po.getPoValue(), status));
            }
        } catch(FinderException fe) {
       System.err.println("finder Ex while getOrdByStat :" +
fe.getMessage());
                throw new OPCAdminFacadeException("Unable to find
```

```
PurchaseOrders"+  " of given status : " +  fe.getMessage());
        }
        return(retVal);   }
                                    oOo
import java.io.Serializable;
import java.util.Iterator;
import java.util.Collection;
import java.util.ArrayList;

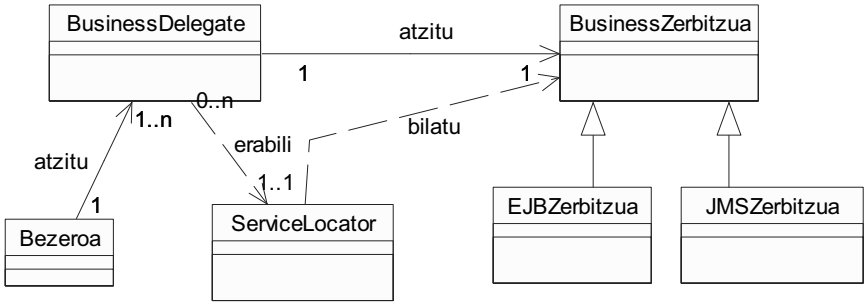
public interface OrdersTO extends Serializable {

    public Iterator iterator();
    public int size();
    public boolean contains(Object o);
    public boolean containsAll(Collection c);
    public boolean equals(Object o);
    public int hashCode();
    public boolean isEmpty();
    public Object[] toArray();
    public Object[] toArray(Object[] a);
    static class MutableOrdersTO extends ArrayList implements
OrdersTO {   }
```

Bezeroaren eskaerak prozesatzeko, aplikazioaren logika guztia ezkutatuta gelditzen zaionez aplikazioaren bezeroari, `OPCAdminFacadeEJB` klasearen `getOrdersByStatus()` metodoak eraiki beharko du itzuliko den objektu bilduma. Metodoak `ProcessManagerLocal` egoera gabeko sesio-bean bat eta `ManagerLocal` entitate-beanak erabiltzen ditu egoera jakin batean dauden eskaera zehatzak (`OrderDetails`) eskuratzeko. Bukatzeko eta salbuespenik altxatzen ez bada, eskuratutako bilduma itzultzen du metodoak.

3.3.6.4. Domeinu eta integrazio mailak: Business Delegate

Demagun aurretik eraikitako aplikazio bat hedatu nahi dugula EJB osagaiak erabilita. Tamalez, lehendik garatutako software gehiena ez zen EJB osagaiak erabilita eraiki, klase arruntak programatuta baizik. Gainera, programatzaileen artean batzuk ez dira oso trebeak EJB osagaiak programatzen. Egoera zertxobait samurtzeko, programatzaile adituek azpiegitura berezi bat prestatuko dute arazoak ekiditeko software berria eraikitzean. Bilera bat egin dute eta erabaki dute irtenbide estandar eta fidagarri bat aplikatuko dutela partaide guztiak bitsetan eta pozarren gelditzeko. Business Delegate (**3.53. irudia**) J2EE diseinu-patroia erabilita uste dute egoera bere onera etorriko dela eta erabakitzen dute diseinu-patroi hori inplementatzea bezeroaren eskaeraren trataeran. Horretarako, `EskaeraDelegate` klasea diseinatu dute domeinuaren zerbitzu nagusiak inplementatzeko, hala nola `Home` objektu-faktorien bilaketa eta eraikuntza. Domeinuaren osagaietara egingo diren deien kudeaketa, Session Façade diseinu-patroi ezaguna inplementatzen duen sesio-beanak kontrolatuko du.



3.53. irudia. Business Delegate diseinu-patroiaren arkitektura orokorra.

Aipatutako trikimailu horiek guztiak ezkutatuta gelditzen zaizkio `EskaeraDelegate` motako objektu baten baliabideak erabiliko dituen bezeroari. Bezero horrek, domeinuaren metodoak erabiltzen ditu zuzenean deiak egikaritzeko baina `EskaeraDelegate` klaseak, proxy-lana gauzatuta, eskaera horiek sesio-beanera bideratzen ditu Session Façade rola betetzen duen osagaiak kudea dezan bezeroaren deia. Hona hemen nola gauzatzen den prozesu guztia iturburu-kodean:

```
public class EskaeraDelegate {

  private EskaeraSesioa sesioa;

  private static final Class homeKlasea =
  EskaeraSesioaHome.class;

  public EskaeraDelegate() throws EskaeraSalbuespena {
    try {
      EskaeraSesioaHome home = (EskaeraSesioaHome)
        ServiceLocator.getInstance().getHome(
          "Eskaera", homeKlasea);
      sesioa = home.create();
    } catch(ServiceLocatorException ex) {
      // Service Locator salbuespena
      // aplikazio salbuespena bihurtu
      throw new EskaeraSalbuespena(...);
    } catch(CreateException ex) {
      // Sesio salbuespena
      // aplikazio salbuespena bihurtu
      throw new EskaeraSalbuespena(...);
    } catch(RemoteException ex) {
      // Remote salbuespena
      // aplikazio salbuespena bihurtu
      throw new EskaeraSalbuespena(...);  }}


  public EskaeraDelegate(String id)
    throws EskaeraSalbuespena {
    super();
```

```
    konexioaEskuratu(id);
  }

  public String eskuratuIdentifikatzailea() {
    try {
      return ServiceLocator.getId(sesioa);
    } catch (Exception e) {
      }}

    public void konexioaEskuratu(String id)
    throws EskaeraSalbuespena {
    try {
      sesioa = (EskaeraSesioa)
                ServiceLocator.getService(id);
    } catch (RemoteException ex) {
      // Remote salbuespena
      // aplikazio salbuespena bihurtu
      throw new EskaeraSalbuespena(...);   }}

  // Domeinuaren metodoak
  // Session Facadera bideratuak proxy eran

 public Eskaera finkatuOraingoEskaera(String eskaeraId)
    throws EskaeraSalbuespena {
    try {
      return sesioa.finkatuOraingoEskaera(eskaeraId);
    } catch (RemoteException ex) {
      // Zerbitzu salbuespena
      // aplikazio salbuespena bihurtu
      throw new EskaeraSalbuespena(...);  }}

  public Eskaera eskuratuEskaeraZehatza()
    throws EskaeraSalbuespena {

    try {
      return sesioa.eskuratuEskaeraZehatza();
    } catch(RemoteException ex) {
      // Zerbitzu salbuespena
      // aplikazio salbuespena bihurtu
      throw new EskaeraSalbuespena(...);  }}

  public void finkatuEskaeraZehatza(Eskaera esk)
    throws EskaeraSalbuespena {
    try {
      sesioa.finkatuEskaeraZehatza(esk);
    } catch(RemoteException ex) {
      throw new EskaeraSalbuespena(...);  }}

  public void gehituEskaeraBerria(Eskaera esk)
    throws EskaeraSalbuespena {
```

```
    try {
      sesioa.gehituEskaera(esk);
    } catch(RemoteException ex) {
      throw new EskaeraSalbuespena(...);  }}

  // beste proxy metodoak
  ...
}

public interface EskaeraSesioa extends EJBObject {

  public Eskaera finkatuOraingoEskaera(
    String eskaeraId) throws
    RemoteException, EskaeraSalbuespena;

  public Eskaera eskuratuEskaeraZehatza()
     throws RemoteException, EskaeraSalbuespena;

  public void finkatuEskaeraZehatza(Eskaera resource)
     throws RemoteException, EskaeraSalbuespena;

  public void gehituEskaera(Eskaera resource)
      throws RemoteException, EskaeraSalbuespena;

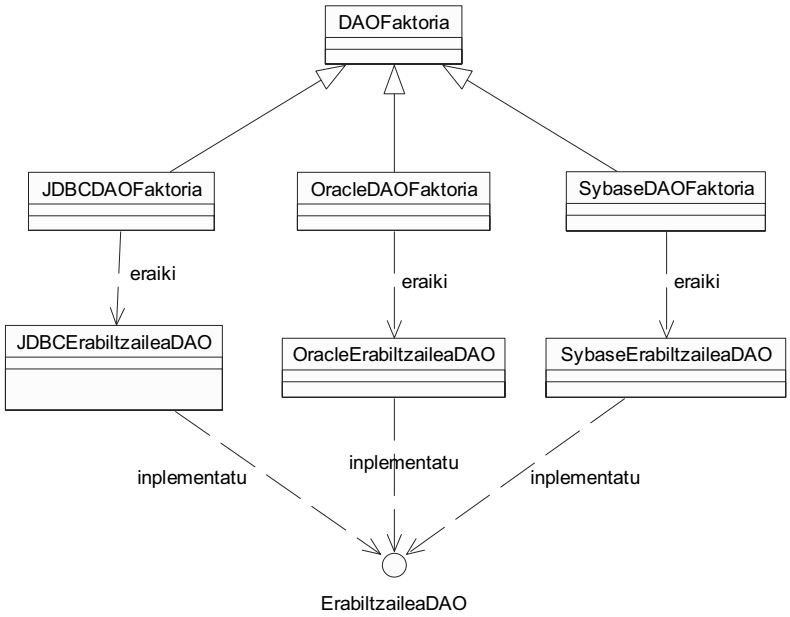
  ...
}
```

`EskaeraDelegate` klasearen metodo eraikitzaileek domeinuaren zerbitzuak antolatzen dituzte `Home` klaseak eskuratuta eta `EJBObject` motako objektuak eraikita. Hain zuzen ere, azken osagai hori izango da Session Façade rola jokatuko duena aplikazioan.

Konturatu ezen `Eskaera` motako objektua esku artean darabilten metodoek sesio-beanari *delegatzen* diotela deiaren egikaritze-lan guztia, beraiek, proxy-lana betetzen dute soilik.

3.3.6.5. Domeinu eta integrazio mailak: DAO (Data Access Object)

Atal honetan aztertuko dugun azken J2EE diseinu-patroia behin eta berriz agertu da aurreko diseinu-patroietan informazioa biltegiratzen zenean. Informazioa biltegiratzeko erabiltzen diren iturriak kudeatzen ditu Data Access Object (DAO) J2EE diseinu-patroiak (**3.54. irudia**). Diseinu-patroi horrek datuetara atzitzeko behar duen logika guztia kapsulatu egiten du eta domeinuaren logikak ez du zertan arduratu informazioa nondik edo nola eskuratzen den. DAO diseinu-patroia erabilita interfazeak izango ditu aplikazioaren bezeroak, datuetara atzitzeko zuzenean.



3.54. irudia. DAO diseinu-patroiaren arkitektura orokorra.

Era asko daude DAO diseinu-patroia inplementatzeko, guk erabiliko duguna **3.54. irudia**n agertzen den Klase Diagramarena izango da. Hasieran, DAO faktoria bat eraikiko dugu DAO mota guztientzat. Faktoria hori eraikitzeko `DAOFaktoria` klase abstraktua diseinatuko dugu eta bere lan bakarra `xxxDAOFaktoria` motako objektuak eraikitzea izango da. Hona hemen `DAOFaktoria` klasearen iturburu-kodea:

```
public abstract class DAOFaktoria {

  // Faktoriak kudeatzen dituen DAO mota zehatzak
  public static final int JDBC = 1;
  public static final int ORACLE = 2;
  public static final int SYBASE = 3;

  //Metodo bat DAO bakoitzarentzat
  //Faktoria zehatz bakoitzak inplementatuko du metodoa

  public abstract ErabiltzaileaDAO getErabiltzaileaDAO();
  ............

  public static DAOFaktoria getDAOFaktoria(int faktoria) {

    switch (faktoria) {
      case JDBC:
          return new JDBCDAOFaktoria();
```

```
      case ORACLE:
          return new ORACLEDAOFaktoria();
      case SYBASE:
          return new SYBASEDAOFaktoria();
      default:
          return null;  }}}
```

`DAOFaktoria` klasean, metodo bakarra definitzen dugu: `getDAOFaktoria()` metodoa. Unean uneko `DAOFaktoria` zehatza eraikitzeko ardura hartzen du metodo estatiko horrek. Klaseak, ordea, baditu beste metodo batzuk erazagutu egiten direnak `abstract` ezaugarria itsatsita, baina klasean definituta ez daudenak. DAO faktoria zehatzak inplementazio egokia esleituko dio metodo bakoitzari, `DAOFaktoria` klasea hedatuta.

Azter ditzagun azaldutako arkitekturaren xehetasunak adibide txiki batekin. Demagun datu-base batera atzitu nahi dugula, `jdbc:odbc` zubia erabilita, erabiltzailearen informazio pertsonala txertatzeko. Lehendabizi, txertaketa gauzatu baino lehen, aplikazioak ziurtatu egingo du erabiltzailea ez dagoela lehenagotik txertatuta datu-basean. Eragiketa hori egiteko, bi klase nagusi diseinatu ditugu DAO diseinu-patroia erabilita, ebazpena generiko eta berrerabilgarri bihurtuz. Lehenengo klaseak (`JDBCDAOFaktoria`) `DAOFaktoria` klase abstraktua hedatuko du inplementazioa itsatsiz `DAOFaktoria` klasearen metodo abstraktu guztiei. Guk, adibidean, `getErabiltzaileaDAO()` metodo abstraktua hartuko dugu bakarrik kontuan, hemendik aurrera. Metodo horrek, `ErabiltzaileaDAO` motako objektu bat itzultzen du aipatutako bigarren klase nagusia (`JDBCErabiltzaileaDAO`) instantziatuta. Hona hemen, bi klase nagusi horien eta `ErabiltzaileaDAO` interfazearen iturburu-kodea:

```
import java.sql.*;

public class JDBCDAOFaktoria extends DAOFaktoria {
  public static boolean hasieratuta=false;
  public static PoolingDataSource dataSource;

  private static void hasieraEgiten() throws OrokorraException{
  try{
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     ObjectPool connectionPool = new GenericObjectPool(null);
                          ConnectionFaktoria connectionFaktoria = new
DriverManagerConnectionFaktoria("jdbc:odbc:NireDB",null);
PoolableConnectionFaktoria poolableConnectionFaktoria = new
PoolableConnectionFaktoria(connectionFaktoria,connectionPool,null,null
,false,true);
     dataSource = new PoolingDataSource(connectionPool);
  }catch(Exception e){
     throw OrokorraException.orokorraError(e);
  }
  }
```

```
public static Connection eraikiKonexioa() throws OrokorraException{
   try{
     if (!hasieratuta){
       hasieratuta=true;
       JDBCDAOFaktoria.hasieraEgiten();
     }
     return (dataSource.getConnection());
   }catch(SQLException e){
     throw OrokorraException.orokorraError(e);
   }
   }

   public ErabiltzaileaDAO getErabiltzaileaDAO() {
     return new JDBCErabiltzaileaDAO();
   }
.......
}}
                                    oOo
import java.sql.*;

public class JDBCErabiltzaileaDAO implements ErabiltzaileaDAO {

   public JDBCErabiltzaileaDAO() {
   }

   public void txertatuErabiltzailea(ErabiltzaileVO erabil1)throws
InsertarException, OrokorraException {
   Connection conn = null;
   Statement stmt = null;
   ResultSet rset = null;

   try{
if (egonBaiErabiltzailea(erabil1.getErabiltzailea())==false){
       conn = JDBCDAOFaktoria.eraikiKonexioa();
       stmt = conn.createStatement();
       stmt.executeUpdate("INSERT into Erabiltzaileak values("
       + "'" + erabil1.getIzena() + "',"
       + "'" + erabil1.getNan() + "',"
       + "'" + erabil1.getHelbidea() + "',"
       + "'" + erabil1.getErabiltzailea() + "',"
       + "'" + erabil1.getGakoa() + "')");
     }else
       throw new InsertarException();
   } catch(SQLException e) {
     e.printStackTrace();
     throw new InsertarException();
   } finally {
     try { rset.close(); } catch(Exception e) { }
     try { stmt.close(); } catch(Exception e) { }
     try { conn.close(); } catch(Exception e) { }
```

```
  }
}
public boolean egonBaiErabiltzailea(String erabiltzailea) throws
OrokorraException{
  Connection conn = null;
  Statement stmt = null;
  ResultSet rset = null;

  try {
    conn = JDBCDAOFaktoria.eraikiKonexioa();
    stmt = conn.createStatement();
    rset = stmt.executeQuery("SELECT * FROM Erabiltzaileak WHERE
(erabiltzailea="+ " '"+ erabiltzailea +"' )");
    if (rset.next())
      return true;
    } catch(SQLException e) {
      e.printStackTrace();
  throw OrokorraException.orokorraError(e);
    }
    return false;
  }
.............
}
                                    oOo

public interface ErabiltzaileaDAO {
   public void txertatuErabiltzailea(ErabiltzaileVO erabil1) throws
InsertarException,OrokorraException ;

   public ErabiltzaileVO egiaztatu(String erabilIzena, String
pasahitza)throws LoginErabiltzaileEzException,OrokorraException ;

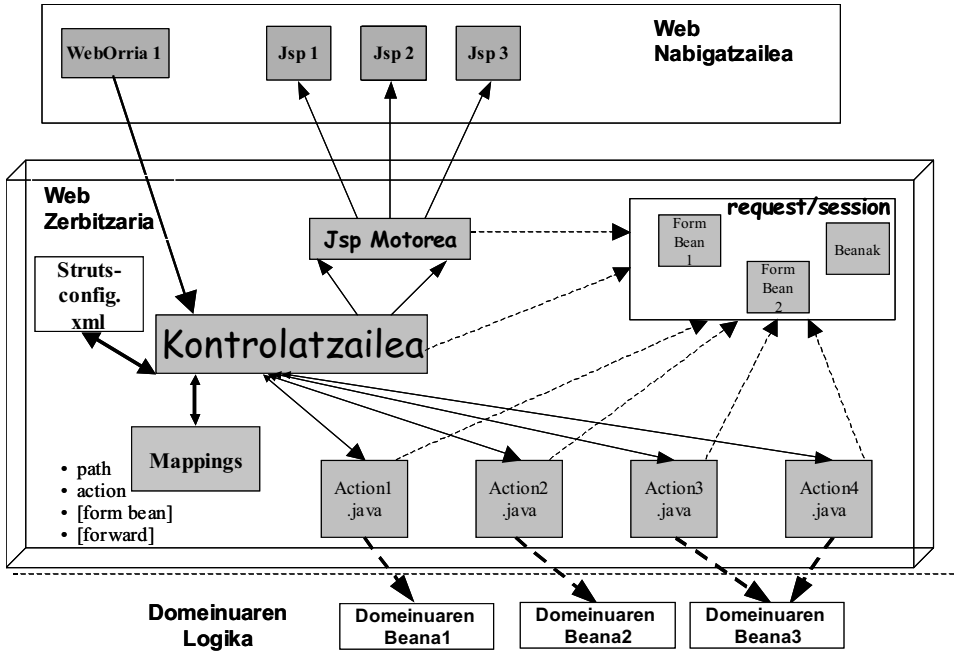
  .........
}
```

Konturatu ezen `JDBCErabiltzaileaDAO` klaseak inplementatu egiten duela `ErabiltzaileaDAO` interfazea. Horrela, edozein objektuk `getErabiltzaileaDAO()` metodoa egikaritzen duenean `ErabiltzaileaDAO` motako objektu bat eskuratuko du, eta gauza izango da `txertatuErabiltzailea()` metodoa eta interfazearen beste metodo guztiak egikaritzeko inplementazio egokia ematen baitie `JDBCErabiltzaileaDAO` klaseak interfazea inplementatzean.

3.3.7. J2EE diseinu-patroien ama: Eredu-Bista-Kontrol eta Struts

Aurreko atal batean aztertu dugunez, aplikazio informatiko baten arkitekturak web eta EJB osagaiak (**3.47. irudia**) dituenean, komeni izaten da aplikazioa zatitzea bere beharrizan funtzionalen arabera. Zatiketa horrek erabat modularizatzen du aplikazioa eta arduren banaketa kapsulatuta gelditzen da ikuspegi

funtzionalaren arabera. Hortaz, urruneko bezeroak eskaera bat luzatzen dionean zerbitzariari, osagai kontrolatzaile bat izango dugu zerbitzarian eskaera jasotzeko eta kudeatzeko. Egin kontu, kudeatu esaten dugula eta ez egikaritu. Eskaeraren domeinuaren logika egikaritzeko, beste osagai laguntzaile batzuk izango ditu kontrolatzaileak, bezeroaren eskaera egikaritzeko. Eskaera egikaritu ondoren, eta agian prozesuaren hasieran ere, eskaeraren emaitzak jaso eta JSP orri bat erabilita gehienetan, urruneko bezeroari bidaltzen dizkio kontrolatzaileak, deiaren bizi-zikloa amaituz.



3.55. irudia. Struts baliabidearen arkitektura orokorra.

Argi dago bezeroaren deiaren prozesua kudeatzeko hiru baliabide funtzional behar ditugula: kontrolatzailea, domeinuaren logika interpretatuko duen osagaia eta emaitza ikusgai utziko duen osagaia. Eta hara non aurkitu dugun bilatzen ari ginen hirukoa: Eredu-Bista-Kontrol edo Model-View-Control J2EE diseinu-patroi ezaguna.

Gurpila asmatzen hasteak asmatuta dagoenean ez du inolako zentzurik informatikan, ez duzulako hobetuko gurpilaren diseinua eta are gutxiago haren inplementazioa. Hori jakinda, edonoren eskura dagoen[45] tresnaz baliatuko gara atal honen xehetasunak azaltzeko: Struts.

45. http://struts.apache.org/

Struts tresna aurreratuak azpiegitura teknologiko guztia prestatzen dio programatzaileari aipatutako Eredu-Bista-Kontrol arkitektura (**3.55. irudia**) txertatzeko garatzen ari den aplikazio informatikoan. XML fitxategi (`struts-config.xml`) bat erabilita adierazten dio programatzaileak prozesu-fluxua Struts-i. Gauzatu beharreko ekintzak nola dauden egituratuta jasotzen du fitxategi horrek:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config SYSTEM "file:///c:/Struts/dtds/struts-
config_1_1.dtd">

<struts-config>

<!- ============ Form Bean hasieraketak =============== ->

<form-beans>

       <form-bean name="loginForm" type="LoginForm"/>
       <form-bean name=". . ." type=". . . "/>
       . . .

</form-beans>

<!- ============ Global Forward hasieraketak ========= ->

<global-forwards>
      <forward name="orokorra" path="/ErrorOrokorra.jsp"/>
      <forward name="arrakasta" path="/Menu.jsp"/>
      <forward name="login" path="/Login.jsp"/>
      <forward name=". . ." path="/xxxx.jsp"/>
      . . .

</global-forwards>

<!- ============ Action Mapping hasieraketak ============= ->

<action-mappings>


       <action path="/Login" type="LoginAction" name="loginForm"
scope="request" input="/Login.jsp" validate="true">
            <forward name="erabiltzaileEz" path="/LoginEz.jsp"/>
      </action>

........
</action-mappings>

</struts-config>
```

Aplikazioaren arlo zehatz bat kontrolatzen du fitxategiaren etiketa bakoitzak eta bertan agertzen den informazio garrantzitsuena `<action-mappings>` etiketaren azpian egongo da. Etiketa horren azpian agertzen den `<action>` etiketa bakoitzak adierazten du zein ezaugarri dituen aplikazioak egikaritu behar duen `Action` motako ekintza bakoitzak. Besteak beste, `path` atributuarekin, ekintzari deituko dion URLa zehazten da, eta, `type` atributuarekin, `Action` motako klasearen izena zein izango den.

Egikaritu behar duen ekintza formulario batekin erlazionatuta badago, `ActionForm` motako klase bat eratuko da programazio mailan eta sarrera egokia idatziko da XML fitxategian, erlazioa adierazteko. Dagokion `<action>` etiketari honako atributu hauek erantsi behar zaizkio:

- `name`. Atributu honetan, `ActionForm` motako klasearen izena zein izango den zehazten da.
- `scope`. Atributu honek adierazten du noraino (`request/session`) hedatzen den `ActionForm`-aren esparrua.
- `input`. Formularioaren bide erlatiboa espezifikatzen du atributu honek.
- `validate`. Kontrolatzaile nagusiak `ActionForm` klasearen `validate()` metodoari deitzea nahi badugu, atributu honek `true` balioa eduki behar du.

`Action` motako objektuak ekintza egikaritu ondoren, posible da adieraztea zein JSP orriri pasatu nahi zaion kontrola XML fitxategian. Hori egiteko, `<forward>` etiketa berezia erabiltzen da.

XML fitxategia konfiguratuta, `Action` eta `ActionForm` motako klaseak programatu behar ditu programatzaileak Struts tresnak bere lana txukun gauzatzeko.

Eredu-Bista-Kontrol diseinu-patroia nola egituratzen eta egikaritzen den aztertzeko, azalduko dugu Struts-ek nola jasotzen eta nola egikaritzen duen urruneko bezeroaren eskaera (**3.55. irudia**). Urruneko bezeroaren HTTP eskaera Struts instalatuta dugun zerbitzarira heltzen denean, lehenengo ongi etorria, kontrolatzaile nagusiak emango dio. Kontrolatzaile nagusi hori `ActionServlet` motako servlet espezializatu bat izango da eta diseinu-patroiaren *Kontrol* rola beteko du arkitektura orokorrean. Bezeroaren eskaera interpretatuko du kontrolatzaileak XML fitxategia erabilita, eta `Action` motako ekintza egokia aukeratuko du eskaerari erantzuna emateko. Sarrerako formularioetan idatzitako datuak `Action` osagaiei pasatzeko, `ActionForm` motako osagaiak erabiltzen dira. Sarrerako formularioaren atributuak izango ditu `ActionForm` motako osagaiak eta atzipen metodoak (`get()` eta `set()`), atributu horien informazioa atzitzeko.

`Action` motako objektuak domeinuaren osagaiak erabiliko ditu bezeroaren eskaera prozesatzeko. Struts erabat agnostikoa da maila honetan eta programatzaileari uzten dio programatzen berak nahi duen eran *Eredu* rola beteko duen maila hau. Klase arruntak edo EJB osagaiak erabil ditzake programatzaileak maila hori programatzeko.

Eskaeraren trataera bukatzen duenean, `ActionForward` motako izen logiko bat itzultzen dio `Action` objektuak kontrolatzaileari. Kontrolatzailea, XML fitxategira joaten da eta izen logikoari dagokion JSP orria bilatzen du. Bukatzeko, JSP orriari deitzen dio HTTP erantzun egokia eraikitzeko. Konturatu Struts-en erabat banatuta daudela Kontrol eta Bista funtzioak.

Zati hau amaitzeko, adibide txiki bat garatuko dugu `Action` eta `ActionForm` motako klaseen ezaugarri zehatzak azaleratzeko. Aplikazioaren erabiltzailearen izena eta pasahitzak egiaztatuko ditu adibideak eta JSP orriak erabilita erantzunak bistaratuko dira. Honako `Action` eta `ActionForm` klaseak beharko ditu adibideak, Struts tresnak ondo gauzatzeko bere lana:

```
  public final class LoginAction extends Action {
  public ActionForward perform(ActionMapping mapping, ActionForm
form, HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException{
    MessageResources messages= getResources();
    HttpSession session = request.getSession();
    try {
      LoginForm loginForm = (LoginForm) form;
      String erabilIzena = loginForm.getErabilIzena();
      String pasahitza = loginForm.getPasahitza();
      IZerbitzuak zerbitzuaInpl = (IZerbitzuak)
servlet.getServletContext().getAttribute("zerbitzuak");
      ErabiltzaileVO erabil1 = zerbitzuaInpl.egiaztatu(erabilIzena,
pasahitza);
      if (erabil1==null){
        return mapping.findForward("erabiltzaileEz");
      }
      if (servlet.getDebug() >=1)
        servlet.log("LoginAction: Erabiltzailea" + erabil1.getIzena()
+ "'ondo identifikatuta " + session.getId());
      session.setAttribute("erabiltzailea", erabil1);

      if (mapping.getAttribute() != null){
        if ("request".equals(mapping.getScope()))
          request.removeAttribute(mapping.getAttribute());
        else
        request.removeAttribute(mapping.getAttribute());
      }
return mapping.findForward("arrakasta");
    }catch (LoginErabiltzaileEzException e){
```

```
       return mapping.findForward("erabiltzaileEz");
     }catch (OrokorraException e){
       MezuaVO mezua=new MezuaVO(e.getRootCause().toString());
       session.setAttribute("mezua", mezua);
       return mapping.findForward("orokorra");
     }catch (Exception e) {
ServletContext servletContext =
servlet.getServletConfig().getServletContext();
       servletContext.log(e.getMessage(), e);
       return mapping.findForward("index");  }}}
                                oOo

public class LoginForm extends ActionForm {

private String erabilIzena;
private String pasahitza;

public LoginForm() {
  reset();
}

public String getErabilIzena() {
  return erabilIzena;
}
public void setErabilIzena(String erabilIzena) {
  this.erabilIzena = erabilIzena.trim();  }

public String getPasahitza() {
  return pasahitza;  }

public void setPasahitza(String pasahitza) {
  this.pasahitza = pasahitza;  }

public void reset(ActionMapping mapping, HttpServletRequest
request) {
  reset();   }

public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  PropertyValidator.validateMandatory(errors,
"erabilIzena",erabilIzena);
  PropertyValidator.validateMandatory(errors, "pasahitza",
pasahitza);
  return errors;  }

private void reset() {
  erabilIzena = null;
  pasahitza = null;}}
```

Konturatu `Action` klaseak nahikoa duela `perform()` metodoa programatuta izatea, metodo hori izango baita kontrolatzaileak deituko duena ekintza egikaritzeko. Gure kasuan, erabiltzailearen izena eta pasahitza eskuratzen ditu metodoak, `ActionForm` motako `LoginForm` objektua erabilita.

Erabiltzailearen egiaztapena, `IZerbitzuak` motako objektu bitartekari batek gauzatuko du, zehatz-mehatz, `ZerbitzuakEJB` motako objektu batek:

```
import java.util.ArrayList;

public interface IZerbitzuak{

  public ErabiltzaileVO egiaztatu(String erabiltzailea, String
pasahitza) throws LoginErabiltzaileEzException, OrokorraException;

public void txertatuErabiltzailea(ErabiltzaileVO erabil1)  throws
InsertarException, OrokorraException;
                                  oOo

public class ZerbitzuakEJB implements IZerbitzuak {

  private FacadeErabiltzailea facadeErabiltzailea;

  private FacadeSaskia facadeSaskia;

  private ProduktuEJBHome produktuHome;

  public ZerbitzuakEJB() {
    hasieratu();
  }

  private void hasieratu() {
    try {
     FacadeErabiltzaileaHome vHome1 =
(FacadeErabiltzaileaHome)EJBHomeFaktoria.getInstance().lookupHome("
FacadeErabiltzailea",FacadeErabiltzaileaHome.class);
      facadeErabiltzailea = vHome1.create();
.. ..
}
    catch (NamingException e) {
      throw new RuntimeException(e.getMessage());
    }
    catch (CreateException e) {
      throw new RuntimeException(e.getMessage());
    }
    catch (RemoteException e) {
      throw new RuntimeException(e.getMessage());   }}
public ErabiltzaileVO egiaztatu(String erabiltzailea, String
pasahitza) throws LoginErabiltzaileEzException, OrokorraException{
```

```
  try{
    ErabiltzaileVO
erabil1=facadeErabiltzailea.egiaztapen(erabiltzailea,pasahitza);
    return erabil1;
  }catch (RemoteException e) {
    throw OrokorraException.orokorraError(e);  }}
```

Aldi berean, `ZerbitzuakEJB` motako objektua Session Façade diseinupatroi ezagunaz baliatuko da sesio-bean bat eraikitzeko eta eskaera prozesatzeko. Horretarako, `EJBObject` interfazea hedatzen duen `FacadeErabiltzailea` interfazea eta `FacadeErabiltzaileBean` sesio-beana erabiliko dira:

```
import javax.ejb.*;

public interface FacadeErabiltzailea extends EJBObject,
IFacadeErabiltzailea{}
                                  oOo


import java.rmi.*;
import javax.ejb.*;

import javax.naming.*;
import java.sql.*;
import javax.sql.*;
import javax.rmi.PortableRemoteObject;


public class FacadeErabiltzaileaBean implements SessionBean,
IFacadeErabiltzailea {

  private boolean hasieratuta=false;

  private ErabiltzaileaEJBHome erabiltzaileaHome;

  private ErabiltzaileaEJB erabiltzailea;
private void hasieraEgin() throws OrokorraException{
    try{
      if (hasieratuta)
        return;
      hasieratuta=true;
      erabiltzaileaHome=
(ErabiltzaileaEJBHome)EJBHomeFaktoria.getInstance().lookupHome("Era
biltzaileaEJB",ErabiltzaileaEJBHome.class);
    }catch(Exception e){
      e.printStackTrace();
      throw OrokorraException.orokorraError(e); }}
public ErabiltzaileVO egiaztapen (String erab, String gakoa) throws
LoginErabiltzaileEzException, OrokorraException{
```

```
     try{
       hasieraEgin();
       ErabiltzaileaEJB erabil1=null;
       erabil1= erabiltzaileaHome.findByPrimaryKey(erab);
       String gakoaBD=erabil1.getGakoa();
       if (gakoa.equals(gakoaBD)){
            erabiltzailea=erabil1;
            return (erabil1.getVO());
       }
} catch (OrokorraException e){
     throw e;
  } catch(Exception e){
     e.printStackTrace();
     throw new LoginErabiltzaileEzException();
  }
     return null;
  }
```

Konturatu ezen, adibidean, diseinu-patroiaren *Eredu* rola egituratzeko EJB osagaiak erabili direla.

3.4. SPRING INGURUNE AURRERATUA

3.4.1. Sarrera

Ignazio Maravillas informatikariak lorerik lore asebete du maitasunaren izaera poliedrikoa. Hala ere, urteak igaro ahala poliedroaren hostoek zorrotzetik kamutserako bidea egin dute patuaren norabidearen aginduetara. Ignazio gajoari geroago eta zailago egiten zaio lore berrien lurruna dastatzea urteak ez baitira alperrik pasatzen gupidagabeko maitasun-ekosisteman. Gorputz geometriko berrien jarrera erasokorrek kikilduta eta ahituta utzi dute informatikariaren bizipoza.

Lanean ere antzeko bilakaera izan du Ignaziok, hasierako Cobol, RPG eta Fortran-ak urrun eta ahaztuta dauzka, Visual Basic ez zuen sekula erabat menderatu eta C bizpahiru proiektutan erabili zuen. Laurogeita hamarreko hamarkadan, bigarren dibortzioaren izapideak egiten ari zela, Javarekin maitemindu zen.

Esan ohi da Javarekin maitemintzea objektuetara zuzendutako garapenarekin maitemintzea dela eta Ignaziori hala gertatu zitzaion. Appletak programatzen ikasi eta bizi-bizi eta objektuetan oinarrituta, bere lehen web aplikazioa eraiki zuen.

Hamarkadaren erdialdera ekosistema aldatu egin zen JavaBean-en espezifikazioaren jaiotzarekin. Bere kate trofikoaren ezaugarriak eguneratu egin ziren osagaiak onartzeko objektuekin gauzatzen ziren garapenetan. Alabaina, eguneroko aplikazioetan faltan botatzen ziren zerbitzu garrantzitsuak, hala nola transakzioen eta ziurtasunaren kudeaketa edota konputazio banatua. Hori konpondu nahian,

1998ko martxoan EJB (Enterprise JavaBean) espezifikazioaren 1.0 bertsioa ematen du argitara Sun Microsystems enpresak

Aurreko ataletan, espezifikazioaren[46] ondorengo bertsioen ezaugarriak eta arkitektura azaldu dira eta argi gelditu da web aplikazioak garatzea teknologia horietan zeharo konplexua dela.

Aztertzen ari garen ingurunean konplexutasuna bi gailurretan kokatzen da: aplikazioaren konplexutasunean eta teknologiaren konplexutasunean. Batak bestea baldintzatu beharko luke eta hala izaten da gehienetan, baina EJBrekin eta J2EErekin, aurreko ataletan aztertu den legez, ez da parekotasun hori azaltzen. Testuinguru horietan teknologiaren konplexutasuna ez da eskalagarria. Osagai bat diseinatu eta programatu nahi bada, nahitaezko hainbat iturburu-kode programatu eta eragiketa gauzatu behar ditu programatzaileak. Hala, osagaiaren bizizikloarekin lotutako metodoak (`ejbCreate()`, `ejbRemove()`, `ejbActivate()`, `ejbPassivate()`...) programatu behar ditu erabili ala ez; osagaiaren `Home`, `Object` eta `Local` interfazeak ezin ditu ahaztu osagaia diseinatzen duenean eta ezarpen deskriptorea behar-beharrezkoa du aplikazioa egikaritzeko.

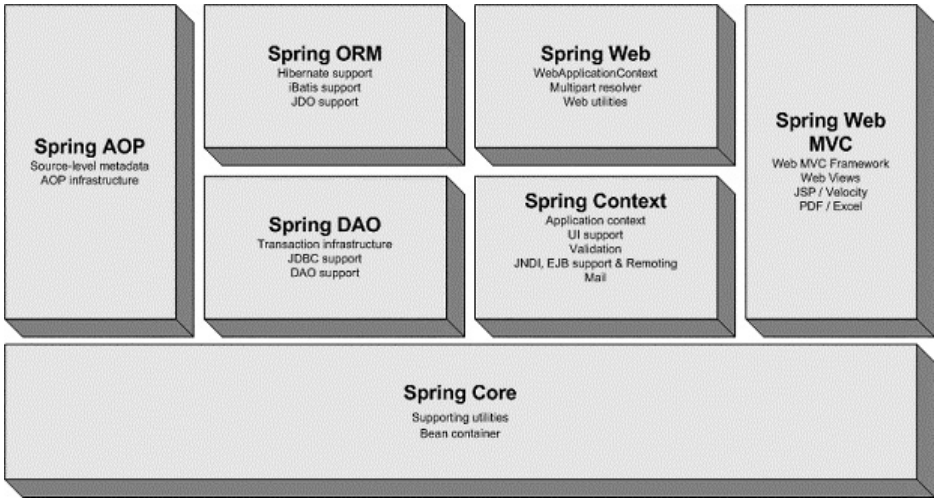
Horrek guztiak, iturburu-kodea astundu eta korapilatu egiten du lotura arriskutsuak sortuz aplikazioaren iturburu-kodean. Horri gehitzen bazaio osagaiaren bilaketa JNDI (Java Naming and Directory Interface) baliabidea erabilita gauzatzen zela, iraunkortasunaren eredua ez zela estandarizatu eta osagaiaren banako probak edukiontziaren barnean gauzatzen zirela edukiontzia behin eta berriz martxan jarrita eta geldituta, programatzaile baikorrena ere etsituta eta ahituta gelditzen zen.

Rod Johnson ez zen horietako bat eta mamuak uxatuta konponbide bila hasi zen. Arazoaren jatorriak eta zergatiak ezagunak zirenez, pare bat liburutan[47] jaso eta dokumentatu zituen.

Arazoaren zergatia ezagutzea interesgarria eta nahitaezkoa zen baina are interesgarriagoa irtenbidea aurkitzea aipatutako arazoei. Nola leundu eta deskorapilatu iturburu-kodea? nola kudeatu iraunkortasuna? nola gauzatu banako probak edukiontzitik kanpo? eta, nola erraztu web aplikazioaren ezarpenaren eta exekuzioaren aurrebaldintzak eta konfigurazioak?

46. Kontu egin, espezifikazio hori J2EE (Java Platform, Enterprise Edition) espezifikazioarekin erlazionatzen dela.

47. J2EE Design and Development . Wrox. 2002.
J2EE Development without EJB. Wrox. 2004.



3.56. irudia. Spring ingurunearen arkitektura orokorra.

Galdera asko eskuz erantzuteko. Softwarearen ingeniaritzaz ari garenez, erantzun dezala ingurune informatiko aurreratu batek. Esan eta egin, eta hona hemen ingurune hori: Spring[48].

Nola leuntzen da eta deskorapilatzen da iturburu-kodea Spring ingurunean? Desakoplatuz iturburu-kodearen osagaien artean gertatzen diren mendekotasun-erlazioak. Horretarako, baztertu egingo da JNDIren bidea osagaiak bilatzeko eta mendekotasun-injekzioa (*dependency injection*) erabiliko da baliabideak osagaietan txertatzeko. Geroxeago aztertuko da eragiketa berri horren esanahia.

Nola kudeatzen da iraunkortasuna? Gurpila berriz asmatu gabe, hau da, garatuta dauden hainbat teknologia ezagun (JPA, Hibernate, JDO, iBATIS, JDBC...) erraz eta azkar integratuz Spring ingurunean, iraunkortasuna inplementatzeko.

Nola gauzatzen dira banako probak edukiontzitik kanpo? Spring erabilita. Ingurune horretan ez du J2EE edukiontzirik martxan jarri beharrik, probatu behar diren osagaiak POJO (Plain Old Java Object) arruntak direlako. Iturburu-kode arrotzen intrusiorik ez denez egiten, osagaiak garbi-garbi programatzen dira egin behar dutena egiteko.

Goian azaldutako Spring ingurunearen ezaugarriek sinplifikatu eta erraztu egiten dute web aplikazioaren garapenaren bizi-zikloaren ezarpena eta exekuzioa. Kontu egin, ingurune horretan, hasierako J2EE teknologiaren osagai konplexua, POJO arrunta bilakatzen dela horrek duen esanahiarekin.

48. Java Development with the Spring Framework. Wrox. 2005.

3.4.2. Spring ingurune aurreratuaren arkitektura orokorra

Spring ingurunearen arkitektura orokorraren (**3.56. irudia**) oinarrizko eta nahitaezko funtzionalitatea edukiontziaren muinean edo bihotzean (Core container) aurkitzen da. Edukiontziaren modulu horretan definitzen da nola eraiki, konfiguratu eta kudeatu beanak. Modulu horren gainean agertzen diren moduluak edukiontziaren zerbitzu bereziak eraikitzeko erabiltzen dira. Laburtuz, lau funtzionalitate egituratzen dira modulu horien bidez:

1. Aspektuetara zuzendutako programazioa (Aspect-Oriented Programming-AOP) gauzatzeko baliabide garatuak eskaintzen ditu AOP moduluak. Spring ingurunearen eta beste AOP inguruneen arteko elkarreragiketak AOP Alliance erakundeak definitutako APIak erabilita gauzatzen dira.

2. Eredu-Bista-Kontrol arkitektura erabili nahi badu aplikazioaren analistak web aplikazioak eraikitzeko, lana errazten dio MVC moduluak, ondo bereiziz domeinuaren objektuak eta kontrolatzaileen logika.

3. Hainbat baliabide eskaintzen dizkio Spring inguruneak programatzaileari iraunkortasuna kudeatzeko JDBC eta DAO (Data Access Object) moduluan. Abstrakzio maila igota, datu-basera atzitzeko programatzaileak idatzi behar duen iturburu-kodea sinplifikatu egiten da. Horrela, datu-basearekin gauzatu behar dituen behe-mailako eragiketak eta konexioak desagertu egiten dira aplikazioaren iturburu-kode nagusitik, aplikazioaren inplementazio garbiagoa lortuz.

   JDBC erabili beharrekoan aplikazioaren iraunkortasuna objektuen eta erlazioen mapaketa (Object-Relational Mapping-ORM) gauzatuta egituratzen bada, O/R Mapping modulua erabilita, hainbat baliabide eskaintzen dizkio inguruneak programatzaileari. Modulu horretan, ingurunearen arkitekturaren beste hainbat lekutan legez, ez da ORM ebazpen berri bat eskaintzen ORM esparruan hainbat aukera eskaintzen baitira, hala nola Hibernate, iBATIS, JDO eta abar. Zubiak eskaintzen ditu inguruneak teknologia horiek arazo gabe integratzeko ingurunearekin garatutako web aplikazioetan, ebazpen malguak eta eraginkorrak eraikita.

4. Mezuen internalizazioa, aplikazioaren gertaeren bizi-zikloa, aplikazioaren egiaztapen-eragiketak eta urruneko transferentziak kudeatzeko, aplikazioaren testuingurua (Application Context) antolatzeko modulua eraiki da ingurunean. Web baliabideak erabiltzen baditu eraiki nahi den aplikazioak, web testuingurura (Web Context) lotutako modulua erabili behar du behar dituen funtzionalitateak eskuratzeko.

Modulu asko eta are baliabide gehiagorekin dena batera eta bat-batean azaltzeko, has gaitezen beraz, apurka-apurka, ingurunearen zehetasunak azaltzen.

Ohi denez, bezeroaren eskaeren adibide ezaguna hartuko da gure lehen Spring aplikazioa eraikitzeko. Aplikazio horretan, `EskaeraList` motako objektua erabiliko da bezeroaren eskaerak biltegiratzeko eta atzitzeko. Objektuak, balizko datu-basearen funtzioa beteko du aplikazioan, eskaerei buruzko informazioa biltegiratzeko. Beraz, lehenengo eta behin, bezeroaren eskaerak eraikiko dira, gero, eskaera horiek `EskaeraList` motako objektuan biltegiratuko dira eta, amaitzeko, eskaerak eskuratuko dira balizko datu-basetik eta irteera estandarretik aterako dira.

Aurreko lerroetan azaldu da ezen Spring ingurunearekin garatutako aplikazioak POJO erako objektuak erabiltzen dituela aplikazioaren osagaiak inplementatzeko; hori egia balitz, aplikazioaren iturburu-kodea klase arruntez osaturik legoke. Ikus dezagun hala den:

```
import java.util.List;

public class EskaeraList {

        private List eskaerak;

        public EskaeraList (){
        }
        public void setEskaerak(List eskaerak) {
                this.eskaerak = eskaerak;
        }
        public List getEskaerak() {
                return eskaerak;
        }
}
                                    oOo
import java.util.List;


public class Eskaera {

        private int eskaeraZenb;
        private List artikuluak;
        private Bezeroa bezeroa;

        public Eskaera () {
        }

        public void setEskaeraZenb(int eskaeraZenb) {
                this.eskaeraZenb= eskaeraZenb;
        }
        public int getEskaeraZenb() {
                return eskaeraZenb;
        }
        public void setBezeroa(Bezeroa bezeroa) {
                this.bezeroa = bezeroa;
```

```
        }
        public Bezeroa getBezeroa() {
                return bezeroa;
        }
        public void setArtikuluak(List artikuluak) {
                this.artikuluak = artikuluak;
        }
        public List getArtikuluak() {
                return artikuluak;
        }

}
                                    oOo
public class Artikulua {

        private String izena;
        private String deskribapena;

        public Artikulua () {
        }

        public void setIzena(String izena) {
                this.izena= izena;
        }
        public String getIzena() {
                return izena;
        }
        public void setDeskribapena(String deskribapena) {
                this.deskribapena= deskribapena;
        }
        public String getDeskribapena() {
                return deskribapena;
        }

}
                                    oOo
public class Bezeroa {

        private String izena;
        private String helbidea;

        public Bezeroa () {
        }

        public void setIzena(String izena) {
                this.izena= izena;
        }
        public String getIzena() {
                return izena;
        }
        public void setHelbidea(String helbidea) {
```

```
            this.helbidea= helbidea;
        }
        public String getHelbidea() {
            return helbidea;
        }
       }
```

Agindu eta hitzartu den bezala, edozein Java aplikazioren itxura hartu du inplementazioak. Iturburu-kodea garbia eta ulergarria da.

Zer da orduan berezi egiten duena Spring ingurunearekin eraikitako aplikazioa? Berezi egiten duena da klaseak nola konfiguratzen eta txertatzen diren aplikazioan.

Spring ingurunean objektuen eraikuntza eta objektuen arteko mendekotasunak XML motako konfigurazio-fitxategiak idatzita adierazten zaizkio edukiontziari. Horrela, beti daki zer egin mendekotasun bat tratatu behar duenean edota non eraiki behar duen objektu bat. Eraikitzen ari garen aplikazioak ere, jakina, bere konfigurazio-fitxategia izango du eta honako hau da:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
        <bean class="EskaeraList">
                <property name="eskaerak">
                        <list>
                                <ref bean="Eskaera1"/>
                                <ref bean="Eskaera2"/>
                        </list>
                </property>
        </bean>

        <bean id="Eskaera1" class="Eskaera">
                <property name="eskaeraZenb">
                        <value>111111</value>
                </property>
                <property name="artikuluak">
                        <list>
                                <ref bean="Artikulu1"/>
                                <ref bean="Artikulu2"/>
                        </list>
                </property>
                <property name="bezeroa">
                        <ref bean="Bezeroa1"/>
                </property>
        </bean>

        <bean id="Eskaera2" class="Eskaera">
                <property name="eskaeraZenb">
                        <value>222222</value>
```

```
                </property>
                <property name="artikuluak">
                        <list>
                                <ref bean="Artikulu2"/>
                                <ref bean="Artikulu3"/>
                        </list>
                </property>
                <property name="bezeroa">
                        <ref bean="Bezeroa1"/>
                </property>
        </bean>

        <bean id="Artikulu1" class="Artikulua">
                <property name="izena">
                        <value>OrdenagailuAAA</value>
                </property>
                  <property name="deskribapena">
                        <value>Ordenagailu POLITA eta EDERRA</value>
                </property>
        </bean>


        <bean id="Artikulu2" class="Artikulua">
                <property name="izena">
                        <value>OrdenagailuBBB</value>
                </property>
                  <property name="deskribapena">
                        <value>Ordenagailu EDERRA eta
POLITA</value>
                </property>
        </bean>

        <bean id="Artikulu3" class="Artikulua">
                <property name="izena">
                        <value>InprimagailuAAA</value>
                </property>
                  <property name="deskribapena">
                        <value>Inprimagailu EDERRA eta POLITA</value>
                </property>
        </bean>


        <bean id="Bezeroa1" class="Bezeroa">
                <property name="izena">
                        <value>Pello Xabier Markiegi</value>
                </property>
                  <property name="helbidea">
                        <value>Askatasuna Etorbidea</value>
                </property>
      </bean>

</beans>
```

Hasierako lerroetan, `EskaeraList` motako objektuaren instantzia deklaratzen du fitxategiak aplikazioa eskaerez hornitzeko. Objektuaren `eskaerak` eremua hasieratzeko `<property>` elementua erabiltzen da. Elementu horrekin, `setEskaerak()` metodoari deitzen dio Spring ingurunearen edukiontziak eta, gainera, `List` motako zerrenda pasatzen dio deiari objektu birekin (`Eskaera1` eta `Eskaera2`). Bestalde, `<ref>` elementuak esaten dio edukiontziari nora joan behar duen beana bilatzera.

Amaitzeko, `<bean>` elementuak adierazten dio edukiontziari nolakoa den klasea eta nola konfiguratu behar duen. Asko laburtuz edukiontziak egiten duena `EskaeraList` motako objektua instantziatzean, honako hau da:

```
EskaeraList eskaeraZerrenda = new EskaeraList();
Eskaera eskaera1 = new Eskaera();
Eskaera eskaera2 = new Eskaera();
List zerrenda = new ArrayList();
zerrenda.add(eskaera1);
zerrenda.add(eskaera1);
eskaeraZerrenda.setEskaerak(zerrenda);
```

XML fitxategiarekin nola gauzatu objektuen instantziazioak eta mendekotasunak adierazten zaio edukiontziari baina nola deitu eta kargatzen da edukiontzia bera? Honako hau programatuta:

```
import java.util.List;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.apache.log4j.Logger;
import java.io.FileInputStream;

public class EskaeraIrakurri {

private static final org.apache.log4j.Logger LOGGER =
org.apache.log4j.Logger
.getLogger(EskaeraIrakurri.class);

        public static void main(String args[]) throws Exception{

       LOGGER.debug("EskaeraIrakurri Aplikazioa martxan");

                BeanFactory beanFactory = new XmlBeanFactory(
                                new
FileInputStream("eskaera_configuration.xml"));

                EskaeraList eskaeraList = (EskaeraList)
beanFactory.getBean("EskaeraList");

                List nireEskaerak = eskaeraList.getEskaerak();
```

```
                for (int i = 0; i < nireEskaerak.size(); i++) {
                        Eskaera oraingoEskaera= (Eskaera)
nireEskaerak.get(i);
         System.out.println("Eskaera Zenbakia : " +
oraingoEskaera.getEskaeraZenb());
         System.out.println("Eskaeraren Bezeroaren Izena : " +
oraingoEskaera.getBezeroa().getIzena());
                        System.out.println();

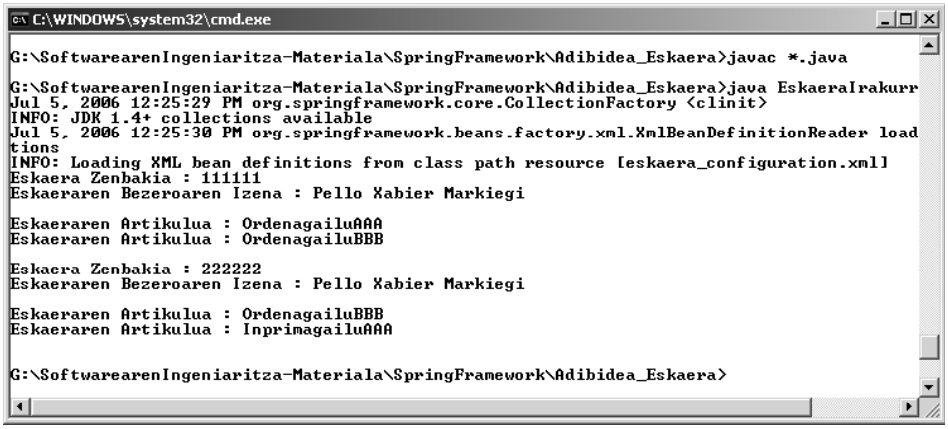
          List nireArtikuluak = oraingoEskaera.getArtikuluak();

                for (int j = 0; j < nireArtikuluak.size(); j++) {
                        Artikulua oraingoArtikulua= (Artikulua)
nireArtikuluak.get(j);
                 System.out.println("Eskaeraren Artikulua : " +
oraingoArtikulua.getIzena());   }
                           System.out.println();
                }
        LOGGER.debug("EskaeraIrakurri Aplikazioaren amaiera ");
      } // main

} //EskaeraIrakurri
```

Iturburu-kodean lehendabizi, `BeanFactory` motako objektua eraikitzen da beanen faktoria sortzeko. Faktoriak `getBean()` metodoa erabiltzen du objektuak eraiki eta aplikazioan txertatzeko. Kontu egin ezen aplikazioak ez duela jakin beharrik objektu horien jatorria non dagoen, hala nola POJO lokala, urruneko web zerbitzua edota mock osagaia izan daiteke eta aplikazioak berdin funtzionatu.

XML fitxategia `XmlBeanFactory` motako objektua eraikita txertatzen da aplikazioaren exekuzio-fluxuan. Bezeroaren eskaerak eskuratzen ditu gainerako iturburu-kodeak eta aplikazioaren irteera estandarretik (**3.57. irudia**) ateratzen ditu.



3.57. irudia. Adibidearen emaitzak irteera estandarrean.

3.4.3. Mendekotasun-injekzioa

Aurreko atalean eraiki den aplikazio txikiaren iturburu-kodea aztertzen bada azaletik eta lehen begiradan, idatzitakoak ez du ezer berezirik. Bakarrik, hasieran, konfigurazio-fitxategia erabilita bi eragiketa txiki egiten dira: Spring ingurunearen edukiontzia kargatu eta beanak atzitu. Beste guztia, ezaguna eta ulergarria da.

Hasierako eragiketa horiek zehatzago aztertuko ditugu atzean dagoen esanahi sakona ulertzeko. Demagun lehen aztertu den `Eskaera` klasea zertxobait aldatzen dela eta honako hau idazten dela iturburu-kodean:

```
public class Eskaera {
private String oharrak;
private FakturaNagusia faktura;

public Eskaera(String oharrak) {
this.oharrak = oharrak;
faktura = new FakturaNagusia();
}

public EmateAgiria bilatuEmateAgiria()
throws EmateAgiriaNotFoundException {
return faktura.bilatuEA();
}
}

public class FakturaNagusia {

public FakturaNagusia() {}

public EmateAgiria bilatuEA() throws EmateAgiriaNotFoundException {
EmateAgiria emateAgiria = null;
// Bilatu  emateAgiria
…
return emateAgiria;
}
}
```

Hau da, `FakturaNagusia` motako eremu berria txertatu da `Eskaera` klasearen eremuen aldean eta `bilatuEmateAgiria()` metodo berria programatu da, zeinak `EmateAgiria` motako objektua itzultzen duen, `FakturaNagusia` motako objektuaren `bilatuEA()` metodoa egikarituta.

Zer gertatu da eguneraketa horiek txertatuta iturburu-kodean? Iturburu-kodea korapilatu egin dela objektuak akoplatuta lehen baino dezente gehiago. Hala, banako proba bat antolatzen bada emate-agiria bilatu den ala ez egiaztatzeko, azaletik dirudi ezen `Eskaera` eta `EmateAgiria` objektuekin ari dela bakarrik proba lanean, baina, iturburu-kodeak argi esaten digu ezen `FakturaNagusia` klasearen baliabideak ere behar dituela proba gauzatzeko, hau da, nahitaez ezkutuko

osagaien portaera ere aztertzen ari da proba, bere azaleko osagaiak aztertzen dituen bitartean. Hona hemen nola gelditzen den azaldu den proba iturburu-kode eran jarrita:

```
import junit.framework.TestCase;
public class EskaeraTest extends TestCase {

public void testBilatuEmateAgiria() {

Eskaera eskaera =
new Eskaera("ArazoGabe");
try {
EmateAgiria emateAgiria = eskaera.bilatuEmateAgiria();
assertNotNull(emateAgiria);
assertTrue(emateAgiria.isAdos());
} catch (EmateAgiriaNotFoundException e) {
fail();
}
}
```

Arazoa konpontzeko bide bat da iturburu-kodea desakoplatzea interfazeak erabilita. Has gaitezen bide hori aztertzen:

```
public interface FakturaI {
public abstract Object bilatuEA() throws FakturaException;
}

public interface EskaeraI {
public Object bilatuEmateAgiria() throws FakturaException;
}

public class FakturaNagusia implements FakturaI {
public FakturaNagusia() {}
public Object bilatuEA() throws FakturaException {
// .........
return new EmateAgiria();
}


public class Eskaera implements EskaeraI {
private FakturaI faktura;
…
public Eskaera(String oharrak) {
faktura = new FakturaNagusia();
…
}
public Object bilatuEmateAgiria() throws FakturaException {
return faktura.bilatuEA();
}
}
```

Arazoa ekiditeko bi interfaze berri programatu dira buruhausteak eman dituen metodoa birmoldatzeko. Inplementazio berrian, `Object` erako osagaia itzultzen du metodoak (`bilatuEmateAgiria`) interfazearen trikimailua erabilita. Horrekin, behintzat, orokortu egin da metodoaren portaera `Object` erako itzulera adierazita, baina, zoritxarrez, nonbait eraiki behar du objektu zehatza exekuzioan arazorik ez izateko. Adibidean, `Eskaera` klasearen metodo eraikitzailean egikaritzen da aipatutako eragiketa:

```
faktura = new FakturaNagusia();
```

Egin den lan guztia arazoa konpontzeko pikutara joan da. Azkenean, berriz, iturburu-kodea akoplatu egin da. Demagun fakturak bi motatakoak izan daitezkeela: `FakturaNagusia` motakoa edo `FakturaZatika` motakoa. Egoera berri horretan `Eskaera` klasearen metodo eraikitzailea birprogramatu eta egokitu egin behar du programatzaileak, faktura mota berriak tratatzeko, fakturen faktoria antzeko zerbait antolatuta.

Arazo bat konpondu nahian beste bat sortu da. Etsi baino lehen azter dezagun Spring inguruneak eskaintzen duen irtenbidea aipatutako arazoa konpontzeko. Interfazeekin egindako lana ez da baztertuko eta berreskuratu egingo da baina iturburu-kodea sinplifikatu egingo da objektuak instantziatzean eta esleipenak egitean. Hona hemen nola gelditzen den iturburu-kodea aipatutako eguneraketak txertatuta:

```
public class Eskaera implements EskaeraI {
private FakturaI faktura;
…
public Eskaera(String oharrak) {
…
}
public EmateAgiria bilatuEmateAgiria() throws FakturaException {
…
return faktura.bilatuEA();
}
public void setFaktura(FakturaI faktura) {
this.faktura = faktura;
}
}
```

Hau da, inplementazioa zeharo desakoplatu da, zeren objektuen arteko mendekotasunak desagertu egin baitira aplikazioaren iturburu-kodetik.

Baina, nola instantziatzen da `Eskaera`? Zein faktura motaren instantzia erabiltzen du `bilatuEmateAgiria()` metodoak? Eta, nola esleitzen zaio `faktura` eremuari instantziatutako faktura mota bat? Spring ingurunearen mendekotasun-injekzioa egikarituta eta XML konfigurazio-fitxategia eraikita, adierazteko nola gauzatu aplikazioaren instantziazioak:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
      <beans>
<bean id="faktura"
    class="FakturaNagusia"/>
<bean id="eskaera"
    class="Eskaera">
<constructor-arg>
    <value>ArazoGabe</value>
</constructor-arg>
<property name="faktura">
    <ref bean="faktura"/>
</property>
</bean>
</beans>
```

`Eskaera` klasearen metodo eraikitzailearen parametroaren balioa `<constructor-arg>` elementuan adierazten da konfigurazio-fitxategian. Bestalde, `Eskaera` klasearen `setFaktura()` metodoaren parametroaren balioa zein izango den finkatzen du `<property>` elementuak.

Kontu egin ezen mendekotasun-injekzio eragiketa berri horrekin aldatu egiten dela instantziazioaren norabidea. Lehen, aplikazioaren objektua arduratzen zen instantziazioez, orain, Spring ingurunean, objektua ez da arduratzen eragiketa horretaz, behar dituenean, inguruneak hornitzen dizkio.

Itzul gaitezen berriro EJB teknologiaren eremura eskaeren adibidearen azterketa amaitzeko. Egoera berri horretan bezeroaren eskaerak web aplikazioa erabilita tratatuko dira eta egoera gabeko bean batek (`EskaeraZerbitzua`) kudeatuko ditu. Hasieran, HTML orria erabilita egiten du web aplikazioaren erabiltzaileak bere eskaera. Eskaera jasotzen du aplikazioaren zerbitzariak eta EJB osagaiari (`EskaeraZerbitzua`) pasatzen dio prozesa dezan. Hona hemen eragiketa horiek egikaritzeko behar duen iturburu-kodea:

```
...
private EskaeraZerbitzua eskaeraZerbitzua;
public void doRequest(HttpServletRequest galdera) {
Eskaera eskaera = createEskaera(galdera);
EskaeraZerbitzua eskaeraZerbitzua = getEskaeraZerbitzua();
eskaeraZerbitzua.createEskaera(eskaera);
}
private EskaeraZerbitzua getEskaeraZerbitzua() throws
CreateException {
if (eskaeraZerbitzua == null) {
Context initial = new InitialContext();
Context myEnv = (Context) initial.lookup("java:comp/env");
Object ref = myEnv.lookup("ejb/EskaeraZerbitzuaHome");
EskaeraZerbitzuaHome home = (EskaeraZerbitzuaHome)
```

```
PortableRemoteObject.narrow(ref, EskaeraZerbitzua.class);
eskaeraZerbitzua = home.create();
}
return eskaeraZerbitzua;
}
```

Iturburu-kodetik zatirik deigarriena `getEskaeraZerbitzua()` metodoa da. Jakin badakigu, osagai bat eskuratu nahi denean EJB teknologian, hainbat eragiketa egiten direla behin eta berriz, hau da, JNDI baliabidearen testuingurua eskuratu, JNDI erabilita osagaiaren EJB `Home` elementua bilatu eta eskuratu, eta amaitzeko, `Home` baliabidea erabili osagaia eraikitzeko eta eskuratzeko. Benetan, lan aspergarria, baldin osagai asko erabiltzen baditu aplikazioak.

Arazoa zertxobait leuntzen da Service Locator diseinu-patroia[49] erabiltzen bada. Horrela, behe-mailako zerbitzuen hornitzaile generikoa eraikitzen da, ahal den neurrian, orokorrak diren eragiketak automatizatzeko. Hona hemen, nola geldituko zen aplikazioari arazoak ematen dizkion `getEskaeraZerbitzua()` metodoa:

```
...

private EskaeraZerbitzua getEskaeraZerbitzua() {
EskaeraZerbitzuaHome home =
ServiceLocator.locate(EskaeraZerbitzuaHome);
EskaeraZerbitzua eskaeraZerbitzua = home.create();
}
```

Hala ere, arazo nagusia ez da desagertzen: osagaien bilaketa nahitaezkoa da eta behin eta berriz gauzatuko da. Arazoa Spring ingurunean kudeatzen bada, ikus dezagun zer gertatzen den:

```
private EskaeraZerbitzua eskaeraZerbitzua;
public void doRefaktura(HttpServletRequest galdera) {
Eskaera eskaera = createEskaera(galdera);
eskaeraZerbitzua.createEskaera(eskaera);
}
public void setEskaeraZerbitzua(EskaeraZerbitzua eskaeraZerbitzua)
{
this.eskaeraZerbitzua = eskaeraZerbitzua;
}
```

Aleluia ! desagertu egin da !

Spring ingurunean ez diogu inongo zerbitzuri deitu beharrik, mendekotasun-injekzioz, aplikazioaren exekuzio-fluxuan txertatzen da osagaia zuzenean.

---

49. `http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html`
`http://java.sun.com/blueprints/patterns/ServiceLocator.htm`

Orain arte azaldu diren adibideetan, Spring ingurunearen oinarrizko funtzionalitateak aztertu dira, agerian uzteko EJB 2.0 espezifikazioaren gabeziak. Geroxeago azalduko denez, EJB 3.0 espezifikazio berrian, autokritika sakona eginda, hainbat berrikuntza onartuko dira aipatutako arazoak konpontzeko. Gainera deigarriena izango da espezifikazio berriaren arkitektura erabat hurbilduko dela aztertzen ari garen Spring ingurune aurreratura.

Amaitzeko, azken adibidea aurkeztuko da zeinak domeinuaren logika eta aplikazioaren iraunkortasuna harremanetan jarriko dituen DAO diseinu-patroi ezaguna erabilita.

Jakin badakigu ezen aplikazioaren betiko hiru geruzak modularizatu gabe inplementatzen badira, iturburu-kodea, monolitikoa eta erabat akoplatua geldituko dela. Aplikazioa txikia bada, agian ez da arazorik egongo etorkizunean betekizun berriak edota aldaketak txertatzeko. Alabaina, gehienetan estu eta larri ibiliko da programatzailea, eguneraketak eta aldaketak txertatzeko. Aplikazioaren konplexutasunari teknologia informatikoaren konplexutasuna gehitu behar dio eta askotan bigarren hori ahaztu egiten du.

Beraz, aplikazioa geruzetan zatitzea nahitaezkoa da teknologia konplexuekin lan egiten denean. Gainera, ikuspegi horrek ohitura onak sartzen ditu erakundean eta, ondorioz, kalitatezko softwarea egitera bultzatzen du programatzailea.

Azken adibidean, estrategia hori erabiliko da, hau da, domeinuaren logika ez da zuzenean jarriko harremanetan iraunkortasunarekin, horretarako, DAO diseinu-patroia erabiliko du bitartekari-lanak egiteko.

Bestalde, aztertuko den adibideak, Spring tresna aurreratuaren JDBC ingurunea azalduko du enpresa baten pertsonalaren domeinuan arreta jarrita. Zehatzmehatz, pertsonalaren soldatak arakatuko ditu inplementatu nahi den kudeaketa-arauak. Langile guztiak aztertu ondoren, bakarrik eskuratu nahi dira, soldata-tarte baten barruan kokatzen diren langileak. Langileak, datu-base erlazionalean daude biltegiratuta eta iturburu-kodea ondo dagoela programatuta ziurtatzeko, amaieran, banako proba bat egikarituko da.

Has gaitezen, beraz, DAO diseinu-patroia inplementatzen duen aplikazioa programatzen:

```
import java.util.List;
import java.util.Map;
public interface ILangileaDAO {

       //SQL galderan egikarituko den Stringa
       public String FIND_BY_SOL_TARTE = "SELECT LANG_IZENA,LANG_ID,
SOLDATA "
       + " FROM LANGILEA "
       + " WHERE SOLDATA >= :MIN_SOLDATA AND SOLDATA <=:MAX_SOLDATA";
```

```
       // Langileen zerrenda itzultzen du

       public List findBySoldataTarte(Map soldataMap);
}
                                   oOo
import java.util.List;
import java.util.Map;

public interface ILangileaBusinessService {
    public List getLangileakSoldataTarte(Map soldataMap);
}
oOo
import ILangileaDAO;
import java.util.List;
import java.util.Map;

public class LangileaBusinessServiceInpl implements
ILangileaBusinessService {

    private ILangileaDAO langDAO;
    public List getLangileakSoldataTarte(Map soldataMap){

        List langList = langDAO.findBySoldataTarte(soldataMap);
        return langList;
    }
    public void setDao(ILangileaDAO langDAO){
        this.langDAO = langDAO;
    }
}
oOo
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;
import Langilea;

public class LangileaDAOInpl implements ILangileaDAO{

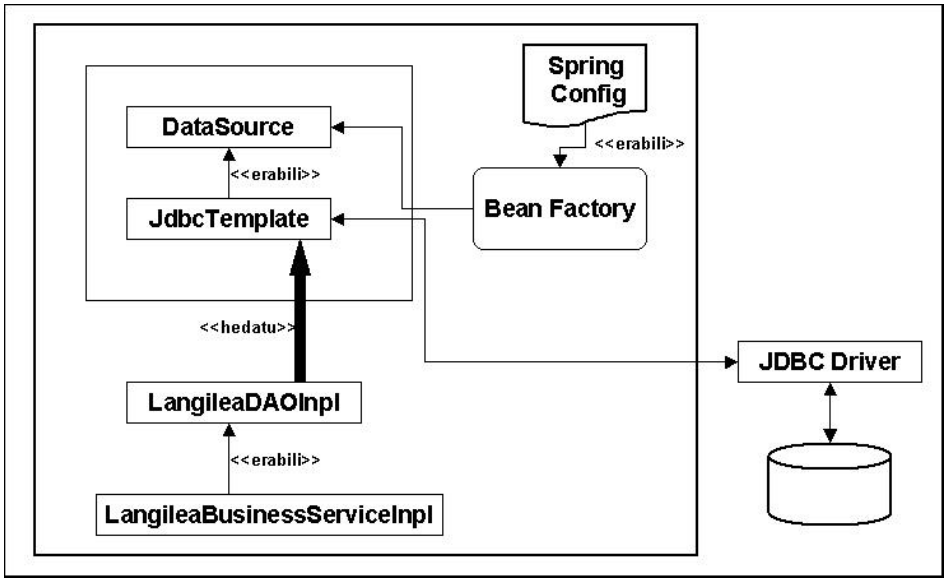
  public List findBySoldataTarte(Map soldataMap)
  {
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    List langList = new ArrayList();

    // Objektu bitartekaria geruza artean mugitzeko
    Langilea lanLangilea = null;
```

```
try{
// Klase laguntzaileak konexioa gauzatzeko
  conn = DriverManager.getConnection();
  pstmt = conn.prepareStatement(FIND_BY_SOL_TARTE);
  pstmt.setDouble(1, Double.valueOf( (String)
      soldataMap.get("MIN_SOLDATA") );
  pstmt.setDouble(2, Double.valueOf( (String)
      soldataMap.get("MAX_SOLDATA") );
  rs = pstmt.executeQuery();
  int lanLangileId = 0;
  String lanLangileIzena = "";
  double lanSoldata = 0.0D;
  while (rs.next()){
    lanLangileId = rs.getInt("LANG_ID");
    lanLangileIzena = rs.getString("LANG_IZENA");
    lanSoldata = rs.getDouble("SOLDATA");
    lanLangilea = new Langilea(lanLangileId,
          lanLangileIzena,
          lanSoldata);
    langList.add(lanLangilea);
  }//amaiera while
}//amaiera try
catch (SQLException sqle){
  throw new DBException(sqle);
}//amaiera catch
finally{
  try{
    if (rs != null){
      rs.close();
    }
  }
  catch (SQLException sqle){
    throw new DBException(sqle);
  }
  try{
    if (pstmt != null){
      pstmt.close();
    }
  }
  catch (SQLException sqle){
    throw new DBException(sqle);
  }
  try{
    if (conn != null){
      conn.close();
    }
  }
  catch (SQLException sqle){
    throw new DBException(sqle);
  }
```

```
    }//amaiera finally
    return langList;
  }//amaiera findBySoldataTarte metodoa
}
```

Hasierako bi interfazeekin adierazten da zeintzuk izango diren DAO diseinu-patroiaren eta domeinuaren logikaren goi-mailako funtzionalitateak. Hurrengo klaseak domeinuaren logika inplementatzen du, eta amaitzeko, DAO diseinu-patroiaren interfazea inplementatzen duen `LangileaDAOInpl` klasea programatzen da. Azken klase hori da benetan interesatzen zaiguna, hori baita aldatuko dena Spring ingurunea erabilita. Klasearen iturburu-kodea aztertzen badugu, berehala konturatzen gara non dagoen arazo larriena: SQL agindu arrunta egikaritzeko hainbat kode programatzen dela behin eta berriz. Hala nola konexioak eta `Statement` motako objektuak eraikiko dira, `ResultSet` motako objektuan gordeko dira emaitzak eta `Langilea` motako objektuak eraikiko dira itzuliko den zerrenda patxadaz eraikitzeko.



3.58. irudia. Adibidearen arkitektura orokorra.

Jakina, arazoak ekiditeko hainbat salbuespen-egoera ere kontuan hartu beharko ditu programatzaileak, kodea hedatuz eta hedatuz bere borondatearen kontra.

Inplementa dezagun aplikazio bera Spring ingurunea erabilita. Aplikazioaren arkitektura nagusia **3.58. irudia**n jasotzen da. Kontu egin ezen kontzeptualki aldaketa bakarra DAO diseinu-patroiaren inplementazioan dagoela. Oraingo honetan, ingurunearen baliabideak hedatzen dira diseinu-patroia inplementatzeko:

```
import java.util.List;
import java.util.Map;
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcDaoSupport;
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.support.JdbcDaoSupport;
import org.springframework.jdbc.core.JdbcTemplate;

public class LangileaDAOInpl extends NamedParameterJdbcDaoSupport
implements ILangileaDAO{

    public List findBySoldataTarte(Map soldataMap){

        NamedParameterJdbcTemplate daoTmplt =
getNamedParameterJdbcTemplate();
        return
daoTmplt.query(ILangileaDAO.FIND_BY_SOL_TARTE,soldataMap,new
LangileaTOMapper());
    }
}
```

Aurreikusi dugun legez, iturburu-kodearen luzera nabari jaitsi da eta are gehiago Spring inguruneareN `RowMapper` baliabidea inplementatuta:

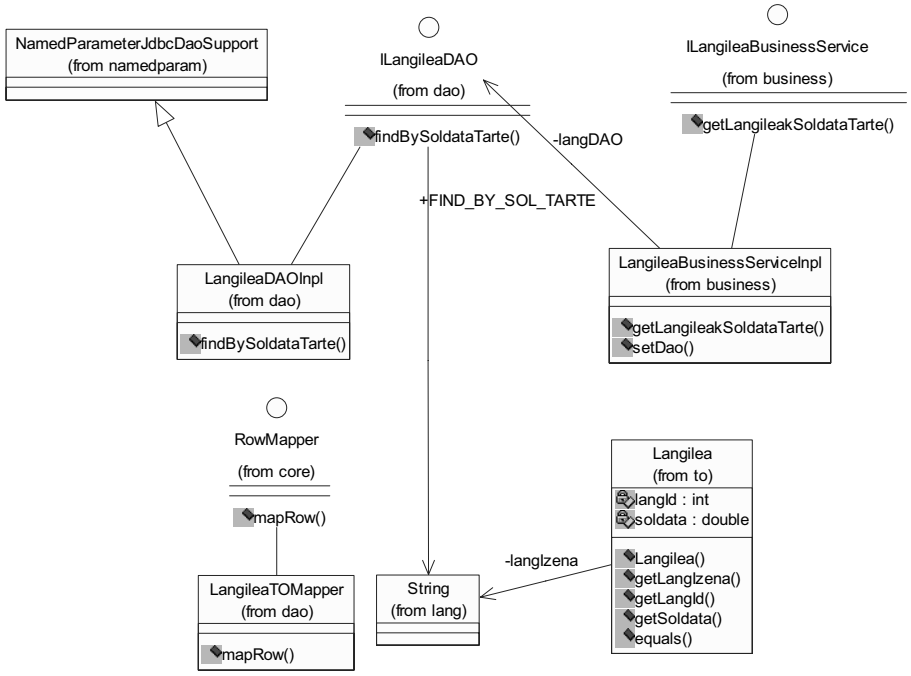
```
import Langilea;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class LangileaTOMapper implements RowMapper{
    public Object mapRow(ResultSet rs, int rowNum) throws
SQLException{
        int langId = rs.getInt("LANG_ID");
        String langIzena = rs.getString("LANG_IZENA");
        double soldata = rs.getDouble("SOLDATA");

        Langilea langilea = new Langilea(langId,langIzena,soldata);

        return langilea;
    }  }
```

Baliabide hori erabilita, SQL galderaren emaitza erabat egituratuta gelditzen da eta arazorik gabe itzultzen du diseinu-patroiak emaitza `List` motako objektuan. **3.59. irudia**k islatzen du nola erlazionatzen diren aplikazioaren klaseak eta inguru-nearen baliabideak.



3.59. irudia. Adibidearen klaseen eta interfazeen erlazioak.

Aplikazioaren inplementazioa amaitzeko, aipatu ditugun osagai guztiak harremanetan jarriko dituen klase nagusia programatzen da:

```
import ILangileaBusinessService;
import java.util.HashMap;
import java.util.Map;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import
org.springframework.context.support.FileSystemXmlApplicationContext;

public class Nagusia {

    public static void main(String[] args) {

        ApplicationContext ac = new
FileSystemXmlApplicationContext("C:/SpringConfig/Spring-Config.xml");
        ILangileaBusinessService business =
(ILangileaBusinessService)ac.getBean("LANG_BUSINESS");
        Map hm = new HashMap();
        hm.put("MIN_SOLDATA","1");
        hm.put("MAX_SOLDATA","1000");
        System.out.println(business.getLangileakSoldataTarte(hm));

    }   }
```

Jakina, DAO diseinu-patroia inplementatzen duen klasearen instantziazioa eta kudeaketa-araua inplementatzen duen klasearen instantziazioa XML konfigurazio-fitxategia prozesatuta gauzatuko ditu Spring inguruneak:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
       <!—Datuen iturburuak konfiguratu —>

<bean id="ITURBURU_NAGUSIA"
              class="org.springframework.jndi.JndiObjectFactoryBean">
              <property name="jndiEnvironment">
                     <props>
                            <prop key="java.naming.factory.initial">
                                  weblogic.jndi.WLInitialContextFactory
                            </prop>
                            <prop key="java.naming.provider.url">
                                  t3://localhost:7001
                            </prop>
                     </props>
              </property>
              <property name="jndiName">
                     <value>
                            jdbc/DBPool
                     </value>
              </property>
       </bean>

    <!— Konfiguratu DAO —>
       <bean id="LANG_DAO" class="dao.LangileaDAOInpl">
              <property name="dataSource">
                     <ref bean="ITURBURU_NAGUSIA"></ref>
              </property>
       </bean>
       <!— Konfiguratu Business Service —>
       <bean id="LANG_BUSINESS"
         class="business.LangileaBusinessServiceInpl">
              <property name="dao">
                     <ref bean="LANG_DAO"></ref>
              </property>
       </bean>
</beans>
```

Dena ondo dagoela egiaztatzeko agindutako banako proba programatzen da:

```
import Langilea;
import java.util.ArrayList;
import java.util.HashMap;
import junit.framework.*;
import ILangileaDAO;
import java.util.List;
import java.util.Map;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.FileSystemXmlApplicationContext;

public class LangileaBusinessServiceInplTest extends TestCase {

    private ILangileaBusinessService langBusiness;
    private Map soldataMap;
    List gureEmaitza;

    protected void setUp() throws Exception {
        initSpringFramework();
        initSoldataMap();
        initEmaitza();
    }
    private void initEmaitza () {
        gureEmaitza = new ArrayList();
        Langilea langilea = new Langilea(2,"Koldo Ameztoy",46.11);
        gureEmaitza.add(langilea);
    }
    private void initSoldataMap() {
        soldataMap = new HashMap();
        soldataMap.put("MIN_SOLADATA",new Double(1));
        soldataMap.put("MAX_SOLDATA",new Double(50));
    }
    private void initSpringFramework() {
ApplicationContext ac = new
FileSystemXmlApplicationContext("C:/SpringConfig/Spring-Config.xml");
        langBusiness =
(ILangileaBusinessService)ac.getBean("LANG_BUSINESS");
    }
    protected void tearDown() throws Exception {
    }
    /**
     * Metodoaren proba getLangileakSoldataTarte
     */
    public void testGetLangileakSoldataTarte() {
        List emaitza =
langBusiness.getLangileakSoldataTarte(soldataMap);
        assertEquals(gureEmaitza, emaitza);
    }

    public void assertEquals(List gureEmaitza, List emaitza)
    {
```

```
Langilea gureLang = (Langilea) gureEmaitza.get(0);
Langilea egungoLang = (Langilea) emaitza.get(0);
if(!gureLang.equals(egungoLang))
{
    throw new RuntimeException("Proba arazoarekin");
} } }
```

Lehenagotik dakigun bezala, interesatzen zaizkigun hasieraketak[50] egiten ditu `setUp()` metodoak eta proba (`testGetLangileakSoldataTarte()`) egikaritzen da.

3.5. LABURPENA

Oinarrizko bezero/zerbitzari aplikazioa eraiki nahi denean, RMI teknologiak eskaintzen duen azpiegitura nahikoa izaten da aplikazioa azkar garatzeko. Urruneko salbuespenak eta zertxobait gehiago kontrolatuta, aplikazioa, ziztu bizian eraikitzen da. Aplikazioaren konplexutasun maila txikia denean, arestian aipatutako bidea egokia izan daiteke eta ez dugu saltsa handiagoetan sartu behar programatzailea.

Alabaina, aplikazioaren konplexutasun maila etorkizunean igo egiten bada betekizun berriak erantsita, edota hasiera-hasieratik konplexua bada, RMIrekin eraikitzen den oinarrizko azpiegitura motz gelditukо da hiru mailako arkitekturaren arlo guztiak inplementatzeko.

Aurkezpen mailaren azaleratzea web nabigatzailea erabilita gauzatzen bada, appleten teknologia erabiliko da geruzaren arlo hori lantzeko.

Urruneko objektuak atzitu eta eskuratu nahi baditu aplikazioak web zerbitzariak erabilita, JSP orriak edota servletak programatu beharko ditu programatzaileak urruneko eskaerak bidaltzeko eta erantzunak jasotzeko. Halaber, edukiontzi aurreratu bat erabili nahi badu aplikazioaren segurtasunaren, transakzioen edota iraunkortasunaren kudeaketa antolatzeko, J2EE teknologiaren baliabideek asko erraztuko diote bidea, edukiontziak kudeatuko baititu zerbitzu horiek, automatikoki.

J2EE teknologiaren hasierako bertsioek[51] edukiontzi pisutsuaren (*heavyweight container*) fama bereganatu zuten hainbat eragiketa edukiontziaren barnean gauzatzen baitziren aplikazioaren garapen bizi-zikloa nabari luzatuz. Probak

50. Kontu egin, hasieraketa, hiru urratsetan gauzatzen dela:
    1. Spring ingurunea martxan jarri.
    2. Finkatu aztertu nahi den tartea.
    3. Eraiki egiaztatu nahi den emaitza.

51. Geroago aztertuko da EJB 3.0 espezifikazioak, aurreko bertsioei egin zaizkien kritikak onartuta, hainbat berrikuntza txertatu dituela edukiontziaren zeregina sinplifikatzeko eta eraginkortasuna hobetzeko.

edukiontziaren barnean gauzatzen ziren eta iraunkortasunaren trataera nahasia eta konplexua zen.

Spring ingurune aurreratuak, edukiontzi leunen (*lightweight container*) printzipioetan oinarrituta, aipatutako arazo gehienak konpontzen ditu arkitektura berria eta egokitua eraikita programatzaileari.

3.6. OINARRIZKO BIBLIOGRAFIA

Ashmore, D. (2004): *The J2EE Architect's Handbook*, Dvt Press.

Johnson, R.; Hoeller, J.; Arendsen, A.; Risberg T. eta Sampaleanu C. (2005): *Professional Java Development with the Spring Framework*, Wrox.

Johnson, R. eta Hoeller, J. (2004): *Expert One-on-One J2EE Development without EJB,* Wrox.

Keith, M. eta Schincariol, M. (2006): *Pro EJB 3: Java Persistence API*, Apress.

Monson-Haefel R.; Burke, B. eta Labourey S. (2001): *Enterprise JavaBeans*, O´Reilly.

Panda, D.; Rahman, R. eta Lane, D. (2007): *EJB 3 in Action*, Manning.

Roman, E.; Sriganesh, R. P. eta Brose, G.(2005): *Mastering Enterprise JavaBeans*, Wiley.

Sierra, K. eta Bates, B. (2003): *Head First EJB*, O´Reilly.

Sullins, B. G. eta Whipple, M. B. (2003): *EJB Cookbook*, Manning.

Williamson, A. R. (1999): *Java Servlets by Example*, Manning.

3.7. WEB GUNE INTERESGARRIAK

Zer dago bertan	Web gunea
Appleten tutoretza	http://java.sun.com/docs/books/tutorial/deployment/applet/appletsonlyindex.html
Servleten tutoretza	http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
JSP orrien informazioa	http://en.wikipedia.org/wiki/JavaServer_Pages
Sarrera J2EE teknologiara	http://java.sun.com/j2ee/overview.html
J2EEren tutoretza	http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
EJB espezifikazioa	http://en.wikipedia.org/wiki/Enterprise_JavaBean
Hibernate/EJB	http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateVSEJB3
Spring ingurunea(I)	http://www.springframework.org/
Spring ingurunea(II)	http://en.wikipedia.org/wiki/Spring_Framework
Spring ingurunea(III)	http://www.springframework.net/

4. Aplikazio informatikoen diseinu motak eta iraunkortasunaren trataera aurreratuak

4.1. SARRERA

Aplikazio informatiko bat eraikitzen hasi baino lehen hainbat erabaki hartzen dira erabat baldintzatuko dituztenak aplikazioaren bai analisia eta diseinua eta bai inplementazioa ere.

Demagun hiru mailako arkitektura ezaguna erabiltzen dela aplikazioa garatzeko. Hori esanda, ibilbidearen lehen urratsa egin dugu baina oraindik hainbat bihurgune eta aldapa gelditzen zaizkigu. Hala nola: arkitektura garatzeko teknologia zein izango da JSF (Java Server Faces) ala Struts? Domeinuaren logika lantzeko era prozedurala edo objektuetara zuzendutakoa erabiliko da? Nola kapsulatuko da domeinuaren logika hori? Aplikazioaren iraunkortasuna garatzeko zein ikuspegi hautatuko da?

Atal honetan galdera horiei erantzuten saiatuko gara era egituratuan, hau da, erantzunaren nondik norakoa baldintzatua etorriko da zirkunstantzien arabera eta egoera horiei aurre egin beharko diegu.

Batzuetan, erakundearen lan-taldearen heldutasun mailak bideratuko du prozesua. Beste batzuetan, erakundearen aurretiko aukera teknologikoek baldintzatuko dute aplikazioaren norabidea. Aplikazioaren konplexutasuna bera ere ez da isilik gelditukо erabakitze-prozesu horretan.

Horrekin guztiarekin esan nahi da ezen informatikan, tamalez, bizitza errealean legez, ez dugula era estandar eta adostu bat beti balio diguna aplikazioak eraikitzeko. Prozesuan, hainbat aldagai agertuko dira balio barik eta balioak eman ahala, alde batera edo bestera eramango du norabide-lemak ontzia.

Jakina, iparrerantz joatea erabaki badugu berokiz hornitu beharko da ontzia, aldiz, hegorantz bagoaz bainujantziak eta itzalpeak izango ditugu lagun. Dena dela, gora edo behera joan, eramango den zama bera izango da, hau da, aztertzen duguna aztertu, bai teknologia aldetik eta bai arkitektura aldetik beti erabiliko dugu domeinu bera gure ibilbidean. Hala, errazago aztertu eta konparatu ahal izango ditu irakurleak aplikazioa eraikitzeko erabili diren estrategia desberdinak.

Hasieran, egoerarik okerrenean jarriko gara eta ikuspegi prozeduralaren ezaugarriak aztertuko dira bere alde ahulak eta sendoak azalduz. Ondoren, ikuspegi ausartagoak eta helduagoak erabilita, domeinu bera nola inplementa daitekeen aztertuko da. Amaitzeko, objektuetara zuzendutako ikuspegia aukeratuko da aplikazioaren eraikuntza-prozesua gauzatzeko, hau da, informazio-sistemaren iraunkortasuna objektuak erabilita eraikiko da zuzenean, inongo mapaketa eta tramankulu barik.

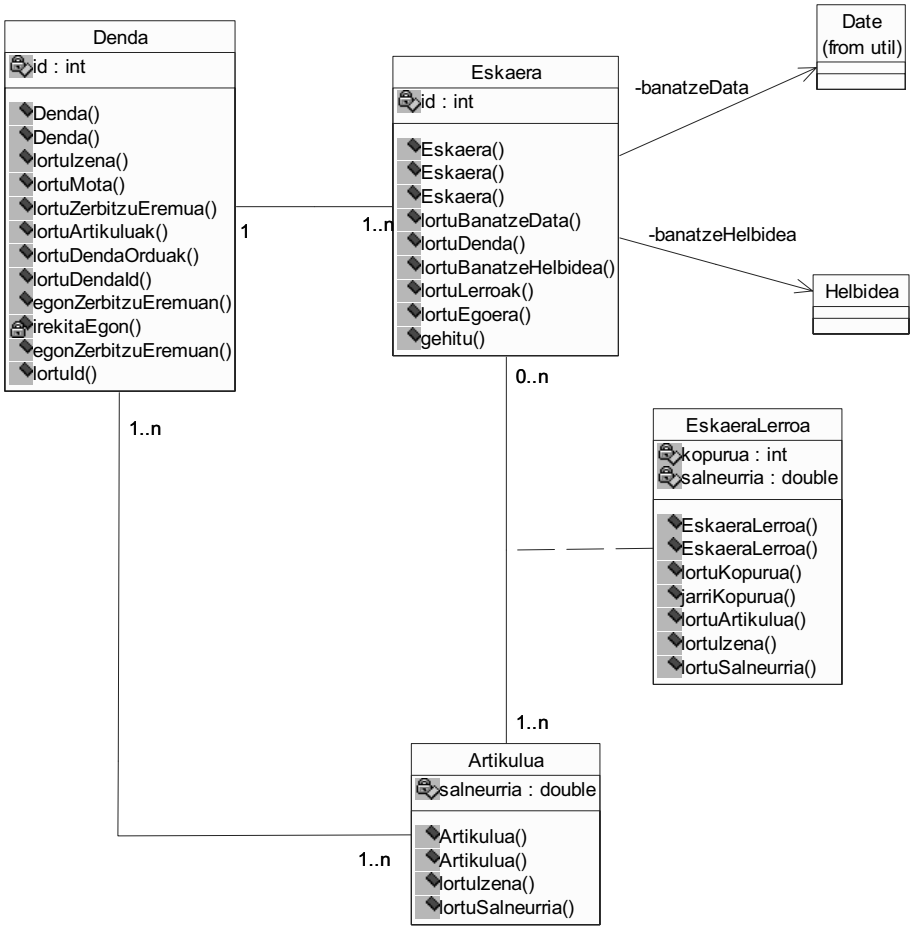
Beraz, has gaitezen ikuspegi guztientzat erabiliko diren domeinuaren betekizunak azaltzen. Aplikazioaren domeinuaren logikak enpresa baten eskaerak kudeatu nahi ditu eta horretarako bertako arduradunarekin egon gara eta honako informazio hau eman digu:

Eskaera egin nahi duen pertsonak enpresaren web gunea erabiliko du eskaera egiteko. Eskaera egiterakoan zein dendak kudeatuko duen eskaera aukeratzen du pertsonak. Hau da, eskaerak dendaren arabera antolatuko dira eta eskaera batek denda horretako produktuak eskatuko ditu. Prozesuaren muinean dagoena da enpresak artikuluak sailkatuta dituela dendetan, alegia, denda bakoitza artikulu mota batean espezializatuko da kostu orokorrak jaisteko eta eraginkortasuna bermatzeko.

Eskaera gauzatu ondoren, dendari bidaltzen dio sistemak. Banatze-data eta banatze-helbideak kontuan hartuta, eskaera tratatzen du dendak eta garraiolari bat aukeratzen du eskaeraren artikuluak banatzeko. Garraiolaria aukeratu ondoren, SMS mezu bat bidaltzen dio sistemak ohartarazteko noiz joan behar duen dendara eta zer jaso behar duen dendatik.

Sistemak adierazitako eguna heltzen denean, dendara joaten da garraiolaria artikulu bila eta adostutako banatze-helbidean uzten ditu artikuluak. Arazorik ez bada sortu garraioan, ontzat ematen du sistemak eta fakturazioa gauzatzen da.

4.1. irudian agertzen da interesatzen zaigun eta landu nahi den prozesuaren zatia egituratzeko erabiliko den Domeinuaren Eredua.



4.1. irudia. Adibidearen Domeinuaren Eredua.

4.2. IKUSPEGI PROZEDURAL AURRERATUA: TRANSACTION SCRIPT ETA IBATIS

Hasieran, aipatutako Domeinuaren Eredua inplementatzeko ikuspegi prozedurala erabiliko da, lan-taldeak ez baitu eskarmentu handirik softwarea garatzen objektuetara zuzendutako teknologiekin. Zorionez, domeinuaren logikak ez duenez buruhauste askorik ematen, ikuspegi hori egokia izan daiteke aplikazioa inplementatzeko. Gauzak zertxobait aldrebesteko, bezeroak azpimarratu du ezen datu-basera egingo diren atzipenak SQL aginduak erabilita gauzatuko direla. Horrek esan nahi du iraunkortasunaren aldea ezin izango dela landu iraunkortasuna kudeatzeko sortu berri diren tresna aurreratuekin, hala nola Hibernate edo JDO (Java Data Objects) estandarrarekin.

Azaldutako domeinuaren analisi eta diseinu osoa azaltzea ez da atal honen helburua, behin aztertuta nola gauzatzen den erabilera-kasu batentzat, behin eta berriz errepikatzen baita aplikatutako metodologia.

Gure helburua, aplikazioa eraikitzeko erabili den metodologia azaltzea da eta ez aplikazio osoaren inplementazioa, hori nahikoa bideratua gelditzen baita analisi eta diseinu on batekin. Beraz, nahikoa zaigu erabilera-kasu bat ondo aztertzea aipatutako ikuspegi prozeduralaren nondik norakoak azaleratzeko.

Aukeratu den erabilera-kasua honako hau da: `Eskaera Egin`. Erabilera-kasu hori ondo aztertzea garrantzitsua da garapen osoarentzat, berak sortuko baitu eskaera eta egituratuta utzi, gainerako erabilera-kasuei.

Erabilera-kasuak honako gertaera-fluxua izango du:

1. Eskaera egin nahi duen pertsonak banatze-data eta helbidea sartzen ditu sistema informatikora. Aplikazioak egiaztatu egiten du sartutako informazioa ondo dagoela eta gutxienez denda bat dagoela erakundean gauza dena eskaera gauzatzeko.

2. Behin-behineko eskaera eraikitzen hasten da sistema informatikoa duen informazioarekin eta dendez osatutako zerrenda bat aurkezten dio eskatzaileari aukera egin dezan.

3. Denda bat aukeratzen du eskatzaileak eta denda horretako artikuluak pantailaratzen dizkio sistemak.

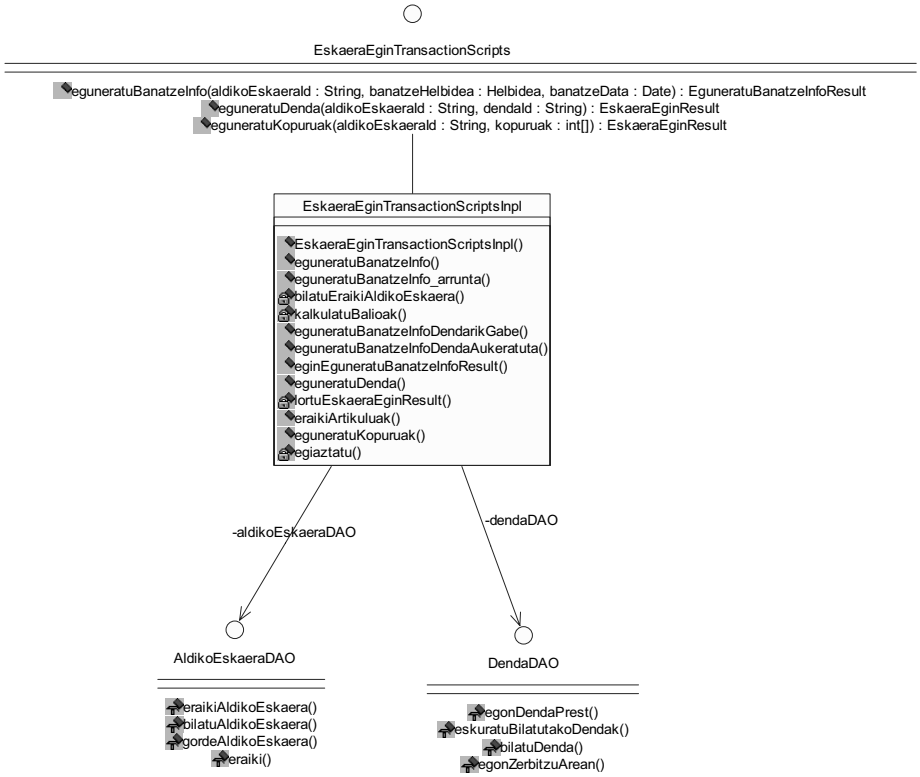
4. Interesatzen zaizkion artikuluak aukeratzen ditu erabiltzaileak adieraziz zenbat nahi dituen bakoitzetik.

5. Eskaerari amaiera emateko ordainketaren informazioa sartzen du erabiltzaileak sistemara.

6. Informazio guztia bildu eta egiaztatu egiten du sistemak eta eskaeraren informazioa pantailaratu egiten da.

7. Eskaeraren ñabardurak posta elektronikoz jasotzen ditu eskatzaileak eta nahi izanez gero, inprimatu egiten da eskaeraren informazioa.

Aztertu dugun erabilera-kasuaren domeinuaren logikak hainbat prozesu eta azpiprozesu ditu eta behin eta berriz jartzen da harremanetan datu-basearekin informazioa atzitzeko edota biltegiratzeko. Errazera jo dezakegu eta eragiketa horiek guztiak isolatuta gauzatu. Hala, aplikazioaren inplementazioa, azkar-azkar eraikiko da eta gero gerokoak. Estrategia horrek, baina, alde ilunak ditu berehala hasiko direnak agertzen.



4.2. irudia. Transaction Script diseinu-patroiaren arkitektura orokorra.

Arazoak ekiditeko, metodologia arautu bat erabiliko dugu erabilera-kasuak antolatzeko. Lehenengo eta behin, erabilera-kasuen domeinuaren logika kapsulatu egingo da Transaction Script[52] patroi ezaguna erabilita. Patroi hori erabilita, domeinuaren logika zatitu eta modularizatu egiten da transakzio-gidoiak erabilita. Aplikazioaren betekizunak ondo aztertu badira eta erabilera-kasu bakoitzaren portaera ondo zehaztu bada, transakzio-gidoi bat prestatzen da erabilera-kasuko. Guk estrategia bera erabiliko dugu azaldutako erabilera-kasua diseinatzeko (**4.2. irudia**) eta inplementatzeko. Irudiak argi adierazten du transakzio-gidoi baten atzean zer dagoen: transakzioak kudeatu eta kapsulatzeko baliabide egituratu bat.

Gidoiak kapsulatu egiten du transakzio bakoitzaren logika eta gidoiaren bezeroak ez du arduratu behar transakzioa egikaritzean nola gauzatzen diren eragiketa guztiak. Prozesua erabat kapsulatuta dago eta gardena da bezeroarentzat. Gidoiaren bezeroak eskaera bat egiten dio gidoiari transakzio eran, transakzioa egikaritzen du gidoiak eta emaitza bezeroari itzultzen dio txintik esan gabe. Lan isila eta eraginkorra aldi berean.

52. http://www.martinfowler.com/eaaCatalog/transactionScript.html
http://www.corej2eepatterns.com/Patterns2ndEd/BusinessObject.htm

4.1. irudiak islatzen du nola antolatzen duen datu-baseko informazioa transakzioaren gidoiak. Jakina, datu-basean dagoen informazioa tauletan eta tupletan dago egituratua, hau da, erabat xehatua eta banatua. Hala, gure erabilera-kasuaren erabiltzaileak eskatu duen eskaera ez dago taula bakarrean biltegiratuta, hainbatetan baizik. Zoritxarrez, gidoiaren bezeroak eta gidoiak berak ere, bere osotasunean erabili nahi du eskaera eta ez zatika behin eta berriz osotuz eta zatituz. Horretarako DAO[53] (Data Access Object) patroi txikiak erabiliko dira datu-basera egin behar ditugun atzipenak gauzatzeko. Guk, adibidez, gure erabilera-kasuarentzat delako bi DAO (`AldikoEskaeraDAO` eta `DendaDAO`) diseinatu ditugu.

Datu-basera atzipen bat egiten denean eskuratzen den informazioa erabat baldintzatua dago datu-basearen diseinura. Askotan, ez esatearren gehienetan, eskuratutako informazioa egituratu nahi izaten da eta egokitu, domeinuaren logikara. Adibidez, aztertzen ari garen erabilera-kasuan, erabiltzaileak egiten duen eskaera ez dator bat datu-baseko eskaera-taularekin. Exekuzio-garaian, memoria nagusian darabilgun eskaera, laster ikusiko dugun legez, askoz ere konplexuagoa da. Sinkronia falta hori konpontzeko egitura berezi batzuk diseinatu ditugu aplikazioaren Klase Diagraman (**4.3. irudia**). Softwarearen ingeniaritzan DTOak[54] (Data Transfer Object) erabilita eraikitzen dira egitura horiek. Geroago aztertuko ditugu batzuk, zehatz-mehatz. Oraingoz, nahikoa zaigu jakitea ezen egitura horiek, informazio-unitateak izango direla transakzio-gidoientzat eta lehen aurkeztu ditugun DAOentzat. Hala, informazioa era egituratuan mugituko da aplikazioan, informazio-unitate kontzeptual trinkoak osotuz.

*Gidoia*ren partaideak aurkeztuta, has gaitezen filmaren planoak grabatzen. Grabazioaren lehen egunean, erabilera-kasuaren transakzio-gidoiaren interfazea zehaztuko da:

```
import java.util.*;

public interface EskaeraEginTransactionScripts {

    public EguneratuBanatzeInfoResult eguneratuBanatzeInfo(
            String aldikoEskaeraId,
            Helbidea banatzeHelbidea, Date banatzeData);

    public EskaeraEginResult eguneratuDenda(
            String aldikoEskaeraId, String dendaId);

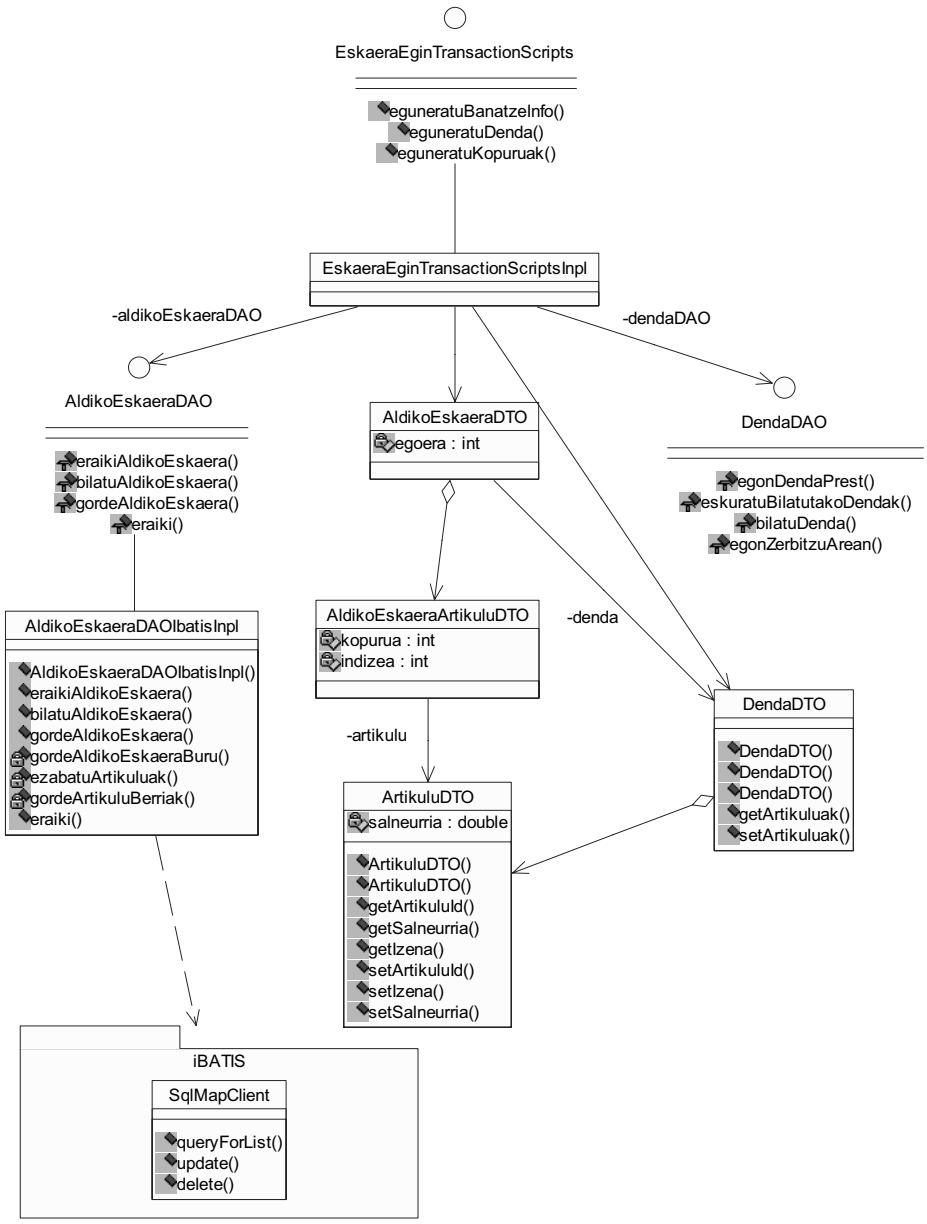
    public EskaeraEginResult eguneratuKopuruak(
            String aldikoEskaeraId, int[] kopuruak);

}
```

53. http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm
http://en.wikipedia.org/wiki/Data_Access_Object

54. http://www.martinfowler.com/eaaCatalog/dataTransferObject.html
http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html

Gidoiaren metodo bakoitza aztertzen bada poliki, argi adieraziko da metodoaren signaturan, metodo bakoitzak eskatuko duena egikaritzeko, eta itzuliko duena, behin egikarituta.



4.3. irudia. Aplikazioaren Klase Diagrama.

Eraikitzen ari garen softwarearen kalitatea bermatzeko, aipatutako interfazearen inplementazioa probatu egingo da era sistematikoan, probak bideratutako garapena[55] (*Test-driven development*) aplikatuz eraikitzen ari garen softwareari.

Metodologia horrekin proba egikaritzen denean eta objektu batek dei egiten dionean beste objektu bati oraindik ez dena inplementatu, honako hau gertatzen da: bere portaera simulatu egiten dela mock[56] objektu bat erabilita. Hala, posible da erakunde informatiko batean, aplikazioak zatika inplementatzea lan-taldeen produktibitatea hobetuz. Ikuspegi horrekin, zatika proba daitezke garapenaren arlo desberdinak eta ez da nahitaezkoa proba gauzatzen denean, zatiak behar dituen objektu guztiak inplementatuta edukitzea.

Probak, egoera zehatza planteatzen dio inplementazioari eta ondo joaten bada exekuzioa, lortzen diren emaitzak nahi zirenak izango dira. Adibidez, gure transakzio-gidoian, `eguneratuBanatzeInfo()` metodoa hainbat egoeratan egikaritu daiteke, hala nola aldiko eskaeraren identifikazioa `null` izan daiteke edo ez, banatze-informazioa ondo etor daiteke edo erroreak izan ditzake.

Demagun aipatutako metodoa probatu nahi dela egoera zehatz bat emanda, hau da, metodoari banatze-informazioa eta `null` bat pasatuko diogu identifikazio gisa. Hori guztia jakinda, proba bat antolatzen da azaldutako egoera probatzeko, mock objektuak erabilita:

```
import java.util.*;

import org.jmock.cglib.*;

public class EskaeraEginTransactionScriptsInplTests
        extends MockObjectTestCase {

    private Mock mockAldikoEskaeraDAO;

    private Mock mockDendaDAO;

    private AldikoEskaeraDAO aldikoEskaeraDAO;

    private DendaDAO dendaDAO;

    private EskaeraEginTransactionScriptsInpl zerbitzu;

    private Mock mockAldikoEskaera;

    private AldikoEskaeraDTO aldikoEskaera;

    public void setUp() throws Exception {
        super.setUp();
```

55. Kontu egin ezen garatze-era hori egilearen aurreko liburuan aztertzen zela JUnit tresna azaltzerakoan.

56. `http://www.mockobjects.com/`
`http://en.wikipedia.org/wiki/Mock_object`

```
        mockAldikoEskaeraDAO = new Mock(
                AldikoEskaeraDAO.class);
        mockDendaDAO = new Mock(
                DendaDAO.class);
        aldikoEskaeraDAO = (AldikoEskaeraDAO) mockAldikoEskaeraDAO
                .proxy();
        dendaDAO = (DendaDAO) mockDendaDAO
                .proxy();
        mockAldikoEskaera = new Mock(
                AldikoEskaeraDTO.class);
        aldikoEskaera = (AldikoEskaeraDTO) mockAldikoEskaera
                .proxy();
        zerbitzu = new EskaeraEginTransactionScriptsInpl(
                aldikoEskaeraDAO, dendaDAO);
    }

    public void testEguneratuBanatzeInfo_ondo()
            throws Exception {

        Helbidea banatzeHelbidea = new Helbidea();
        Calendar c = Calendar.getInstance();
        c.add(Calendar.HOUR, 2);
        Date banatzeData = c.getTime();
        List bilatutakoDendak = Collections
                .singletonList(new DendaDTO());

        mockAldikoEskaeraDAO.expects(once()).method(
                "eraikiAldikoEskaera").will(
                returnValue(aldikoEskaera));

        mockDendaDAO
                .expects(once())
                .method("eskuratuBilatutakoDendak")
                .with(eq(banatzeHelbidea),
                        eq(banatzeData))
                .will(
                        returnValue(bilatutakoDendak));

        mockAldikoEskaera.expects(once()).method(
                "setBanatzeHelbidea").with(
                eq(banatzeHelbidea));
        mockAldikoEskaera.expects(once()).method(
                "setBanatzeData").with(
                eq(banatzeData));
        mockAldikoEskaera
                .expects(once())
                .method("setEgoera")
                .with(
                        eq(AldikoEskaera.BANATZE_INFO_ADIERAZITA));
```

```
        mockAldikoEskaeraDAO.expects(once()).method(
                "gordeAldikoEskaera").with(
                eq(aldikoEskaera));

         EguneratuBanatzeInfoResult emaitza = zerbitzu
                .eguneratuBanatzeInfo_arrunta(null,
                        banatzeHelbidea, banatzeData);

         assertEquals(
                EguneratuBanatzeInfoResult.DENDA_AUKERATUTA,
                emaitza.getEgoeraGako());
         assertSame(aldikoEskaera, emaitza.getAldikoEskaera());
         assertSame(bilatutakoDendak,
emaitza.getBilatutakoDendak());
    }
}
```

Kontu egin ezen probak bi metodo egikaritzen dituela: `setUp()` eta `testEguneratuBanatzeInfo_ondo()`.

Lehenengoa (`setUp()`), ezaguna zaigu JUniten ere erabiltzen baitzen. Metodoak probaren hasieraketak gauzatzen ditu, mock objektuak eraikita eta gidoiaren inplementazioa sortuta.

Hasieraketak gauzatuta, proba egikaritzen da, aipatutako egoera simulatuta, `testEguneratuBanatzeInfo_ondo()` metodoan. Begi bistakoa da ezen probaren muina metodoaren azken aldean aurkitzen dela, hor egikaritzen baita gidoia. Azken lerroetan probatzen da, itzulitako emaitzak, espero genuen informazioa itzultzen duela.

Azpiegitura guztia prestatuta dugunez, gelditzen zaigun azken lana transakzio-gidoia inplementatzea da:

```
import java.util.*;


public class EskaeraEginTransactionScriptsInpl
        implements EskaeraEginTransactionScripts {

    private DendaDAO dendaDAO;

    private AldikoEskaeraDAO aldikoEskaeraDAO;

    public EskaeraEginTransactionScriptsInpl(
            AldikoEskaeraDAO aldikoEskaeraDAO,
            DendaDAO dendaDAO) {
        this.aldikoEskaeraDAO = aldikoEskaeraDAO;
        this.dendaDAO = dendaDAO;
    }
```

```
    public EguneratuBanatzeInfoResult eguneratuBanatzeInfo(
            String aldikoEskaeraId,
            Helbidea banatzeHelbidea, Date banatzeData) {

        AldikoEskaeraDTO aldikoEskaera =
             bilatuEraikiAldikoEskaera(aldikoEskaeraId);

        if (aldikoEskaera.getDenda() == null)
            return eguneratuBanatzeInfoDendarikGabe(
                    aldikoEskaera, banatzeHelbidea,
                    banatzeData);
        else
            return eguneratuBanatzeInfoDendaAukeratuta(
                    aldikoEskaera, banatzeHelbidea,
                    banatzeData);
    }

    public EguneratuBanatzeInfoResult eguneratuBanatzeInfo_arrunta(
            String aldikoEskaeraId,
            Helbidea banatzeHelbidea, Date banatzeData) {

        AldikoEskaeraDTO aldikoEskaera =
bilatuEraikiAldikoEskaera(aldikoEskaeraId);

        Calendar lehenBanatzeData = Calendar
                .getInstance();
        lehenBanatzeData.add(Calendar.HOUR, 1);

        if (banatzeData.before(lehenBanatzeData
                .getTime())) {
            AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
            return eginEguneratuBanatzeInfoResult(

EguneratuBanatzeInfoResult.BANATZE_INFO_EZ_EGOKIA,
                    aldikoEskaera, balioak, null);
        }

        List dendaZerrenda = dendaDAO
                .bilatuDendaZerrenda(
                        banatzeHelbidea, banatzeData);

        if (dendaZerrenda.isEmpty()) {
            AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
            return eginEguneratuBanatzeInfoResult(
                    EguneratuBanatzeInfoResult.EZ_DENDA_PREST,
                    aldikoEskaera, balioak, null);
        }
```

```
        aldikoEskaera
                .setBanatzeHelbidea(banatzeHelbidea);
        aldikoEskaera.setBanatzeData(banatzeData);
        aldikoEskaera
                .setEgoera(AldikoEskaera.BANATZE_INFO_EGINDA);

        aldikoEskaeraDAO.gordeAldikoEskaera(aldikoEskaera);

        return new EguneratuBanatzeInfoResult(
                EguneratuBanatzeInfoResult.DENDA_AUKERATUTA,
                aldikoEskaera, dendaZerrenda);
    }

    private AldikoEskaeraDTO bilatuEraikiAldikoEskaera(
            String aldikoEskaeraId) {
        if (aldikoEskaeraId == null)
            return aldikoEskaeraDAO
                    .eraikiAldikoEskaera();
        else
            return aldikoEskaeraDAO
                    .bilatuAldikoEskaera(aldikoEskaeraId);
    }

    private AldikoEskaeraBalioak kalkulatuBalioak(
            AldikoEskaeraDTO aldikoEskaera) {
        return new AldikoEskaeraBalioak();
    }

    public EguneratuBanatzeInfoResult
eguneratuBanatzeInfoDendarikGabe(
            AldikoEskaeraDTO aldikoEskaera,
            Helbidea banatzeHelbidea, Date banatzeData) {

        List dendaZerrenda = dendaDAO
                .bilatuDendaZerrenda(
                        banatzeHelbidea, banatzeData);

        if (dendaZerrenda.isEmpty()) {
            AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
            return eginEguneratuBanatzeInfoResult(
                    EguneratuBanatzeInfoResult.EZ_DENDA_PREST,
                    aldikoEskaera, balioak, null);
        }

        aldikoEskaera
                .setBanatzeHelbidea(banatzeHelbidea);
        aldikoEskaera.setBanatzeData(banatzeData);
        aldikoEskaera
                .setEgoera(AldikoEskaera.BANATZE_INFO_EGINDA);
```

```
        aldikoEskaeraDAO.gordeAldikoEskaera(aldikoEskaera);

        AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
        return eginEguneratuBanatzeInfoResult(
                EguneratuBanatzeInfoResult.DENDA_AUKERATUTA,
                aldikoEskaera, balioak,
                dendaZerrenda);
    }

    public EguneratuBanatzeInfoResult
eguneratuBanatzeInfoDendaAukeratuta(
            AldikoEskaeraDTO aldikoEskaera,
            Helbidea banatzeHelbidea, Date banatzeData) {
        AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
        if (dendaDAO.egonZerbitzuArean(aldikoEskaera
                .getDenda().getDendaId(),
                banatzeHelbidea, banatzeData)) {

            aldikoEskaera
                    .setBanatzeHelbidea(banatzeHelbidea);
            aldikoEskaera.setBanatzeData(banatzeData);


aldikoEskaera.setEgoera(AldikoEskaera.BANATZE_INFO_ADIERAZITA);

            aldikoEskaeraDAO
                    .gordeAldikoEskaera(aldikoEskaera);
            return eginEguneratuBanatzeInfoResult(
                    EguneratuBanatzeInfoResult.GORDE_DENDA,
                    aldikoEskaera, balioak, null);
        } else if (dendaDAO
                .egonDendaPrest(
                        banatzeHelbidea, banatzeData)) {
            return eginEguneratuBanatzeInfoResult(
                    EguneratuBanatzeInfoResult.BAIEZTATU_ALDAKETA,
                    aldikoEskaera, balioak, null);
        } else
            return eginEguneratuBanatzeInfoResult(
                    EguneratuBanatzeInfoResult.EZ_DENDA_PREST,
                    aldikoEskaera, balioak, null);

    }

    public EguneratuBanatzeInfoResult
eginEguneratuBanatzeInfoResult(
            int egoeraGako,
            AldikoEskaeraDTO aldikoEskaera,
            AldikoEskaeraBalioak balioak,
```

```
            List dendaZerrenda) {
        aldikoEskaera.setBalioak(balioak);
        return new EguneratuBanatzeInfoResult(
                egoeraGako, aldikoEskaera,
                dendaZerrenda);
    }

    public EskaeraEginResult eguneratuDenda(
            String aldikoEskaeraId, String dendaId) {
        AldikoEskaeraDTO aldikoEskaera =
bilatuEraikiAldikoEskaera(aldikoEskaeraId);

        DendaDTO denda = dendaDAO
                .bilatuDenda(dendaId);

        if (aldikoEskaera.getEgoera() == AldikoEskaera.BERRIA
                || aldikoEskaera.getEgoera() ==
AldikoEskaera.EGINDA)
            throw new RuntimeException("Egoera = "
                    + aldikoEskaera.getEgoera());

        AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
        if (dendaDAO.egonZerbitzuArean(denda
                .getDendaId(), aldikoEskaera
                .getBanatzeHelbidea(), aldikoEskaera
                .getBanatzeData())) {
            aldikoEskaera.setDenda(denda);
            aldikoEskaera.setArtikuluak(new ArrayList());
            aldikoEskaera
                    .setEgoera(AldikoEskaera.DENDA_AUKERATUTA);
            aldikoEskaeraDAO
                    .gordeAldikoEskaera(aldikoEskaera);
            return lortuEskaeraEginResult(
                    EskaeraEginResult.OK, aldikoEskaera,
                    balioak);
        } else {
            return lortuEskaeraEginResult(
                    EskaeraEginResult.BAZTERTU,
                    aldikoEskaera, balioak);
        }
    }

    private EskaeraEginResult lortuEskaeraEginResult(
            int egoeraGako,
            AldikoEskaeraDTO aldikoEskaera,
            AldikoEskaeraBalioak balioak) {
        aldikoEskaera.setBalioak(balioak);
        return new EskaeraEginResult(egoeraGako,
                aldikoEskaera);
```

```
    }

    public static void eraikiArtikuluak(
            AldikoEskaeraDTO aldikoEskaera,
            Collection artikuluZerrenda) {
        Iterator it = artikuluZerrenda.iterator();
        ArrayList ALartikuluak = new ArrayList();
        int ind = 0;
        while (it.hasNext()) {
            ArtikuluDTO art = (ArtikuluDTO) it.next();
            ALartikuluak
                    .add(new AldikoEskaeraArtikuluDTO(
                            ind++, art));
        }
        aldikoEskaera.setArtikuluak(ALartikuluak);
    }

    public EskaeraEginResult eguneratuKopuruak(
            String aldikoEskaeraId, int[] kopuruak) {
        AldikoEskaeraDTO aldikoEskaera =
bilatuEraikiAldikoEskaera(aldikoEskaeraId);

        if (aldikoEskaera.getEgoera() !=
AldikoEskaera.DENDA_AUKERATUTA
                && aldikoEskaera.getEgoera() !=
AldikoEskaera.PREST_BIDALTZEKO
                && aldikoEskaera.getEgoera() !=
AldikoEskaera.PREST_KOKATU)
            throw new RuntimeException();

        List artikuluakL = aldikoEskaera.getArtikuluakL();

        if (artikuluakL.size() != kopuruak.length)
            throw new RuntimeException(
                    "Ez dute neurri bera. Eduki behar: "
                            + artikuluakL.size()
                            + "beste hau du: "
                            + kopuruak.length);

        List artikuluZerrenda = aldikoEskaera.getDenda()
                .getArtikuluZerrenda();
        List ALartikuluak = new ArrayList();
        Iterator it = artikuluZerrenda.iterator();
        int ind = 0;
        for (int i = 0; i < kopuruak.length; i++) {
    ArtikuluDTO artikulua = (ArtikuluDTO) it.next();
            int kopurua = kopuruak[i];
            if (kopurua > 0)
                ALartikuluak
                        .add(new AldikoEskaeraArtikuluDTO(
```

```
                                ind++, kopurua,
                                artikulua));
        }
        aldikoEskaera.setArtikuluak(ALartikuluak);

        aldikoEskaera
                                        .setEgoera(egiaztatu()  ?
AldikoEskaera.PREST_BIDALTZEKO
                        : AldikoEskaera.DENDA_AUKERATUTA);

        aldikoEskaeraDAO.gordeAldikoEskaera(aldikoEskaera);

        AldikoEskaeraBalioak balioak =
kalkulatuBalioak(aldikoEskaera);
        return lortuEskaeraEginResult(
                EskaeraEginResult.ADOS, aldikoEskaera,
                balioak);
    }

    private boolean egiaztatu() {
        return true;
    }

}
```

Gidoiak hainbat metodo inplementatzen ditu baina guri interesatzen zaiguna `eguneratuBanatzeInfo_arrunta()` da.

Hasieran, aldiko eskaera bat eraikitzen du metodoak, `AldikoEskaeraDAO` objektuaren baliabideak erabilita. Gero, egiaztatu egiten da banatze-data etorkizunean kokatzen dela. Hori eginda, `DendaDAO` objektuaren baliabideak erabiltzen dira denden zerrenda egokia eskuratzeko. Itzulitako zerrenda hutsik badago, DTO bat itzultzen da `EZ_DENDA_PREST` egoera-kodearekin. Aldiz, zerrendak balioak baditu, hau da, dendak itzuli badira, beste DTO bat itzultzen du metodoak, `DENDA_AUKERATUTA` egoera-kodearekin. Balio hori itzuli baino lehen DTOan, beste bi lan gauzatzen ditu exekuzio-fluxuak: aldiko eskaeraren egoera eguneratu egiten da, `BANATZE_INFO_EGINDA` balioa esleituta aldiko eskaeraren egoera-kodean, eta biltegiratu egiten da aldiko eskaera `AldikoEskaeraDAO` objektuaren `gordeAldikoEskaera()` metodoa egikarituta.

Beste metodoen exekuzioak antzeko portaera duenez, ez gara luzatuko azalpenetan. Horra hor, zuretzat ariketa, irakurle.

Transakzio-gidoia diseinatzerakoan argi gelditu da gidoia diseinatzea eta inplementatzea ez dela bakarrik gidoiaren interfazea programatzea eta inplementatzea. Bidean, DTO objektu motak eta DAO objektu motak diseinatu eta inplementatuko dira, ikuspegi prozedurala ondo erabili eta txertatu nahi bada, erakunde informatikoan.

Transakzio-gidoien aurkezpenean objektu mota baten eta bestearen ezaugarriak aztertu ditugu kontzeptualki baina ez gara sartu xehetasunetan. Azaldu dugu zertarako erabiltzen diren, baina ez nola dauden egituratuta barrutik. Ondorengo orrialdeetan, aztertzen ari garen erabilera-kasuko aldiko eskaeraren arloan finkatuko dugu arreta eta harekin erlazionatuta dauden DAO eta DTO objektu motak aztertuko ditugu.

DTO objektu mota batek ordezkatzen ari den osagaiaren datuen informazioa izango du soilik eta hori izango da bere helburua: informazio hori atzigarri jartzea era egituratuenean atzitze-metodoak (`get()` eta `set()`) eskainiz informazioa eskuratzeko. Horrela, DTO bat aplikazioan zehar erabiltzen denean, ziurtatuta dago edozein bezero-osagaik beti izango duela eskuragarri osagaiaren datuen aldea.

Adibide gisa, hona hemen, transakzio-gidoian aldiko eskaerarentzat eraiki den DTOa:

```
import java.util.*;

public class AldikoEskaeraDTO {

    private String aldikoEskaeraId;

    private int egoera;

    private Helbidea banatzeHelbidea;

    private Date banatzeData;

    private DendaDTO denda;

    private List artikuluakL = new ArrayList();

    private AldikoEskaeraOsoa osoa;

    public AldikoEskaeraDTO() {
    }

    public AldikoEskaeraDTO(String aldikoEskaeraId, int egoera) {
        this.aldikoEskaeraId = aldikoEskaeraId;
        this.egoera = egoera;
    }

    public AldikoEskaeraDTO(String aldikoEskaeraId, int egoera,
            Helbidea banatzeHelbidea, Date banatzeData,
            DendaDTO denda) {
        this(aldikoEskaeraId, egoera);
        this.banatzeHelbidea = banatzeHelbidea;
        this.banatzeData = banatzeData;
        this.denda = denda;
    }
```

```
    public boolean equals(Object x) {
        if (!(x instanceof AldikoEskaeraDTO))
            return false;
        AldikoEskaeraDTO besteBat = (AldikoEskaeraDTO) x;
        return izanBerdin(aldikoEskaeraId,
besteBat.aldikoEskaeraId)
                && izanBerdin(banatzeHelbidea,
besteBat.banatzeHelbidea)
                && izanBerdin(banatzeData, besteBat.banatzeData)
                && izanBerdin(denda, besteBat.denda)
                && izanBerdin(artikuluakL, besteBat.artikuluakL);
    }

    private boolean izanBerdin(Object x, Object y) {
        return x == null || y == null ? x == y : x.equals(y);
    }

    public int hashCode() {

        return super.hashCode();
    }

    public String getAldikoEskaeraId() {
        return aldikoEskaeraId;
    }

    public DendaDTO getDenda() {
        return denda;
    }

    public int getEgoera() {
        return egoera;
    }

    public Helbidea getBanatzeHelbidea() {
        return banatzeHelbidea;
    }

    public Date getBanatzeData() {
        return banatzeData;
    }

    public OrdainketaInformazioa getOrdainketaInformazioa() {
        return null;
    }

    public List getArtikuluakL() {
        return artikuluakL;
    }
```

```
public void setBanatzeHelbidea(Helbidea banatzeHelbidea) {
    this.banatzeHelbidea = banatzeHelbidea;
}

public void setBanatzeData(Date banatzeData) {
    this.banatzeData = banatzeData;
}

public void setEgoera(int egoera) {
    this.egoera = egoera;
}

public void setDenda(DendaDTO denda) {
    this.denda = denda;
}

public void add(AldikoEskaeraLineItemDTO lineItem) {
    artikuluakL.add(lineItem);
}

public void setArtikuluakL(List newArtikuluakL) {
    this.artikuluakL = newArtikuluakL;
}


public void setAldikoEskaeraId(String aldikoEskaeraId) {
    this.aldikoEskaeraId = aldikoEskaeraId;
}

public String getDendaId() {
    return getDenda() != null ? getDenda().getDendaId()
            : null;
}

public void setOsoa(AldikoEskaeraOsoa osoa) {
    this.osoa = osoa;
}  }
```

Aztertzen ari garen transakzio-gidoiaren Klase Diagraman (**4.2.** eta **4.3 irudiak**) jartzen bada arreta, berehala konturatzen gara ezen datu-basearekin dituen harreman guztiak DAO objektu motak erabilita gauzatzen dituela transakzio-gidoiak. Hau da, datu-basera egingo diren atzipen guztien logika, objektu mota horietan kapsulatzen da. Diseinu horrekin transakzio-gidoiak ez du arduratu behar lehen azaldu ditugun DTOak eraikitzeaz eta antolatzeaz, hori DAO objektuaren lana izango da.

Aplikazioaren diseinatzaileak, beraz, lehen egin duen legez transakzio-gidoiarekin, hemen ere, osagaien DAO interfazeak diseinatu eta inplementatu beharko ditu. Gure kasuan, aldiko eskaerari prestatu diogun DAO interfazea honako hau da:

```
public interface AldikoEskaeraDAO {

    AldikoEskaeraDTO eraikiAldikoEskaera();

    AldikoEskaeraDTO bilatuAldikoEskaera(String aldikoEskaeraId);

    void gordeAldikoEskaera(AldikoEskaeraDTO aldikoEskaera);

    void eraiki(AldikoEskaeraDTO aldikoEskaeraBalioak);

}
```

Lehen egin den legez transakzio-gidoiarekin, hemen ere, metodologia bera erabiliko da DAO interfazea inplementatzeko, hau da, probak bideratutako garapenaren ikuspegia. Eraiki den DAO objektu motaren inplementazioa egokia dela ziurtatzeko, proba sorta diseinatuko dugu mock objektu motak erabilita. Proba horietan DAO osagaia egoera guztietan jarriko da bere portaera aztertzeko. Egoera bakoitza egikarituko da eta probak ziurtatu egin behar du egikaritutakoa eta espero genuen emaitza bat datozela.

Jakina, egoerak asko izan daitezke eta mota askotakoak, guk bi diseinatu ditugu eta horiek probatuko ditugu:

```
import java.util.*;

import org.jmock.cglib.*;
import org.jmock.core.*;
import org.springframework.orm.ibatis.*;

public class AldikoEskaeraDAOIbatisInplMockTests extends
        MockObjectTestCase {

    private Mock mockSqlMapClientTemplate;

    private AldikoEskaeraDAOIbatisInpl dao;

    private AldikoEskaeraDTO aldikoEskaera;

    protected void setUp() throws Exception {
        super.setUp();
        mockSqlMapClientTemplate = new Mock(
                SqlMapClientTemplate.class);
        SqlMapClientTemplate sqlMapClientTemplate =
(SqlMapClientTemplate) mockSqlMapClientTemplate
                .proxy();
        aldikoEskaera = new AldikoEskaeraDTO();

        dao = new AldikoEskaeraDAOIbatisInpl(sqlMapClientTemplate);
    }

   public void testBilatuEraikiAldikoEskaera_berria() {
        mockSqlMapClientTemplate.expects(once()).method("insert")
```

```
                .with(eq("txertatuAldikoEskaera"),
                        new MatchNewAldikoEskaera()).will(
                        returnValue(aldikoEskaera));
        AldikoEskaeraDTO emaitza = dao.eraikiAldikoEskaera();
        assertSame(aldikoEskaera, emaitza);
    }

    public void testBilatuEraikiAldikoEskaera_ezberria() {
        mockSqlMapClientTemplate.expects(once()).method(
                "queryForObject").with(eq("bilatuAldikoEskaera"),
                eq("10")).will(returnValue(aldikoEskaera));
        AldikoEskaeraDTO emaitza = dao.bilatuAldikoEskaera("10");
        assertSame(aldikoEskaera, emaitza);
    }
.............
}
```

Proba, betiko hasieraketak (`setUp()`) gauzatuta hasiko da eta, ondoren, bi proba txiki egikarituko ditu. Batekin, aldiko eskaera berri bat ondo txertatzen dela datu-basean probatuko da eta, bestearekin, `10` identifikadorearekin biltegiratuta dagoen aldiko eskaera eskuratzen eta egiaztatzen saiatuko da proba.

Proba prestatu ondoren, hirugarren urratsa gauzatuko da, aldiko eskaeraren DAO objektu mota inplementatuta:

```
import java.util.*;

import org.springframework.orm.ibatis.*;
import org.springframework.orm.ibatis.support.*;

public class AldikoEskaeraDAOIbatisInpl extends SqlMapClientDaoSupport
implements
              AldikoEskaeraDAO {

       public AldikoEskaeraDAOIbatisInpl(SqlMapClientTemplate
sqlMapClientTemplate) {
              setSqlMapClientTemplate(sqlMapClientTemplate);
       }

       public AldikoEskaeraDTO eraikiAldikoEskaera() {
              AldikoEskaeraDTO aldikoEskaera = new AldikoEskaeraDTO();
              getSqlMapClientTemplate().insert("txertatuAldikoEskaera",
aldikoEskaera);
              return aldikoEskaera;
       }

       public AldikoEskaeraDTO bilatuAldikoEskaera(String
aldikoEskaeraId) {
              AldikoEskaeraDTO aldikoEskaeraDTO = (AldikoEskaeraDTO)
getSqlMapClientTemplate()
```

```
                        .queryForObject("bilatuAldikoEskaera",
aldikoEskaeraId);
            return aldikoEskaeraDTO;
      }

      public void gordeAldikoEskaera(AldikoEskaeraDTO aldikoEskaera) {
            gordeAldikoEskaeraBuru(aldikoEskaera);
            ezabatuArtikuluak(aldikoEskaera);
            gordeArtikuluBerriak(aldikoEskaera);
      }

      private void gordeAldikoEskaeraBuru(AldikoEskaeraDTO
aldikoEskaera) {

getSqlMapClientTemplate().update("gordeAldikoEskaera", aldikoEskaera);
      }

      private void ezabatuArtikuluak(AldikoEskaeraDTO aldikoEskaera) {

getSqlMapClientTemplate().update("ezabatuArtikuluak", aldikoEskaera);

      }

      private void gordeArtikuluBerriak(final AldikoEskaeraDTO
aldikoEskaera) {
            final List lerroak = aldikoEskaera.getArtikuluak();
            if (lerroak == null || lerroak.isEmpty())
                  return;
            getSqlMapClientTemplate().execute(
                        new
TxertatuArtikuluakIBatisCallback(aldikoEskaera));
      }

      public void eraiki(AldikoEskaeraDTO aldikoEskaeraD) {
            String aldikoEskaeraId =
eraikiAldikoEskaera().getAldikoEskaeraId()
                        .toString();
            aldikoEskaeraD.setAldikoEskaeraId(aldikoEskaeraId);
            gordeAldikoEskaera(aldikoEskaeraD);
      }   }
```

Inplementazioaren gauzarik harrigarriena da ez dela inondik ere agertzen SQL galderarik. Nola izan daiteke? ez al zen DAO objektua harremanetan jartzen zena datu-basearekin? Non daude ResulSet objektuak? Erantzuna: datu-basera egingo diren atzipenak iBATIS teknologia erabilita gauzatuko dira eta ez JDBC erabilita, zuzenean. Hau da, betikoa softwarearen ingeniaritzan, prozesua sinplifikatu, abstrakzio maila igota. Teknikoki esanda, erregulartasunak eta lan mekanikoak automatizatu, balio erantsia itsatsiz softwarearen garapenaren arlo konplexuei. Asko eta merke programatu, edo, gutxi eta garesti, kalitatea erantsita.

Gainera, atzipenak, konexioak eta transakzioen kudeaketa sinplifikatzeko, Spring plataformaren baliabideak erabiliko ditu inplementazioak.

Hala, DAO objektu motak Spring plataformaren `SqlMapClientDaoSupportklasea` hedatuko du eta `SqlMapClientTemplate` klasea erabiliko du, SQL aginduak egikaritzeko. Prozesua martxan hasteko, metodo eraikitzaileak `SqlMapClientTemplate` motako objektu bat jasotzen du parametro gisa eta `SqlMapClientDaoSupportklasea` klasearen `setSqlMapClientTemplate()` metodoari deitzen dio aipatutako parametroa finkatzeko `SqlMapClientTemplate` bezala.

Inplementazioaren metodoek aipatutako Template-a `get()`[57] metodo bat erabilita eskuratuko dute behin eta berriz. Adibidez, `bilatuAldikoEskaera()` metodoak aldiko eskaeraren DTOa itzultzen du, iBATIS teknologiaren baliabideak erabilita. Gure kasuan, `Select` motako SQL galdera egikaritzen da, aldiko eskaera eskuratzeko. Aldiko eskaerak, uste genuen bezala, bere artikuluak, denda eta dendaren artikuluak izango ditu.

Hona hemen *mirakulua* gauzatzeko iBATIS teknologiak erabiltzen duen XML fitxategia:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap
    PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
    "http://www.ibatis.com/dtd/sql-map-2.dtd">

<sqlMap>

<insert id="txertatuAldikoEskaera"
    parameterClass="AldikoEskaeraDTO">
       INSERT INTO ALDIKO_ESKAERA(EGOERA,BERTSIOA) VALUES(0,0)
       <selectKey resultClass="java.lang.String"
keyProperty="aldikoEskaeraId" >
          CALL IDENTITY()
       </selectKey>
</insert>


<select id="bilatuAldikoEskaera" parameterClass="java.lang.String"
resultMap="AldikoEskaeraResultMap">
       select *
       from ALDIKO_ESKAERA o left outer join DENDA r on r.denda_id =
o.denda_id
       where
       o.aldiko_eskaera_id = #value#
</select>
```

57. Zehatz-mehatz, `getSqlMapClientTemplate()`.

```
<resultMap id="AldikoEskaeraResultMap"
       class="AldikoEskaeraDTO">
       <result property="aldikoEskaeraId" column="ALDIKO_ESKAERA_ID"/>
       <result property="egoera" column="EGOERA"/>
       <result property="banatzeData" column="BANATZE_DATA"/>

       <result property="artikuluak" column="ALDIKO_ESKAERA_ID "
select="bilatuArtikuluakL"/>
       <result property="denda.dendaId" column="DENDA_ID"/>
       <result property="denda.izena" column="IZENA"/>
       <result property="denda.artikuluak" column="DENDA_ID"
select="bilatuArtikuluak"/>
</resultMap>

<resultMap id="AldikoEskaeraArtikuluResultMap"
       class="AldikoEskaeraArtikuluDTO">
       <result property="kopurua" column="KOPURUA"/>
       <result property="indizea" column="ART_INDIZE"/>
       <result property="artikulu.artikuluId" column="ARTIKULU_ID"/>
       <result property="artikulu.izena" column="IZENA"/>
       <result property="artikulu.salneurria" column="SALNEURRIA"/>
</resultMap>

<select id="bilatuArtikuluakL" parameterClass="java.lang.String"
resultMap="AldikoEskaeraArtikuluResultMap">
       select *
       from ALDIKO_ESKAERA_ARTIKULU lerro, ARTIKULU art
       where
       lerro.aldiko_eskaera_id = #value:NUMERIC#
       AND art.artikulu_id = lerro.artikulu_id
</select>


<select id="bilatuArtikuluak" parameterClass="java.lang.String"
resultMap=" ArtikuluakResultMap ">
       select *
       from ARTIKULU art
       where art.denda_id = #value:NUMERIC#
</select>

<resultMap id="ArtikuluakResultMap"
       class="ArtikuluDTO">
       <result property="artikuluId" column="ARTIKULU_ID"/>
       <result property="izena" column="IZENA"/>
       <result property="salneurria" column="SALNEURRIA"/>
</resultMap>

</sqlMap>
```

XML agiriak hainbat etiketa ditu, baina guri, `insert` eta `select`-ekin adierazita daudenak interesatzen zaizkigu, hor gauzatzen baitira SQL aginduak. Har dezagun bat: `bilatuAldikoEskaera`.

Aukeratu dugun zati hori, mapaketa gauzatzeko agindua da. Barruan dagoen SQL agindua aztertzen bada, berehala konturatuko gara ezen `String` motako argumentu bat eskatzen duela aginduak eta, egikaritu ondoren, aldiko eskaeraren informazioa eskuratzen duela esleitutako denda barne. Emaitza eraikitzeko eta nahi dugun DTOekin lotzeko, `resultMap` (`AldikoEskaeraResultMap`) erako egitura eraikitzen da.

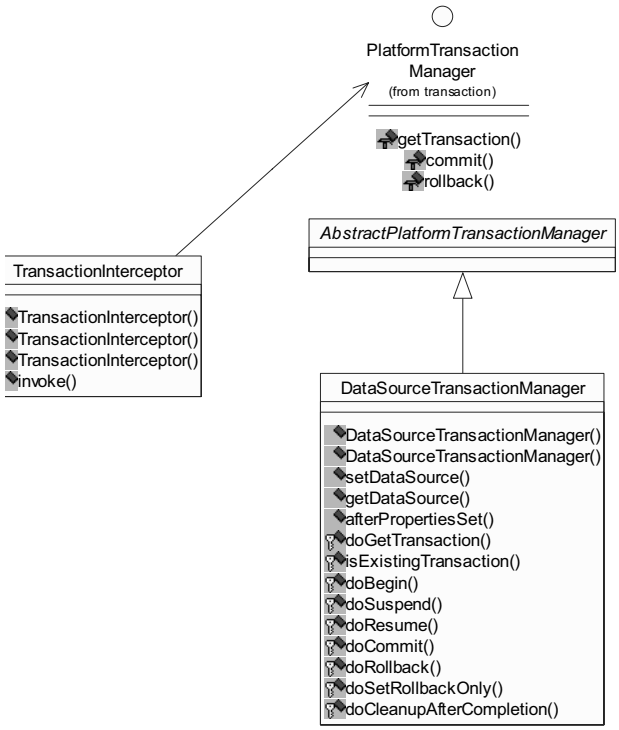
`AldikoEskaeraResultMap` egiturak lan interesgarriena egiten du, hau da, berak adierazten du nola gauzatuko den mapaketa datu-baseko informazioaren eta DTOko informazioaren artean. Kontu egin mapaketa gauzatzeko, agindu habiaratuak (`bilatuArtikuluakL` eta `bilatuArtikuluak`) erabilita, eskuratzen direla aldiko eskaeraren artikuluak eta dendaren artikuluak.

Horra hor, bada, objektuen eta datu-baseko erlazioen eta taulen arteko mapaketak gauzatzeko eta datu-basera egingo diren atzipenak kudeatzeko iBATIS teknologiak behar duen informazioaren azalpena. Azken XML fitxategi horrek argi adierazten dio aplikazioari nola interpretatu behar duen aplikazioan era automatikoan datu-basetik eskuratzen den eta datu-basera bidaltzen den informazioa. Mapaketa gauzatzeko, egikaritzen den SQL aginduaren informazioa adierazten du agindu bakoitzak, eta batez ere, nola eta aplikazioaren Klase Diagramako norekin gauzatu behar duen mapaketa. Hala, era malguan eta gardenean egikarituko dira mapaketa gauzatzeko aginduak aplikazio nagusian eta ez da inongo kateaketarik programatuko, iBATISek gauzatuko du azpiprozesua eta ez guk, JDBC erabilita.

Amaierara heltzen gaude, programatu ditugu transakzio-gidoia, aplikazioak behar dituen DAOak eta DTOak eta iBATIS teknologiak behar duen informazioa. Gelditzen zaiguna da datu-basera egingo diren transakzioak kudeatzea eta JDBC konexioak ondo antolatzea. Spring plataformak baliabide egokiak ditu aipatutako eragiketak gauzatzeko, beraz, bere esku geldituko da lan hori. **4.4. irudia**k azaltzen digu nola egituratzen diren plataformak beharko dituen baliabideak.

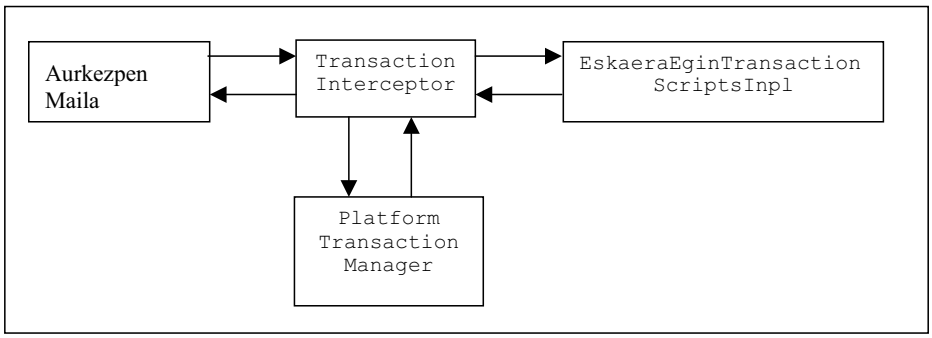
Lehenengo gauza ondo ulertu behar dena da Spring plataformak ez duela zuzen-zuzenean eta bat-batean transakzioa gauzatzen, hau da, EskaeraEginTransaction Scripts transakzio-gidoiak transakzio bat hasten duenean ez du zuzen-zuzenean egikarituko, Springek detektatuko du deia eta maskaratu egingo du objektu harrapari bat erabilita (`TransactionInterceptor`). Objektu hori, proxy bat izango da aplikazioarentzat eta berak jakin gabe, hainbat kode egikarituko da eskaera transakzionala bihurtzeko. Kontu egin ezen kode hori Springek egikarituko duela eta gardena dela aplikazio nagusiarentzat. Hala, transakzio lokaletarako, hau da, datubase bakarrera atzitzen denean aplikazioa, **4.4. irudia**n agertzen diren Spring plataformaren klaseak eta interfazeak erabiliko dira. Hona hemen prozesuaren urratsak:

1. Lehen esan dugun legez, gidoiak hasten du prozesua transakzio bat eskatuta.
2. Transakzioa egikaritu baino lehen, `TransactionInterceptor` klaseak harrapatu egiten du deia.
3. `TransactionInterceptor` klaseak transakzioa hasten du `PlatformTransactionManager` motako objektu bati deituta. Objektu mota horrek Strategy patroiaren diseinua du eta `TransactionInterceptor` motako objektuak erabiltzen du, transakzioak hasi, amaitu (*commit*) eta deuseztatzeko (*roll back*).
4. Datu-basera egingo diren atzipenak JDBC erabilita egingo direnez, `DataSourceTransactionManager` motako objektua erabiltzen da transakzioa kudeatzeko JDBCren `Connection` interfazea erabilita.
5. Transakzio-harrapariak, hau da, `TransactionInterceptor` motako objektuak, gidoian inplementatutakoa egikaritzeko agintzen du.
6. Emaitza jasotzen da.
7. `TransactionInterceptor` motako objektuak bukatutzat ematen du transakzioa `PlatformTransactionManager` motako objektuari dei eginda, commit-a gauzatzeko.



4.4. irudia. Spring plataformaren baliabideak adibidea egikaritzeko.

Horra hor, bada, Spring plataformaren lana, transakzioa detektatzeko, maskaratzeko eta kudeatzeko. Azaldutako azpiegitura guztia, aztertzen ari garen transakzio-gidoian kokatuko bagenu, **4.5. irudia**n agertzen den exekuzio-fluxua egikarituko litzateke.



4.5. irudia. Spring plataformaren exekuzio-fluxua.

Kontu egin ezen kateatu behar duguna honako hau dela: aplikazioaren klaseak, Spring plataformaren klaseak, iBATIS teknologiaren klaseak eta JDBCren klaseak. Bakoitzak, jakina, bere lana txukun bete beharko du: aplikazioaren klaseek domeinuaren logika, Spring plataformaren klaseek transakzioak, iBATIS teknologiaren klaseek objektuen eta taulen arteko mapaketa eta JDBCren klaseek datu-basearekin sortuko diren konexioak. Nola? **4.6. irudia**n agertzen den bezala.

Irudiaren azalpenaren nondik norakoak errazteko, Spring plataformak erabiliko duen XML fitxategiaren edukia azalduko dugu:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

   <bean id="SqlMapClient"
       class="org.springframework.orm.iBatis.SqlMapClientFactoryBean">
<property name="configLocation" value="/sqlMap-config.xml" />
           <property name="dataSource" ref="DataSource" />
       </bean>

       <bean id="SqlMapClientTemplate"

           class="org.springframework.orm.iBatis.SqlMapClientTemplate">
              <property name="exceptionTranslator"
ref="ExceptionTranslator" />
              <property name="sqlMapClient" ref="SqlMapClient" />
       </bean>
```

```
        <bean id="AldikoEskaeraDAO" lazy-init="true"
              class="AldikoEskareaDAOIbatisInpl">
              <constructor-arg ref="SqlMapClientTemplate" />
        </bean>

        <bean id="DendaDAO" lazy-init="true"
              class="DendaDAOIbatisInpl">
              <constructor-arg ref="SqlMapClientTemplate" />
        </bean>

    <bean id="EskaeraEginTransactionScripts" lazy-init="true"
        class="EskaeraEginTransactionScriptsInpl">
        <constructor-arg ref="AldikoEskaeraDAO" />
        <constructor-arg ref="DendaDAO" />
    </bean>

    <bean id="DataSourceTransactionInterceptor"

class="org.springframework.transaction.interceptor.TransactionIntercep
tor">
        <property name="transactionManager"
            ref="DataSourceTransactionManager" />
        <property name="transactionAttributeSource">
            <ref bean="nolaMetodoak" />
        </property>
    </bean>

    <bean id="nolaMetodoak"

class="org.springframework.transaction.interceptor.MatchAlwaysTransact
ionAttributeSource">
        <property name="transactionAttribute">
            <value>PROPAGATION_REQUIRED</value>
        </property>
    </bean>

    <bean id="EskaeraEginTransactionScriptsProxyCreator"

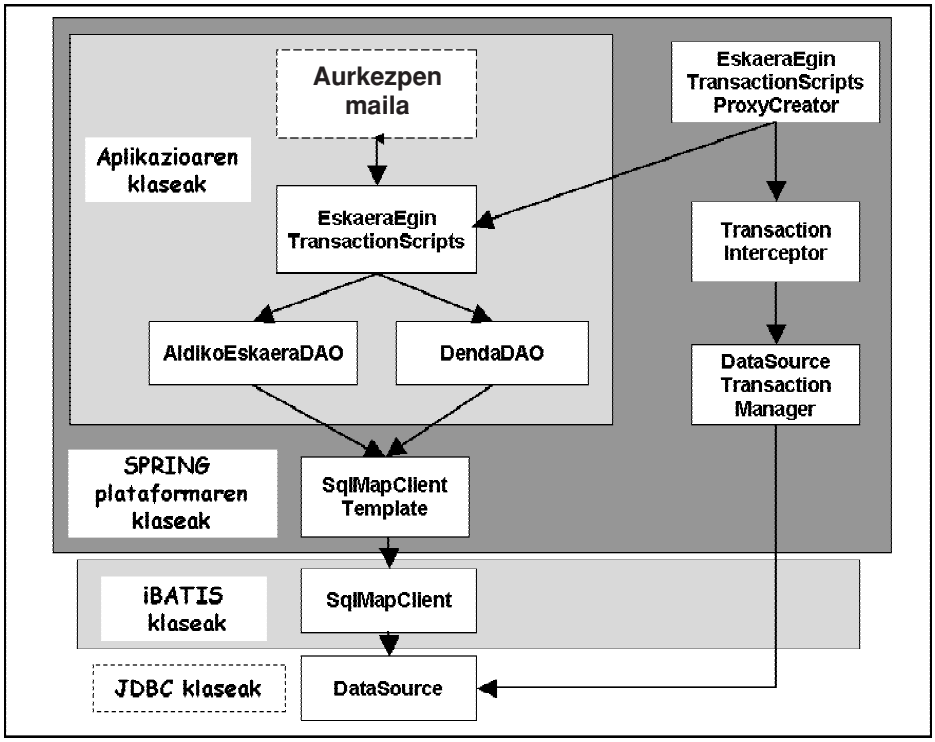
class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCr
eator">
        <property name="beanNames">
            <list>
                <idref bean="EskaeraEginTransactionScripts" />
            </list>
        </property>
        <property name="interceptorNames">
            <list>
                <idref bean="DataSourceTransactionInterceptor" />
```

```
            </list>
        </property>
    </bean>

    <bean id="DataSourceTransactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionMan
ager">
        <property name="dataSource" ref="DataSource" />
    </bean>

</beans>
```



4.6. irudia. Adibideko teknologien arteko harremanak.

Fitxategian agertzen diren Spring plataformaren bean-definizioak hiru multzotan sailkatzen dira:

1. iBATIS teknologia eta Spring plataforma harremanetan jartzeko beanak.
2. Aplikazioan erabiliko diren DAOen eta transakzio-gidoiaren inplementazioak eraikiko dituzten beanak.
3. Amaieran adieraziko dira datu-basearekin gauzatuko diren konexioak eta aplikazioak sortuko dituen transakzioak kudeatzeko definizioak.

Zehatz-mehatz, lehenengo bean-definizioarekin (`id="SqlMapClient"`) honako hau gauzatzen da: iBATIS teknologiaren `SqlMapClient` osagaia eraikitzen da iBATISentzat sortu ditugun fitxategi deskriptoreak erabilita. Fitxategi horien erreferentziak, `sqlMap-config` XML fitxategian aurkitzen dira:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMapConfig
PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
"http://www.ibatis.com/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
  <sqlMap resource="AldikoEskaera.xml" />
  <sqlMap resource="Denda.xml" />
</sqlMapConfig>
```

Spring plataformaren `SqlMapClientTemplate` osagaia eraikitzeko balio izango du bigarren bean-definizioak (`id="SqlMapClientTemplate"`).

Transakzio-gidoiak erabiliko dituen DAOen inplementazioak eraikitzeko balio izango dute hirugarren eta laugarren bean-definizioek (`id="AldikoEskaeraDAO"` eta `id="DendaDAO"`) eta, aldi berean, `SqlMapClientTemplate` osagaiarekin kateatzeko.

Aztertzen ari garen transakzio-gidoiaren inplementazioa eraikitzeko balio izango du bosgarren bean-definizioak (`id="EskaeraEginTransactionScripts"`). Kontu egin, eraikitze-fase horretan, goian aipatu ditugun bi DAOak injektatzen direla.

Seigarren bean-definizioa (`id="DataSourceTransactionInterceptor"`) da garrantzitsuena, hor konfiguratzen baita `TransactionInterceptor` klase transakzio-harraparia. Kontu egin ezen objektu horrek (**4.5. irudia**) bideratuko dituela gidoiaren deiak `PlatformTransactionManager` motako osagaira. Horrela, deia, transakzionala bihurtuko da.

Seigarren bean-definizioak lan erdia egin du; beste erdia, zortzigarren bean-definizioak (`id=""EskaeraEginTransactionScriptsProxyCreator`) egingo du. **4.6. irudia**n ikusten den legez, eta definizioak adierazten duen modura, bean-definizio horrekin, transakzio-gidoiaren inplementazioa eta objektu harraparia (`TransactionInterceptor`) kateatuta gelditzen dira. Kateaketa gauzatzeko, Spring plataformaren `BeanNameAutoProyCreator` osagaia erabiltzen da.

4.3. IRAUNKORTASUNAREN TRATAERA AURRERATUA: HIBERNATE

Azken urteotan, softwarearen eraikuntzaren bilakaeran, objektuetara zuzendutako ikuspegia da nagusi. Nagusitze hori, bestalde, ez da txiripaz eskuratutako zerbait, sakon-sakonean dagoena da teknologiaren bilakaerak berak sortu duela nagusitasun hori.

Softwarearen konplexutasunak softwarea modularizatzera eramaten du programatzailea. Horrela, konplexutasunaren arazoa leundu egiten da eta programak, irakurgarriagoak eta egituratuagoak bihurtzen dira, azken batean, erabilgarriagoak eta kontrolatuagoak programatzailearentzat. Programak modularizatu ondoren, objektuetara zuzendutako paradigmara gelditzen den jauzia, oso txikia eta naturala da.

Ikuspegi horrek, baina, buruhausteak sortzen ditu aplikazioaren informazio-sistemaren iraunkortasunarekin harremanetan jartzen denean.

Informazioa gaur egun, eta etorkizun hurbilean ere hala izango dela dirudi, datu-baseetan biltegiratzen da. Jakina, datu-baseak erabiltzeak informazioa biltegiratzeko, taulak, tuplak eta zutabeak erabiltzea dakar, hor ez dago ez objekturik, ez herentziarik, ez mezurik. Desdoitze-arazo[58] bat gertatzen da bi mundu horien artean. Batean, Eredu Erlazionala da nagusi eta, bestean, objektuetara zuzendutako programazioa.

Oraintsu arte desdoitze-arazo hori programatzailearen bizkarrean kokatzen zen behin eta berriz, Sisiforen harria legez. Aurreko orrietan azaldu dugunez iBATIS teknologia erabilita, arazoa zertxobait leuntzen zen baina, hala ere, programatzailearen esku gelditzen zen desdoitze hori konpontzea.

Aplikazioaren domeinua txikia denean, JDBC zuzenean erabilita edota iBATIS teknologia, nahikoa izan daitezke aplikazioaren inplementazioa eraikitzeko. Aldiz, hainbat objekturekin ari garenean memorian edota aplikazioaren domeinua konplexua denean, beste irtenbide bat bilatu behar dugu nahitaez.

Irtenbide horren estrategiak betikoa egin beharko luke, abstrakzio maila igo eta lan mekanikoa automatizatu. Horrela, domeinuaren logikan jar dezake arreta programatzaileak, eta desdoitze-arazoak tresna informatiko aurreratu bati utzi, hark konpon ditzan objektuen eta taulen arteko elkarrizketan sortzen diren arazoak.

Aipatutako tresna informatiko aurreratuari eskatuko dizkiogun ezaugarriak honako hauek izango dira:

- Objektuen eta datu-basearen eskemen arteko mapaketa erabat deskribatzailea izatea.
- API bat edukitzea objektuak eraiki, irakurri, eguneratu eta ezabatzeko.
- Galderetarako lengoaia edukitzea bilaketa aurreratuak eta egokituak gauzatzeko.
- Transakzioak ondo kudeatzea, hau da, integritatea zaintzea eta eguneraketa konkurrenteak gauzatzea.

58. Ingelesez, delako desdoitzeari, *Object-Relational Impedance Mismatch* deitzen zaio. http://en.wikipedia.org/wiki/Object-Relational_impedance_mismatch

- Objektuak memoriara ekartzeko baliabide egokiak edukitzea.
- Tresnak (*caching*) edukitzea datu-basera egingo diren atzipenak minimizatzeko.

Ezaugarri horiek betetzen dituzten plataformei ORM (Object/Relational Mapping) lan-markoak deituko diegu eta liburu honetan bi aztertuko ditugu: Hibernate[59] eta JDO[60] (Java Data Object) estandarra.

Hasteko, biek ala biek, XML agiriak erabilita gauzatzen dute O/R mapaketa eta ez dute DDL (Data Definition Language) fitxategirik erabili behar datubasearen eskema definitzeko.

Objektuak eraikitzeko, irakurtzeko, eguneratzeko eta ezabatzeko APIak oso antzekoak dira bi lan-markoetan. **4.7. taula**n azaltzen dira lan-marko bakoitzak eskaintzen dituen interfazeak. Lehenengo lerrokoek, konexioak eraikitzeko balio dute. Bigarren lerroan daudenek, objektuak eraikitzeko, kargatzeko eta ezabatzeko balioko dute. Hirugarren lerroko interfazeekin, galderak gauzatu daitezke eta azken lerrokoekin, transakzioak kudeatuko dira.

Eginkizuna	JDO lan-markoa	Hibernate lan-markoa
Konexio-faktoria eraiki	`PersistenceManagerFactory`	`SessionFactory`
Konexioak kudeatu	`PersistenceManager`	`Session`
Galderak antolatu	`Query`	`Query`
Transakzioak kudeatu	`Transaction`	`Transaction`

4.7. taula. JDO eta Hibernate lan-markoen interfaze nagusiak.

4.8. taulak jasotzen du zeintzuk diren `Session` eta `PersistenceManager` interfazeen metodo nagusiak.

Eginkizuna	Hibernate `Session`	JDO `PersistenceManager`
Objektua biltegiratu	`save()`	`makePersistent()`
Objektua atzitu	`load()`	`getObjectById()`
Objektua ezabatu	`delete`	`deletePersistent`
Galdera eraiki	`createQuery()` `createNamedQuery()`	`newQuery()` `newNamedQuery()`
Atzitu objektu transakzionala	`beginTransaction()`	`currentTransaction()`
Konexioa itxi	`close()`	`close()`

4.8. taula. `Session` eta `PersistenceManager` interfazeen metodo nagusiak.

59. `http://www.hibernate.org/`

60. `http://java.sun.com/products/jdo/`
JDOz ari garenean, jakina, JDO estandarra inplementatzen duen baliabideaz ari gara.

Biek ala biek, baliabideak eskaintzen dituzte aipatutako hirugarren, laugarren, bosgarren eta seigarren multzoko ezaugarriak inplementatzeko.

Programazio-lengoaiekin gertatzen den bezala, ORM lan-markoekin antzekoa gertatzen da: behin paradigma finkatuta eta adostuta lan-marko estandar batek eduki behar dituen ezaugarriak, ez du zentzu askorik bi edo hiru lan-marko aztertzeak aldi berean. Bat ondo aztertuta nahikoa da. Beraz, txanpona bota airera eta... Hibernate atera da.

Orain arte egin den legez eta ikuspegi prozedurala lekuko, datozen orrialdeetan ez da Hibernate lan-markoaren eskuliburua idatziko. Hori eginda[61] dago eta ez gara gurpila asmatzen hasiko. Horren ordez, probak bideratutako garapena erabilita eta hasieran azaldu dugun domeinua hartuta, inplementazio berri bat eraikitzen saiatuko gara, Hibernate lan-marko berria erabilita. Horrela, neurtu eta azaldu ahal izango dugu non kokatzen den Hibernate lan-markoa, testuinguru berri horretan. Hau da, honako galdera hauei erantzun nahi zaie: Zein da Hibernateren balio erantsia iBATIS teknologia baztertzeko? Noiz erabili JDBC huts-hutsik eta noiz Hibernate? Zenbat falta zaio Hibernateri objektuetara zuzendutako datu-basea izateko?

Ikuspegi prozedurala aztertzen ari ginela, behin eta berriz, probak bideratutako garapenaren teknikak erabili dira transakzio-gidoiaren portaera egituratzeko eta DAOen exekuzioa egiaztatzeko. Eraikitzen ari ginen softwarearen kalitatea ona izango zela bermatuta gelditzen zen.

Eskatutako probak ondo egikaritzen bazituen softwareak, ziur geunden, egoera horietan behintzat, ondo erantzungo zuela aplikazioak. Hala, egoera kritikoak aurretik simula daitezke *ad hoc* erako probak diseinatuta eta aplikazioari eskatuta egoera horietan egikaritzeko.

Lan egiteko era horrek aurreikusi egiten ditu egoerak bezeroari aplikazioa entregatu baino lehen, eta ez denez batere intrusiboa aplikazioarentzat probak diseinatzea, aplikazioaren iturburu-kodea kapsulatuta eta *garbi* mantentzen da une oro prozesu osoan.

xxUnit[62] familiako tresnek baliabideak eskaintzen dizkiote programatzaileari eraikitzen ari den softwarean aipatutako probak diseinatu eta egikaritzeko. Adibidez, DbUnit-ek edota ORMUnit-ek, aplikazioaren iraunkortasunaren geruzan finkatzen dute arreta, aztertzeko eta egiaztatzeko txertaketak, eguneraketak eta

61. Java Persistence with Hibernate. Manning, 2006.
http://www.manning.com/bauer2/

62. http://www.junit.org
http://www.dbunit.org/
http://xmlunit.sourceforge.net/
http://www.nunit.org/

ezabaketak ondo gauzatzen direla edota, galderak ondo egikaritzen direla, eta, emaitza egokiak itzultzen dituztela.

Halaber, behar-beharrezkoa izango da egiaztatzea ezen taulen eta objektuen artean gertatzen den mapaketa ondo diseinatuta dagoela, ez bakarrik taula mailan, baita zutabe mailan ere. Adibide gisa, hona hemen proba bat, ORMUnit tresnaren `HibernateSchemaTests` klasea erabilita gauzatzen dena:

```
public class GureHibernateSchemaTests
        extends HibernateSchemaTests {

    @Override
    protected String[] getConfigLocations() {
        return new String[] {};

    }

    @Override
    protected String getParentFactoryKey() {
        return "spring-context.xml";
    }

    @Override
    protected String getParentSelector() {
        return "hibernate-domain-model-ref-context.xml";
    }

    public void test() throws Exception {
        assertDatabaseSchema();
    }  }
```

`assertDatabaseSchema()` metodoa egikarituta, probak egiaztatzen du datu-basearen eskema bat datorrela O/R mapaketarekin. Probarekin datu-basearen taulak eta zutabeak aztertzen dira akatsak atzemateko.

Demagun diseinatu nahi den proba, azaldu dena baino zehatzagoa izatea nahi dela, hau da, osagai zehatz baten egoera aztertu nahi da diseinatu den O/R mapaketa ondo gauzatzen dela egiaztatzeko. Adibidez, egiaztatu nahi bada `AldikoEskaera` eta `Denda` klaseak ondo erlazionatuta daudela `ALDIKO_ESKAERA` eta `DENDA` taulekin, honako iturburu-kode hau egikaritu beharko genuke ORMUnit tresnaren `HibernateMappingTests` klasea hedatuta:

```
import java.sql.*;
import java.util.*;

import org.hibernate.*;

public class GureHibernateMappingTests extends
HibernateMappingTests {
```

```
       @Override
       protected String[] getConfigLocations() {
            return HibernateTestConstants.HIBERNATE_DOMAIN_CONTEXT;

       }

       public void testAldikoEskaeraMapping() throws SQLException,
                    HibernateException {

            assertClassMapping(AldikoEskaera.class, "ALDIKO_ESKAERA");
            assertIdField("id", "ALDIKO_ESKAERA_ID");
            assertVersionField("bertsioa", "BERTSIOA");
            assertField("egoera", "EGOERA");

            assertManyToOneField("denda", "DENDA_ID");

            assertComponentField("banatzeHelbidea");
            ComponentFieldMapping banatzeHelbidea =
getComponentFieldMapping("banatzeHelbidea");

       banatzeHelbidea.assertNonPersistentFields(Collections.EMPTY_SET);

            assertCompositeListField("artikuluak");
            CompositeListFieldMapping artikuluak =
getCompositeListFieldMapping("artikuluak");
            artikuluak.assertTable("ALDIKO_ESKAERA_ARTIKULU");
            artikuluak.assertForeignKey("ALDIKO_ESKAERA_ID");
            artikuluak.assertIndexColumn("ART_INDIZE");
            artikuluak.assertField("kopurua", "KOPURUA");
            artikuluak.assertManyToOneField("artikulu", "ARTIKULU_ID");

       artikuluak.assertNonPersistentFields(Collections.singleton("id"));

              assertAllFieldsMappedExcept("artikuluak.id");
       }

       public void testDendaMapping() throws SQLException,
HibernateException {
            assertClassMapping(Denda.class, "DENDA");
            assertIdField("id", "DENDA_ID");
            assertField("izena", "IZENA");
            assertOneToManyListField("artikuluak", "DENDA_ID",
                      "ARTIKULU_INDIZE");

            assertSetField("dendaOrduak");
            CompositeSetFieldMapping dendaOrduak =
getCompositeSetFieldMapping("dendaOrduak");
            dendaOrduak.assertField("astekoEguna", "ASTEKO_EGUNA");
            // ...
            dendaOrduak.assertNonPersistentFields("id");

                    }
```

```
    public void testEskaeraMapping() throws Exception {
         assertClassMapping(Eskaera.class, "ESKAERA_EGIN");
         assertAllFieldsMapped();
    }   }
```

`AldikoEskaera` klasearen eta `ALDIKO_ESKAERA` taularen artean gertatzen den mapaketa ondo gauzatzen dela egiaztatzen du lehenengo metodoak (`testAldikoEskaeraMapring()`). Kontu egin, halaber, klaseak dituen asoziazioak, banatze-helbidea eta eskaeraren artikuluak egiaztatzen direla.

Bigarren metodoarekin (`testDendaMapping()`) antzeko eragiketak gauzatzen dira, baina `Denda` klasea eta `DENDA` taula aztertuta.

Hasierako bi proba horiekin egiaztatzen da mapaketak eta eskemak ondo diseinatuta eta inplementatuta daudela baina, aplikazioa eta aplikazioaren domeinuko osagaiak programatzen hasita, egiaztatzen da osagai horien portaera aztertzea.

Lehen, ikuspegi prozeduralean, transakzio-gidoiak eta DAOak inplementatu baino lehen edo nahi bada aldi berean, probak diseinatu dira osagai horien portaera egiaztatzeko, probak bideratutako garapenaren ezaugarriak erabilita. Orain, aplikazioaren iraunkortasunaren geruza inplementatu nahian gabiltza, Hibernate erabilita, eta, lehen egin den legez, probak diseinatuko dira iraunkortasunaren inguruan egikaritzen diren eragiketak ondo gauzatzen direla egiaztatzeko.

Aldiko eskaerarekin ari garenez aspalditik, osagai horren portaera egiaztatu nahi da eta, horretarako, honako proba sorta hau programatu da, aztertzeko nola erantzuten dion egoera bakoitzari:

```
import java.util.*;

import org.hibernate.*;

public class HibernateAldikoEskaeraIraunkortasunaTests extends
             HibernatePersistenceTestsWithResetDatabase {

       private String aldikoEskaeraId;

       private String dendaId;

       private HibernateDendaGordailuInpl dendaGordailu;

       public HibernateAldikoEskaeraIraunkortasunaTests() {
              super();
       }

       @Override
       protected String[] getConfigLocations() {
              return new String[] {};
```

```
	}

	@Override
	protected String getParentFactoryKey() {
		return "spring-context.xml";
	}

	@Override
	protected String getParentSelector() {
		return "hibernate-domain-model-ref-context.xml";
	}

	public void onSetUp() throws Exception {
		super.onSetUp();
		dendaGordailu = new HibernateDendaGordailuInpl(
				getHibernateTemplate());
		Denda d = DendaNagusi.eginDenda();
		save(d);
		dendaId = d.getId();
	}

	public void testArrunta() throws Exception {

		eraikiAldikoEskaera();

		logger.debug("-----Deitzen eguneratuBanatzeInfo");
		eguneratuBanatzeInfo();

		System.out
				.println("---Deitzen eguneratuBanatzeInfo -
berriz");
		eguneratuBanatzeInfo();

		logger.debug("----------Deitzen eguneratuDenda");
		eguneratuDenda(dendaId);

		System.out
				.println("----Deitzen eguneratuDenda -
berriz");
		eguneratuDenda(dendaId);

		logger.debug("-----Deitzen eguneratuKopuruak1");
		eguneratuKopuruak1();

		logger.debug("----------Deitzen eguneratuKopuruak2");
		eguneratuKopuruak2();

		System.out
				.println("----Deitzen eguneratuKopuruak1 -
berriz");
		eguneratuKopuruak1();
```

```
            logger.debug("----------Deitzen
ezabatuAldikoEskaera");
            ezabatuAldikoEskaera();

            // ...

      }

      private void eraikiAldikoEskaera() {
            AldikoEskaera aldikoe = new AldikoEskaera();
            save(aldikoe);
            aldikoEskaeraId = aldikoe.getId();
      }

      private void eguneratuBanatzeInfo() {
            doWithTransaction(new TxnCallback() {
                  public void execute() throws Exception {
                        Date banatzeData =
DendaNagusi.eginBanatzeData();
                        logger.debug("banatzeData=" + banatzeData);
  Helbidea banatzeHelbidea = new Helbidea("kale1", null, "Irun",
"Gipuzkoa",
                                          "94619");

   AldikoEskaera aldikoe = (AldikoEskaera) load(AldikoEskaera.class,
                                          aldikoEskaeraId);
    int eguneratuBanatzeInfoResult = aldikoe.eguneratuBanatzeInfo(
                                          dendaGordailu, banatzeHelbidea,
banatzeData,
                                          true);

 assertEquals(EskaeraEginEgoeraGako.ADOS, eguneratuBanatzeInfoResult);
                  }
            });
      }

      private void eguneratuDenda(final String dendaId) {
            doWithTransaction(new TxnCallback() {
                  public void execute() throws Exception {
      AldikoEskaera aldikoe = (AldikoEskaera)
load(AldikoEskaera.class,
                                          aldikoEskaeraId);
                        Denda r = (Denda) load(Denda.class, dendaId);
      boolean eguneratuDendaResult = aldikoe.eguneratuDenda(r);
            assertTrue(eguneratuDendaResult);
                  }
            });
      }

      private void eguneratuKopuruak1() {
            doWithTransaction(new TxnCallback() {
                  public void execute() throws Exception {
```

```
                        AldikoEskaera aldikoe = loadAldikoEskaera();
                        aldikoe.eguneratuKopuruak(new int[] { 1, 2 });
                }

        });
    }

    private void eguneratuKopuruak2() {
        doWithTransaction(new TxnCallback() {
            public void execute() throws Exception {
                AldikoEskaera aldikoe = loadAldikoEskaera();
                aldikoe.eguneratuKopuruak(new int[] { 0, 3 });
            }
        });
    }


    private AldikoEskaera lortuAldikoEskaera() {
        logger.debug("Eskuratzen :");
        AldikoEskaera aldikoe = (AldikoEskaera)
load(AldikoEskaera.class,
                        aldikoEskaeraId);
        return aldikoe;
    }

    private void ezabatuAldikoEskaera() {
        doWithTransaction(new TxnCallback() {
            public void execute() throws Exception {
                AldikoEskaera aldikoe = lortuAldikoEskaera();
                delete(aldikoe);
            }
        });
    }

    public void testDendarekin() throws Exception {
        eraikiAldikoEskaera();

        eguneratuBanatzeInfo();

        eguneratuDenda(dendaId);

        logger.debug("Eskuratzen");

        loadAldikoEskaera();

    }   }
```

Ohiko hasieraketak gauzatuta, `onSetUp()` metodoarekin, `testArrunta()` metodoa egikarituta, `AldikoEskaera` klasearen portaera aztertzen da. Lehenengo eta behin, `AldikoEskaera` klasearen instantzia eraikitzen da eta biltegiratu egiten da Hibernateren `save()` metodoa egikarituta. Biltegiratutako osagaiaren identifikadorea eskuratu egiten da, proban zehar erabiltzeko.

`eguneratuBanatzeInfo()` metodoarekin `AldikoEskaera` klasearen instantzia biltegiratu berria eskuratzen da, eta eguneratu egiten zaio banatze-informazioa bere `eguneratuBanatzeInfo()` metodoa egikarituta.

Aurreko metodoarekin egin dugun legez, `eguneratuDenda()` metodoarekin ere, lehenengo eta behin, `AldikoEskaera` instantzia eskuratzen da Hibernateren `load()` metodoa egikarituta. Denda eguneratzen dio metodoak `eguneratuDenda()` metodoa egikarituta eta egiaztatu egiten da eragiketa ondo gauzatu dela.

Egiaztatu ondoren dendarekin egin daitezkeen eragiketak ondo egikaritzen direla, aldiko eskaeraren artikuluen kopuruak aldatu eta eguneratuko dira, `eguneratuKopuruakxx()` metodoak erabilita. Kontu egin ezen bigarren metodoarekin aldiko eskaeraren lehen artikulua ezabatu egiten dela zerrendatik.

Proba arrunta (`testArrunta()`) bukatzen da `ezabatuAldikoEskaera()` metodoa egikarituta. Lehenengo eta behin, biltegiratutako aldiko eskaera eskuratzen da eta, gero, ezabatu egiten da, Hibernateren `delete()` metodoa egikarituta.

Prozesu osoari amaiera emateko diseinatu den osagaiari (`AldikoEskaera`) O/R mapaketa definitu behar zaio ondo kudeatzeko osagaiaren iraunkortasuna. Hibernatek informazio hori XML fitxategi batean jasotzen duenez, hori idatziko da prozesuari amaiera emateko:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping default-access="field"
       package="pakete">
       <class name="AldikoEskaera" table="ALDIKO_ESKAERA"
              dynamic-update="true">
              <id name="id" column="ALDIKO_ESKAERA_ID" type="int"
                     unsaved-value="-1">
                     <generator class="native">
                            <param
name="sequence">UNIQUE_ID_SEQUENCE</param>
                     </generator>
              </id>
              <version name="bertsioa" column="BERTSIOA" />
              <property name="banatzeData" column="BANATZE_DATA"
                     type="timestamp" />
              <property name="egoera" column="EGOERA" />

              <component name="banatzeHelbidea">
                     <property name="kale1" column="BANATZE_KALE1" />
                     <property name="kale2" column="BANATZE_KALE2" />
                     <property name="hiri" column="BANATZE_HIRI" />
```

```
                    <property name="egoera" column="BANATZE_EGOERA" />
                    <property name="postaKodea"
column="BANATZE_POSTA_KODEA" />
             </component>

             <many-to-one name="denda" column="DENDA_ID"
                    cascade="none" />

             <list name="artikuluak"
table="ALDIKO_ESKAERA_ARTIKULU"
                    cascade="all">
                    <key column="ALDIKO_ESKAERA_ID" />
                    <index column="ARTIKULU_INDIZE" />
                    <composite-element class="AldikoEskaeraArtikulu">
                           <property name="kopurua" column="KOPURUA" />
                           <many-to-one name="artikulu"
column="ARTIKULU_ID"
                                  class="Artikulua" cascade="none" />
                    </composite-element>
             </list>

      </class>

      <query
             name="AldikoEskaera.bilatuAldikoEskaeraDendaArtikuluekin1">
             <![CDATA[
from AldikoEskaera ae
 left outer join fetch ae.denda
 left outer join fetch ae.artikuluak as artikulu
 left outer join fetch artikulu.artikulua
 where ae.id = ?
]]>
      </query>

      <query
             name="AldikoEskaera.bilatudAldikoEskaeraDendaArtikuluekin">
             <![CDATA[
from AldikoEskaera ae
 left outer join fetch ae.denda as d
 left outer join fetch d.artikuluak
 where ae.id = ?
]]>
      </query>

      <query name="AldikoEskaera.bilatudAldikoEskaeraArtikuluekin">
             <![CDATA[
from AldikoEskaera ae
 left outer join fetch ae.artikuluak
 where ae.id = ?
]]>
      </query>

</hibernate-mapping>
```

XML fitxategiak honako elementu hauek ditu:

- `AldikoEskaera` klasearen mapaketa konfiguratzen du `class` elementuak, `ALDIKO_ESKAERA` taularekin erlazionatuz. Horrela, aplikazioaren domeinuko klasea eta datu-baseko taula kateatuta gelditzen dira.
- Gako nagusiaren propietatea definitzen du `id` elementuak, adierazita, taularen zein zutabetan bilatu behar duen.
- `banatzeData` eta `egoera` eremuen kateaketa, `property` elementua erabilita gauzatzen da.
- `banatzeHelbidea` eremuaren egitura konplexuagoa denez, `component` elementua erabiltzen da mapaketa gauzatzeko.
- Aldiko eskaeraren artikuluak taula berri batean biltegiratzen direnez, lehenengo eta behin, mapaketa gauzatuko da eta, gero, adierazi egingo da nola erlazionatuko (`many-to-one`) den aldiko eskaerarekin. Ezaugarri horiek guztiak `list` elementuaren barnean adierazten dira.
- Amaitzeko, `AldikoEskaera` osagaia `Denda` osagaiarekin erlazionatuta dagoela adierazten da, `<many-to-one>` elementua erabilita.

Amaierako hiru `query` elementuak `AldikoEskaeraGordailu` osagaiak erabiltzen ditu aldiko eskaerak eskuratzeko datu-basetik, Hibernate lan-markoa erabilita, oraintxe azalduko dugun legez.

Orain arte egindakoaren hausnarketa txiki bat egingo dugu. Hasieran, ORM lan-markoen aurkezpena egin da JDO eta Hibernateren ezaugarri nagusiak aztertuta. Hori egin ondoren, Hibernate lan-markoan finkatu da arreta nagusia eta probak bideratutako garapenaren ikuspegia hartuta, aztertu da nola egituratzen den aplikazioaren domeinuaren iraunkortasuna, Hibernate erabilita. Horretarako, domeinuko klase (`AldikoEskaera`) garrantzitsu bat hautatu da eta diseinatu eta inplementatu egin da.

Beraz, iraunkortasunarekin lotuta dagoen azpiegitura guztia prestatuta utzi da, programatzaileak, erabili eta txerta dezan aplikazio nagusian.

Aplikazioko osagaien eta iraunkortasunaren osagaien arteko kateaketa ez da edozelan gauzatuko, kapsulatu egingo da gordailu erako interfazeak diseinatuta eta inplementatuta. Guk gure adibideko aldiko eskaerarentzat honako interfazea erabiliko dugu:

```
import java.util.*;

public interface AldikoEskaeraGordailu {

       AldikoEskaera eraikiAldikoEskaera();

       AldikoEskaera bilatuAldikoEskaera(String aldikoEskaeraId);
```

```
        AldikoEskaera bilatuEraikiAldikoEskaera(String aldikoEskaeraId);

        AldikoEskaera eraikiAldikoEskaera(
                Helbidea banatzeHelbidea,
                Date banatzeData,
                Denda denda,
                List artikuluak);
}
```

Interfazearen helburu nagusia da iraunkortasuna maskaratzea, kapsulatuz O/R mapaketaren trikimailu guztiak. Hala, baliabidearen osagai-bezeroak nahikoa du jakitea metodoaren signatura zein den osagaia (`AldikoEskaera`) eskuratzeko arazorik gabe. Hemen ez dago ez JDBCrik ez SQL galderarik, hau da, iraunkortasunaren geruzak ez du inongo interferentziarik sortzen aplikazio nagusiaren iturburu-kodean.

Lehen egin den legez, hemen ere, probak bideratutako garapenaren metodologia erabiliko da, interfazea inplementatzen duen klasearen portaera egokia egiaztatzeko. Horretarako, honako proba hau programatu da gordailuaren portaera aztertzeko.

```
import java.util.*;

import ormunit.hibernate.*;

public class HibernateAldikoEskaeraGordailuTests extends
        HibernatePersistenceTests {

    @Override
    protected String[] getConfigLocations() {
        return HibernateTestConstants.HIBERNATE_DOMAIN_CONTEXT;

    }

    private HibernateAldikoEskaeraGordailuInpl gordailu;

    protected AldikoEskaera aldikoEskaera;

    protected void onSetUp() throws Exception {
        super.onSetUp();
        gordailu = new HibernateAldikoEskaeraGordailuInpl(
                    getHibernateTemplate());
    }

    public void testEraikiAldikoEskaera() throws Exception {
        doWithTransaction(new Callback() {
        public void execute() throws Exception {
        aldikoEskaera = gordailu.bilatuEraikiAldikoEskaera(null);
         }

         });
```

```
            String id = aldikoEskaera.getId();
            AldikoEskaera aldikoEskaera2 = (AldikoEskaera)
load(AldikoEskaera.class, id);
            assertNotNull(aldikoEskaera2);
      }

      public void testBilatuAldikoEskaeraDendaArtikuluekin1()
                  throws Exception {
            Denda d = DendaNagusi.eginDenda();
            save(d);
            final String dendaId = d.getId();

            AldikoEskaera ae1 =
gordailu.bilatuEraikiAldikoEskaera(null);
            final String aldikoEskaeraId = ae1.getId();

            doWithTransaction(new TxnCallback() {
                  public void execute() throws Exception {
                        Date banatzeData =
DendaNagusi.eginBanatzeData();
   Helbidea banatzeHelbidea = new Helbidea("kale1", null, "Irun",
"Gipuzkoa",
                                          "94619");

AldikoEskaera ae = (AldikoEskaera) load(AldikoEskaera.class,
                                          aldikoEskaeraId);
  HibernateDendaGordailuInpl dendaGordailu = new
HibernateDendaGordailuInpl(
                                          getHibernateTemplate());
      int eguneratuBanatzeInfoResult = po.eguneratuBanatzeInfo(
                                          dendaGordailu,
banatzeHelbidea, banatzeData,
                                          true);

 assertEquals(EskaeraEginEgoeraGako.ADOS, eguneratuBanatzeInfoResult);
                  }
            });

            doWithTransaction(new TxnCallback() {
                  public void execute() throws Exception {
      AldikoEskaera ae = (AldikoEskaera) load(AldikoEskaera.class,
                                          aldikoEskaeraId);
                        Denda d = (Denda) load(Denda.class, dendaId);
                        boolean eguneratuDendaResult =
ae.eguneratuDenda(d);
                        assertTrue(eguneratuDendaResult);
                  }
            });

            doWithTransaction(new TxnCallback() {
                  public void execute() throws Exception {
```

```
            AldikoEskaera ae = (AldikoEskaera)
load(AldikoEskaera.class,
                                        aldikoEskaeraId);
                            ae.eguneratuKopuruak(new int[] { 1, 2 });
                    } }    );


            logger.debug("eskuratzen");
            AldikoEskaera ae2 = gordailu
                            .bilatuAldikoEskaeraDendaArtikuluekin
(aldikoEskaeraId);
            assertEquals(ae2.getId(), aldikoEskaeraId);
            logger.debug("nabigatzen");
            assertNotNull(((AldikoEskaeraArtikulu)
ae2.getArtikuluak().get(0))
                            .getArtikulu().getIzena());
            logger.debug("eginda");
      }

      public void testBilatuAldikoEskaeraDendaArtikuluekin ()
                  throws Exception {
            AldikoEskaera ae1 =
gordailu.bilatuEraikiAldikoEskaera(null);
            String aeId = ae1.getId();
AldikoEskaera ae2 = gordailu
                            .bilatuAldikoEskaeraDendaArtikuluekin(aeId);
      }

      public void testBilatuAldikoEskaeraArtikuluekin() throws
Exception {
            AldikoEskaera ae1 =
gordailu.bilatuEraikiAldikoEskaera(null);
            String aeId = ae1.getId();
            AldikoEskaera ae2 =
gordailu.bilatuAldikoEskaeraArtikuluekin(aeId);
      }   }
```

`testxxx()` metodoak aztertzen badira zehatz-mehatz, berehala ikusiko da metodo bakoitzak `AldikoEskaera` osagaia erabiltzen duela informazioa datubasetik eskuratzeko. Informazioa eskuratu ondoren, gordailuaren inplementazioaren baliabideak egikaritzen dira, egoera bakoitzean nola erantzuten duten aztertzeko.

Probak programatuta has gaitezen gordailuaren inplementazioarekin.

Jakin badakigu, biltegiratutako informazioa eskuratu nahi badu gordailuaren inplementazioak, `Connection` eta `Query` interfazeak erabili behar dituela. Objektu iraunkorrak eraikitzeko, kargatzeko eta ezabatzeko, `Connection` interfazea erabiliko du gordailuak. Aldiz, `Query` interfazearekin, galderak egikarituko ditu objektu bat baino gehiago eskuratzeko.

Gordailuaren inplementazio-prozesua zeharo zailtzen da, Hibernate lanmarkoaren APIak erabiltzen badira soilik iraunkortasuna kudeatzeko. Guk, oraintxe aztertuko dugun legez, Spring plataformaren ORM baliabideak erabiliko ditugu prozesua sinplifikatzeko.

Inplementazioaren trikimailu guztiak iturburu-kodean islatuko direnez, hona hemen gordailuaren inplementazioaren iturburu-kodea:

```
import java.util.*;

import org.hibernate.*;
import org.springframework.orm.hibernate3.*;
import org.springframework.orm.hibernate3.support.*;

public class HibernateAldikoEskaeraGordailuInpl extends
HibernateDaoSupport
              implements AldikoEskaeraGordailu {

       public HibernateAldikoEskaeraGordailuInpl(HibernateTemplate
template) {
              setHibernateTemplate(template);
       }

       public AldikoEskaera eraikiAldikoEskaera() {
              AldikoEskaera ae = new AldikoEskaera();
              getHibernateTemplate().save(ae);
              return ae;
       }

       public AldikoEskaera bilatuAldikoEskaera(String aldikoEskaeraId)
{
              return (AldikoEskaera)
getHibernateTemplate().load(AldikoEskaera.class,
                           new Integer(aldikoEskaeraId));
       }

       public AldikoEskaera bilatuAldikoEskaera1(String
aldikoEskaeraId) {
              SessionFactory sessionFactory = getSessionFactory();
       Session session = SessionFactoryUtils.getSession(sessionFactory,
true);
              try {
   return (AldikoEskaera) session.load(AldikoEskaera.class, new
Integer(
                                  aldikoEskaeraId));
              } catch (HibernateException e) {
                     throw
SessionFactoryUtils.convertHibernateAccessException(e);
              } finally {
                     SessionFactoryUtils.releaseSession(session,
sessionFactory);
```

```
            }
      }

      public AldikoEskaera eraikiAldikoEskaera(Helbidea
banatzeHelbidea,
                  Date banatzeData, Denda denda,
                  List artikuluak) {
            AldikoEskaera ae = new AldikoEskaera(banatzeHelbidea,
banatzeData,
                        denda, artikuluak);
            getHibernateTemplate().save(ae);
            return ae;
      }

      public AldikoEskaera bilatuEraikiAldikoEskaera(String
aldikoEskaeraId) {
            return aldikoEskaeraId == null ? eraikiAldikoEskaera()
                        : bilatuAldikoEskaera(aldikoEskaeraId);
      }

      public AldikoEskaera bilatuAldikoEskaeraDendaArtikuluekin1(
                  String aldikoEskaeraId) {
            return (AldikoEskaera) getHibernateTemplate()
                        .findByNamedQuery(
            "AldikoEskaera.bilatuAldikoEskaeraDendaArtikuluekin1",
                                    new
Integer(aldikoEskaeraId)).get(0);
      }

      public AldikoEskaera bilatuAldikoEskaeraDendaArtikuluekin(
                  String aldikoEskaeraId) {
            return (AldikoEskaera)
getHibernateTemplate().findByNamedQuery(
            "AldikoEskaera.bilatuAldikoEskaeraDendaArtikuluekin",
                        new Integer(aldikoEskaeraId)).get(0);
      }

      public AldikoEskaera bilatuAldikoEskaeraArtikuluekin(String
aldikoEskaeraId) {
            return (AldikoEskaera)
getHibernateTemplate().findByNamedQuery(
                  "AldikoEskaera.findAldikoEskaeraArtikuluekin",
                  new Integer(aldikoEskaeraId)).get(0);
      }   }
```

Diseinu aldetik eta inplementazioaren goiburuari erreparatzen badiogu, konturatuko gara ezen gordailuaren inplementazioak `HibernateDaoSupport` klasea hedatzen duela eta, jakina, `AldikoEskaeraGordailu` interfazea inplementatzen duela. Interfazearen inplementazioa begi-bistakoa da hala diseinatu baita gordailua. `HibernateDaoSupport` klaseak beste helburu bat du: Spring

plataformaren ORM baliabideak txertatzea gordailuaren inplementazioan, konexioak eta transakzioak kudeatzeko. Gainera, `HibernateTemplate` osagaiaren baliabideak ere programatzailearen esku jartzen ditu, Hibernateren lana errazteko, sesioak irekitzeko eta kudeatzeko.

Lana egituratzeko, gordailuaren inplementazioaren metodo eraikitzaileak `HibernateTemplate` motako objektua jasotzen du parametro gisa. Metodo eraikitzaileak egiten duen lan bakarra da jasotako `HibernateTemplate` osagaia esleitzea eta finkatzea, `HibernateDaoSupport` klasearen `setHibernateTemplate()` metodoa egikarituta. Horrela, gordailuaren inplementazioko edozein metodok aipatutako `template`-a eskuratu nahi badu, nahikoa du `getHibernateTemplate()` metodoa egikaritzea. Adibidez, `eraikiAldikoEskaera()` eta `bilatuAldikoEskaera()` metodoek estrategia hori erabiltzen dute Hibernate lan-markoaren baliabideetara atzitzeko. Lehenengokoak, Hibernateren `Session` interfazearen `save()` metodoa egikaritzen du eta, bigarrenak, `load()` metodoa. Zehatz-mehatz, honako hau egikaritzen da `template`-rekin: Hibernateren `Session` osagaia eskuratu, metodoa (`save()` edo `load()`) egikaritu eta beharrezkoa denean `Session`-a itxi.

Kontura gaitezen `template`-ak nola sinplifikatzen duen prozesua, `bilatuAldikoEskaera1()` metodoak beste bide bat hartu du Spring plataforma eta Hibernate lan-markoa harremanetan jartzeko. Horrek, jakina, zaildu egingo du metodoaren inplementazioa, salbuespenak eta sesioaren kudeaketa, programatzailearen esku gelditzen baitira.

Metodoaren lehenengo bi lerroekin sesio bat irekitzen da Spring plataformaren `SessionFactoryUtils` osagaia erabilita. Sesioa ireki ondoren, lehen bezala aurreko metodoekin, Session osagaiaren `load()` metodoa egikaritzen da `AldikoEskaera` motako objektu bat eskuratzeko. Kontu egin ezen iturburu-kode hori `try/catch/finally` motako egitura batean txertatzen dela altxa daitezkeen salbuespenak tratatzeko eta bukaeran sesioa ixteko, `releaseSession()` metodoa egikarituta.

Gordailuaren inplementazioaren azken hiru metodoek (`bilatuAldikoEskaeraDendaArtikuluekin1()`,`metodoek(bilatuAldikoEskaeraDendaArtikuluekin()`, `bilatuAldikoEskaeraArtikuluekin()`) Hibernateren XML fitxategian adierazitako `query`[63] elementuak erabiltzen dituzte aldiko eskaera eskuratzeko `AldikoEskaera` osagaiarentzat.

63. `query` elementuek *eager-loading* (kargatze azkarra) erako atzipenak gauzatzen dituzte, hau da, `SELECT` agindu bakarrarekin hainbat objektu eskuratzen dira aldi berean. Bestalde, *lazy-loading* (kargatze nagia) era erabiliko balitz, ORM lan-markoak eskaeraren arabera eskuratuko lituzke objektuak.

4.4. DOMEINUAREN LOGIKAREN TRATAERA POJO-FAÇADE BIDEZ

Aplikazio informatiko baten eraikuntzan hainbat erabaki hartu behar du erakunde informatikoak programatzen hasi baino lehen. Lantzean behin, erabaki horietako batzuk bezeroek finkatzen dituzte sistemaren betekizunetan eta erakunde informatikoak egokitu egin behar ditu aplikazioaren eraikuntzaren zenbait arlo betekizun horietara. Adibidez, eraiki behar duen aplikazioa integratu behar badu aurretik eraikitako beste batekin, bigarren horrek aurrebaldintzak jarriko dizkio bai teknologia aldetik bai analisi eta diseinu aldetik.

Iraunkortasunaren geruza inplementatzeko JDBC teknologia erabili badu zuzenean lehenengo aplikazioak, zaila egingo da Hibernate lan-markoa edota JDO estandarra inplementatzen duen lan-markoa erabiltzea bigarren aplikazioan, lehenengokoari, aldaketa sakonak egin gabe iraunkortasunaren geruzan. Ildo beretik, antzeko egoeran egongo litzateke erakunde informatikoa, lehenengo aplikazioan, domeinuaren logika, ikuspegi prozedurala erabilita antolatu bada. Testuinguru horretan, objektuetara zuzendutako garapena aplikatzea ia ezinezkoa izango da, batez ere, Domeinuaren Eredua bera bada bi aplikazioetarako. Egoera ez bada asko okertu, baina, badago bide bat arazoa konpontzeko domeinuaren logika kapsulatuta, Façade[64] patroi ezaguna erabilita.

Domeinuaren logika kapsulatuta gelditzeak Façade patroiaren atzean, abantailak dakarzkio aurkezpen mailari, behar dituen zerbitzuak zuzenean eska baititzake, arduratu gabe nola gauzatzen diren. Badaki, orakulu antzeko zerbait eskura duela eta aplikazioaren erabiltzaileak zerbait eskatzen dionean egikaritzeko, beti galde diezaioke orakulu horri zein den erantzuna.

Erantzuna, `Result` erako egitura batean itzuliko dio Façade-orakuluak, eta ez du inongo DAO edota DTOrik behar, informazioa eraikitzeko, eskuratzeko, eguneratzeko edota ezabatzeko.

Aplikazioa inplementatzeko EJB teknologia erabiliko balitz, patroiaren ezaugarriak `Session` erako osagai batean kapsulatuko lirateke baina, hala, edukiontzi pisutsua (Heavyweight Container) erabili beharko luke aplikazioa garatzeko horrek dakartzan abantaila eta desabantaila guztiekin. Geroago aztertuko dugu bide hori.

Goian azaldu denaren zehaztasunak argitzeko, aurreko orrietan erabili den domeinua berreskuratuko da eta erabilera-kasu txiki bat aztertuko da.

Aplikazioaren erabiltzaileak zehaztu ondoren eskaera bat gauzatu nahi duela, adierazi eta aukeratu behar du zein dendak hornituko dion eskaera. Urrats hori beteta, dendaren artikuluak aukeratzeko eta artikulu bakoitzaren kopurua sartzeko eskatuko dio aplikazioak. Informazio guztia sartuta, eskaera onartzen du aplikazioak eta banatze-prozesua hasten da martxan.

64. Informazio gehiago, egilearen aurreko liburuaren, 290. eta 325. orrialdeetan.

Horra hor, bada, egikaritu nahi den prozesuaren aurkezpen maila. Kontu egin ezen diseinatu eta inplementatu nahi den Façade patroiaren nondik norakoak aurkezpen mailak bideratuko dituela, hau da, patroia, aurkezpen mailaren beharren arabera diseinatuko da. Bertan egiten diren galderek eta jasotzen diren erantzunek bideratuko dituzte aplikazioan egikarituko diren Façade patroiak.

Goian azaldu den testuinguruan inplementatu nahi bada Façade patroia, honako urrats hauek gauzatu behar dira:

1. Façade patroiaren interfaze publikoa diseinatu.
2. Diseinatutako interfazearen metodoak inplementatu, domeinuaren logika txertatuta.
3. `Result` motako objektua inplementatu, jasotzen diren emaitzak era egituratuan bidaltzeko aurkezpen mailari.
4. Spring plataformaren objektu harrapariak (`Transaction Interceptor`) konfiguratu, egikarituko diren transakzioak kudeatzeko.

Lehenengo urratsean aipatzen den interfazea diseinatzeko, aurkezpen mailaren portaera aztertzen da ondo, hau da, erabiltzaileak sartu edota aukeratzen duen informazioa eta aurkezpen mailan azalerazten den informazioa aplikazioaren barrenetik aztertzen dira.

Aurkezpen mailan ageri diren sarrera-irteera horien azterketaren emaitzak aztertuta, honako interfaze publiko hau diseinatu da:

```
import java.util.*;

public interface EskaeraEginFacade {

    public EskaeraEginFacadeResult eguneratuBanatzeInfo(
            String aldikoEskaeraId, Helbidea banatzeHelbidea,
            Date banatzeData);

    public EskaeraEginFacadeResult eguneratuDenda(
            String aldikoEskaeraId, String dendaId);

    public EskaeraEginFacadeResult eguneratuKopuruak(
            String aldikoEskaeraId, int[] kopuruak);

    public EskaeraEginFacadeResult EskaeraEgin(String
aldikoEskaeraId);

    public EskaeraEginFacadeResult egiaztatuBanatzeInfoAldaketa(
            String aldikoEskaeraId, Helbidea banatzeHelbidea,
            Date banatzeData);

    public EskaeraEginFacadeResult getAldikoEskaera(String
aldikoEskaeraId);
}
```

Interfazearen metodoek aurkezpen mailatik jasotako informazioa erabiltzen dute parametroetan, eta metodoa egikaritu ondoren, `Result` motako objektu egituratua itzultzen dute. Itzulitako objektu egituratu hori hirugarren urratsean inplementatuko da, eta aurkezpen mailak erabiliko du eskaeraren informazioa kudeatzeko. Horrela, aurkezpen mailak, adierazpide bateratua eta adostua erabilita, beti izango du kontrolatua eskaera informazio aldetik.

Interfazea diseinatuz gero, inplementazioa erantsiko zaio bigarren urratsean. Laster ikusiko den legez, metodoen inplementazioa ez da oso zaila izango, domeinuaren logika ez baita metodo horietan egongo txertatua, erabiltzen diren domeinuko objektuetan baizik.

Prozesu horren bidez, domeinuko objektuetan delegatzen du metodoak domeinuaren logika. Bere eginbeharra izango da eraikitzen den informazio guztia kudeatzea eta kontrolatzea, `Result` motako objektua itzultzeko.

Aurreko egoeretan egin den bezala, hemen ere, probak bideratutako garapena erabiliko da interfazea inplementatzeko. Hala, interfazearen metodoen portaeran soilik jarriko da arreta, hori baita inplementatu eta probatu nahi dena eta ez domeinuaren klaseetan txertatuta dagoen domeinuaren logika edota datu-basearekin izaten den harremana. Horretarako, `Mock`[65] motako objektuak eraikiko dira probaren hasierako `setUp()` metodoan eta, hala, probaren testak nahi ditugun bidetik bideratuko dira

Adibidez, dendaren eguneratzea probatu nahi bada, honako iturburu-kodea erabili beharko du aplikazioak:

```
import java.util.*;

import org.jmock.*;
import org.jmock.core.*;

public class EskaeraEginFacadeMockTests extends MockObjectTestCase {

    private Mock mockGordailua;

    private Mock mockEskaeraEginZerbitzua;

    ..................

    public void setUp() {
        mockGordailua = new Mock(DendaGordailu.class);
        mockEskaeraEginZerbitzua = new
Mock(EskaeraEginZerbitzua.class);
        mockResultFaktoria = new Mock(
                EskaeraEginFacadeResultFaktoria.class);
```

65. `http://en.wikipedia.org/wiki/Mock_Object`

```
        EskaeraEginZerbitzua = (EskaeraEginZerbitzua)
mockEskaeraEginZerbitzua
                .proxy();
        resultFaktoria = (EskaeraEginFacadeResultFaktoria)
mockResultFaktoria
                .proxy();
        DendaGordailu = (DendaGordailu) mockGordailua
                .proxy();
        eskaeraEginFacade = new EskaeraEginFacadeImpl(
                EskaeraEginZerbitzua, DendaGordailu,
                resultFaktoria);

        dendak = new ArrayList();

        aldikoEskaeraId = "aldikoEskaeraId";
        banatzeHelbidea = DendaTestData.lortuHelbidea();
        banatzeData = DendaTestData.eraikiBanatzeData();

        aldikoEskaera = new AldikoEskaera();
    }

    public void testEguneratuDenda() throws Exception {
        String dendaId = "dendaId";

        EskaeraEginZerbitzuaResult EskaeraEginZerbitzuaResult = new
EskaeraEginZerbitzuaResult(
                EskaeraEginEgoeraGako.ADOS, aldikoEskaera);

            EskaeraEginFacadeResult resultFaktoriaResult = new
EskaeraEginFacadeResult(
                EskaeraEginEgoeraGako.ADOS, aldikoEskaera,
                bilatutakoDendak);

        mockEskaeraEginZerbitzua.expects(once()).method(
                "eguneratuDenda").with(eq(aldikoEskaeraId),
                eq(dendaId)).will(
                (returnValue(eskaeraEginZerbitzuaResult)));

        mockResultFaktoria.expects(once()).method("egin").with(
                eq(EskaeraEginEgoeraGako.ADOS), eq(aldikoEskaera))
                .will(returnValue(resultFaktoriaResult));

        EskaeraEginFacadeResult result = eskaeraEginFacade
                .eguneratuDenda(aldikoEskaeraId, dendaId);

        assertSame(resultFaktoriaResult, result);
    }}
```

Hasierako `setUp()` metodoak, proba gauzatzeko behar dituen `Mock` motako objektuak eraikitzen ditu lehendabizi, eta gero, `EskaeraEginFacade` motako objektua eraikitzen du, objektu horiek erabilita metodo eraikitzailean.

`EskaeraEginFacade` motako objektua, `testEguneratuDenda()` metodoan erabiliko da proba gauzatzeko. Kontu egin ezen aipatutako metodoaren mamia azken lerroetan dagoela, hau da, `EskaeraEginFacade` motako objektuaren `eguneratuDenda()` metodoa egikaritzen denean eta emaitza `assertSame()` metodoan txertatzen denean emaitzak erkatzeko.

Beste proba bat gauzatu nahiko balitz, antzeko metodologia erabili beharko luke: beharrezko `Mock` objektuak eta `EskaeraEginFacade` motako objektua eraiki eta proba programatu, objektu horien portaera aztertzeko.

Diseinatu den proba konpilatu eta exekutatu nahi bada, `EskaeraEginFacade` motako objektua eskuragarri izan behar du probak. Horretarako, Façade patroia inplementatzeko azaldu den bigarren urratsean murgiltzen gara buru-belarri, eta, inplementazioa gehitzen zaio `EskaeraEginFacade` interfaze publikoari:

```
import java.util.*;

public class EskaeraEginFacadeInpl implements EskaeraEginFacade {

    private DendaGordailu dendaGordailu;

    private EskaeraEginFacadeResultFaktoria resultFaktoria;

    private EskaeraEginZerbitzua zerbitzua;

    public EskaeraEginFacadeInpl(EskaeraEginZerbitzua zerbitzua,
            DendaGordailu dendaGordailu,
            EskaeraEginFacadeResultFaktoria resultFaktoria) {
        this.zerbitzua = zerbitzua;
        this.dendaGordailu = dendaGordailu;
        this.resultFaktoria = resultFaktoria;
    }

    public EskaeraEginFacadeResult eguneratuBanatzeInfo(
            String aldikoEskaeraId, Helbidea banatzeHelbidea,
            Date banatzeData) {
        EskaeraEginZerbitzuaResult result = zerbitzua
                .eguneratuBanatzeInfo(aldikoEskaeraId,
                        banatzeHelbidea, banatzeData);
        AldikoEskaera aldikoEskaera = result.getAldikoEskaera();
        if (result.getEgoeraGako() == EskaeraEginEgoeraGako.ADOS) {
            List dendak = (List) dendaGordailu
                    .eskuratuBilatutakoDendak(banatzeHelbidea,
                            banatzeData);
            //...
            return resultFaktoria.egin(EskaeraEginEgoeraGako.ADOS,
                    aldikoEskaera, dendak);
        } else {
            return resultFaktoria.egin(
                    EskaeraEginEgoeraGako.BANATZE_INFO_EZ_EGOKIA,
```

```
                    aldikoEskaera, null);
        }
    }

    public EskaeraEginFacadeResult eguneratuDenda(
            String aldikoEskaeraId, String dendaId) {
        EskaeraEginZerbitzuaResult result = zerbitzua.eguneratuDenda(
                aldikoEskaeraId, dendaId);
        return resultFaktoria.egin(result.getEgoeraGako(), result
                .getAldikoEskaera());

    }

    public EskaeraEginFacadeResult eguneratuKopuruak(
            String aldikoEskaeraId, int[] kopuruak) {
        EskaeraEginZerbitzuaResult result =
zerbitzua.eguneratuKopuruak (
                aldikoEskaeraId, kopuruak);
        return resultFaktoria.egin(EskaeraEginEgoeraGako.ADOS, result
                .getAldikoEskaera(), null);
    }

    public EskaeraEginFacadeResult eskaeraEgin(String aldikoEskaeraId)
{
        EskaeraEginZerbitzuaResult result = zerbitzua
                .EskaeraEgin(aldikoEskaeraId);
        return resultFaktoria.egin(EskaeraEginEgoeraGako.ADOS, result
                .getEskaera());
    }

    public EskaeraEginFacadeResult egiaztatuBanatzeInfoAldaketa(
            String aldikoEskaeraId, Helbidea banatzeHelbidea,
            Date banatzeData) {
        throw new Salbuespena();
    }

    public EskaeraEginFacadeResult getAldikoEskaera(
            String aldikoEskaeraId) {
        throw new Salbuespena();
    }  }
```

Guri dagokigun `eguneratuDenda()`[66] metodoa aztertzen bada, argi ikusten da zein den prozesua metodo bakoitza egikaritzeko. Lehenengo eta behin, `EskaeraEginZerbitzua` motako objektuaren metodo egokia egikaritzen da emaitza egokia eskuratzeko eta, gero, `Faktoria` motako objektuaren `egin()` metodo egokia egikaritzen da, `EskaeraEginFacadeResult` motako objektua eskuratzeko.

66. Gogoratu ezen metodo hori dela `testEguneratuDenda()` metodoak aztertzen duena aurreko proban.

Gure kasuan, lehendabizi, `EskaeraEginZerbitzua` motako objektuaren `eguneratuDenda()` metodoa egikaritzen da `EskaeraEginZerbitzuaResult` motako objektua eskuratzeko, eta gero, objektu horren baliabideak (`getEgoeraGako()` eta `getAldikoEskaera()`) erabiltzen dira `Faktoria` motako objektuaren `egin()` metodoan, `EskaeraEginFacadeResult` motako objektua itzultzeko.

Badirudi orain artekoa ikusita, bide bihurri eta konplexuena aukeratu dela ditxosozko Façade patroia inplementatzeko. Ez da hala, objektu mota bakoitza eginbehar bat betetzeko diseinatu eta inplementatu da. Arestian azaldu da ezen Façade patroiak domeinuaren logika delegatu egiten ziola beste norbaiti diseinua sinplifikatu eta inplementazioa errazteko. Jakina, domeinuaren logika nonbait inplementatzen da eta horretarako daude `Zerbitzua` motako objektuak: gauzak errazteko eta lana kentzeko Façade patroiari. Hala, Façade patroia, zerbitzu horietaz baliatzen da domeinuaren logika inplementatzeko. Gero, `Faktoria` motako objektuen baliabideak (`egin()`) erabiltzen ditu emaitza eraiki eta ondo itzultzeko metodo bakoitzaren bezeroari.

Façade patroiko metodo bakoitzak itzultzen duen `Result` motako objektuaren inplementazioa hirugarren urratsean gauzatzen da:

```
import java.util.*;

public class EskaeraEginFacadeResult {

    private int egoeraGako;
    private AldikoEskaeraZehatza aldikoEskaera;
    private List dendak;
    private Eskaera eskaera;

    public EskaeraEginFacadeResult(
        int egoeraGako,
        AldikoEskaeraZehatza aldikoEskaera,
        List dendak) {
        this.egoeraGako = egoeraGako;
        this.aldikoEskaera = aldikoEskaera;
        this.dendak = dendak;
    }

    public EskaeraEginFacadeResult(
        int egoeraGako,
        AldikoEskaeraZehatza aldikoEskaera) {
        this(egoeraGako, aldikoEskaera, null);
    }

    public EskaeraEginFacadeResult(int egoeraGako, Eskaera eskaera) {
        this.egoeraGako = egoeraGako;
        this.eskaera = eskaera;
    }
```

```
    public Eskaera getEskaera() {
        return eskaera;
    }
    public int getEgoeraGako() {
        return egoeraGako;
    }

    public AldikoEskaeraZehatza getAldikoEskaera() {
        return aldikoEskaera;
    }

    public List getDendak() {
        return dendak;
    }}
```

Mota horretako objektuak emaitza egituratuak eraikitzen ditu metodo eraikitzaileekin eta metodoak inplementatzen ditu edukietara atzitzeko. Horrela, aurkezpen mailak ez du egokitzapen-lanik egin behar emaitzen grafoa eraikitzeko. Behar duen informazioa objektu egituratu bakarrean datorkio eta ez zatika.

`Result` motako objektu egituratuan dauden objektu iraunkorrak, zuzenean atzitu daitezke JDO edota Hibernate bezalako teknologiak erabilita. Horrela egingo balitz, iraunkortasunaren kudeaketa erabat lotuta gelditzuko litzateke erabiltzen den teknologiari. Horren ordez, irtenbide generikoagoa aukeratu da iraunkortasunaren arlo hori lantzeko eta `ResultFaktoria` motako objektuak erabiliko dira:

```
import java.util.*;

public interface EskaeraEginFacadeResultFaktoria {

    EskaeraEginFacadeResult egin(
        int egoeraGako,
        AldikoEskaera aldikoEskaera,
        List dendak);

    EskaeraEginFacadeResult egin(
            int egoeraGako,
            AldikoEskaera aldikoEskaera);

    EskaeraEginFacadeResult egin(int egoeraGako, Eskaera eskaera);
}
```

Hala, Façade patroiaren diseinua eta erabiliko den teknologia, iraunkortasuna kudeatzeko, banandu egiten dira arkitektura malguagoa lortzeko aplikazioarentzat. Demagun patroiaren objektuen iraunkortasuna Hibernate teknologia erabilita gauzatzen dela:

```
import java.util.*;

import org.springframework.orm.hibernate3.*;
import org.springframework.orm.hibernate3.support.*;
```

```
public class HibernateEskaeraEginFacadeResultFaktoria extends
        HibernateDaoSupport implements
        EskaeraEginFacadeResultFaktoria {

    public HibernateEskaeraEginFacadeResultFaktoria(
            HibernateTemplate hibernateTemplate) {
        setHibernateTemplate(hibernateTemplate);
    }

    public EskaeraEginFacadeResult egin(int egoeraGako,
            AldikoEskaera aldikoEskaera, List dendak) {

getHibernateTemplate().initialize(aldikoEskaera.getArtikuluak());
        Denda denda = aldikoEskaera.getDenda();
        if (denda != null) {
            List artikuluak = denda.getArtikuluak();
            getHibernateTemplate().initialize(artikuluak);
        }
        return new EskaeraEginFacadeResult(egoeraGako, aldikoEskaera,
                dendak);
    }

    public EskaeraEginFacadeResult egin(int egoeraGako,
            AldikoEskaera aldikoEskaera) {
        return egin(egoeraGako, aldikoEskaera, null);
    }

    public EskaeraEginFacadeResult egin(int egoeraGako, Eskaera
eskaera) {
        getHibernateTemplate().initialize(eskaera.getArtikuluak());
        getHibernateTemplate().initialize(eskaera.getDenda());
        return new EskaeraEginFacadeResult(egoeraGako, eskaera);
    }}
```

Iturburu-kodean adierazten den legez, Hibernate teknologia txertatu nahi bada osatzen ari garen diseinuan, arestian azaldu den interfazea (`EskaeraEginFacadeResultFaktoria`) inplementatuko da `egin()` metodoak inplementatuta. Horretarako, lehen egin zen bezala, hemen ere, metodo eraikitzaileari `HibernateTemplate` motako objektua pasatzen zaio, `setHibernateTemplate()` metodoa egikaritzeko. Horrela, `egin()` metodo bakoitzak nahikoa du `getHibernateTemplate()` metodoa erabiltzea hasieraketak gauzatzeko. Hala, ziurtatuta (`initialize()`) gelditzen da behar duen informazioa eskuragarri dagoela patroiarentzat.

Façade patroiaren ezarpena laugarren urratsean gauzatuko da Spring plataformaren objektu harrapariak (`Transaction Interceptor`) erabilita. Lehen egin den bezala, hemen ere, generikoa den arloa XML fitxategi berezi batean adieraziko da eta Hibernate teknologiarekin lotua dagoena beste batean.

Hona hemen, Spring plataformak erabiliko duen XML fitxategiaren edukia, Façade patroiaren alde generikoa kudeatzeko:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

 <beans default-lazy-init="true">

       <bean id="EskaeraEginZerbitzua"
                    lazy-init="true"
              class="EskaeraEginZerbitzuaInpl">
              <constructor-arg>
                <ref bean="AldikoEskaeraGordailuaInpl" />
              </constructor-arg>
              <constructor-arg>
                <ref bean="DendaGordailuInpl" />
              </constructor-arg>
              <constructor-arg>
                <ref bean="EskaeraGordailua" />
              </constructor-arg>
       </bean>

       <bean id="EskaeraEginFacade"
                    lazy-init="true"
              class="EskaeraEginFacadeInpl">
              <constructor-arg>
                <ref bean="EskaeraEginZerbitzua" />
              </constructor-arg>
              <constructor-arg>
                <ref bean="DendaGordailuInpl" />
              </constructor-arg>
              <constructor-arg>
                <ref bean="ResultFaktoria" />
              </constructor-arg>
       </bean>

       <bean id="TransactionRetryInterceptor"
             class="TransactionRetryInterceptor">
             <property name="maxRetryCount"><value>5</value></property>
       </bean>

    <bean id="matchAllMethods"

class="org.springframework.transaction.interceptor.MatchAlwaysTrans
actionAttributeSource" />

    <bean id="TransactionInterceptor"

class="org.springframework.transaction.interceptor.TransactionInterce
ptor">
```

```
        <property name="transactionManager">
            <ref bean="myTransactionManager" />
        </property>
        <property name="transactionAttributeSource">
            <ref bean="matchAllMethods" />
        </property>
    </bean>

    <bean id="transactionProxyCreator"

class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCr
eator">
        <property name="beanNames">
            <list>
                <idref bean="EskaeraEginFacade" />
            </list>
        </property>
        <property name="interceptorNames">
            <list>
                <idref bean="TransactionInterceptor" />
            </list>
        </property>
    </bean>
</beans>
```

Lehenengo bean-definizioak, patroiaren zerbitzuak kudeatuko dituen osagaia eraikitzeko balio du. Gogoratu osagai horrek kudeatzen diola patroiari domeinuaren logika. Hala, patroiaren diseinua zeharo errazten da, murriztu eta modularizatu egiten baitira ardurak.

Bigarren bean-definizioarekin, Façade patroiaren osagai nagusia eraikitzen da behar dituen osagaiak erabilita.

Spring plataformaren objektu harrapariaren (`Transaction Interceptor`) konfigurazioa, XML fitxategiaren bigarren zatian gauzatzen da. Hala, `EskaeraEginFacade` osagaia transakzionala bihurtzen da eta lotua gelditzen da Spring plataformara.

Amaitzeko, bigarren XML fitxategia eraikitzen da aplikazioa datu-basearekin harremanetan jartzeko:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

       <bean id="AldikoEskaeraGordailuaInpl" lazy-init="true"
             class="HibernateAldikoEskaeraGordailuaInpl">
```

```
            <constructor-arg ref="HibernateTemplate" />
      </bean>

      <bean id="DendaGordailuInpl" lazy-init="true"
            class="HibernateDendaGordailuInpl">
            <constructor-arg ref="HibernateTemplate" />
      </bean>

      <bean id="ResultFaktoria" lazy-init="true"
            class="HibernateEskaeraEginFacadeResultFaktoria">
            <constructor-arg ref="HibernateTemplate" />
      </bean>

      <bean id="HibernateTemplate" lazy-init="true"

class="org.springframework.orm.hibernate3.HibernateTemplate">
            <property name="sessionFactory"
ref="niresessionFactory" />
      </bean>

      <bean id="myTransactionManager" lazy-init="true"
class="org.springframework.orm.hibernate3.HibernateTransactionManager"
>
            <property name="sessionFactory" ref="niresessionFactory" />
      </bean>

      <bean id="niresessionFactory" lazy-init="true"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
          <property name="mappingLocations">
          <list>
           <value>classpath:AldikoEskaera.hbm.xml</value>
          .....
             </list>
          </property>

          <property name ="hibernateProperties">
          <props>
           ....
          </props>
          </property>
      </bean>
    ......
</beans>
```

Hibernateren testuingurutik aplikazioaren osagaietara doan bidea egin nahi bada, atzetik aurrera aztertu behar dugu XML fitxategia. Hala, `niresessionFactory` bean-definizioak Hibernateren `SessionFactory` osagaia eraikitzen du hainbat mapaketa-fitxategi erabilita.

Transakzioen kudeaketa, `myTransactionManager` bean-definizioak antolatuko du eta `HibernateTemplate` osagaiak eraiki berri den `SessionFactory`-a izango du erreferentziatua.

Amaitzeko, aplikazioaren domeinuko osagaien eta emaitzen osagaien instantziazioa, hasierako hiru bean-definizioekin gauzatzen da. Kontu egin ezen hiruretan `HibernateTemplate`-a injektatzen dela osagaia eraikitzeko.

4.5. APLIKAZIOAREN OSAGAIEN INPLEMENTAZIOA EJB 3.0 ERABILITA

Softwarea garatzen duen edozein erakundek jakin daki softwarea organikoa dela, hau da, denboraz, ezaugarri batzuk desagertu egiten direla, landu gabe gelditu zirenak garatu egiten direla eta, ez gutxitan, berriak eransten direla softwarea eguneratzeko eta ekosistema berrietara egokitzeko.

Ekosistemaren konplexutasun mailak aldaketen abiaduran du eragina. Ekosistema egonkorrak ez du aldaketa handirik eskatzen, gauzak dauden horretan egonda nahikoa da sistema osoa martxan edukitzeko aldaketa txikiekin. Ekosistemaren konplexutasuna handia denean, edozein aldaketa txikik ondorio handiak eduki ditzake sistema osoan. Espezie bakoitzak azkar erantzun behar dio aldaketari eta erantzun egokia bilatuta gainera, bere biziraupena baitago jokoan.

Softwarearen teknologien ekosistema konplexuan antzeko zerbait gertatu zitzaien EJB 2 espezifikazioa inplementatzen zutenei. Spring, JDO eta Hibernate teknologia berrien sorkuntzak haizeak harrotu zituen EJB 2 teknologiaren inguruan. Bakoitzak bere aldetik, teknologiaren alde ahulak plazaratu zituen enperadorea biluztuz. Batek, astuna deitu zion, besteak, korapilatsua. Lotsatia ere ez zen atzean gelditzekoa eta ahopetik, zaharkitua atera zitzaion. Enperadore azkarra berehala konturatu zen zerbait egin behar zuela erresuma berreskuratzeko eta horretarako, JCPko (Java Community Process) partaideei deitu eta konklabea antolatu zuen EJBren espezifikazio publiko eta ireki berria eraikitzeko. Izena asmatzen ez zuten denbora askorik eman: EJB 3.

Aurrekoan bezala, espezifikazio berrian, hiru bean mota daude: sesio-beanak, mezuak bideratutako beanak eta entitate-beanak. Lehen biak, edukiontziaren barnean bizi dira eta bere kontrolpean egikaritzen dira beharrezkoak diren zerbitzuak. Entitate-beanak, ostera, edukiontziaren kontrolpetik at bizi dira iraunkortasun-hornitzailearen (*persistence provider*) agindupean.

Lehen hurbilketan, iraunkortasun-hornitzailea, ORM (Object-Relational Mapping) teknologia bat litzateke zeinak EJB 3ren Java Persistence APIa (JPA) inplementatzen duen. JPAren zerbitzuak egikaritzeko jarraitzen den eredua eta JDBC teknologiaren APIarekin jarraitzen dena antzekoak dira.

JPAk estandar bat definitzen du honako hau gauzatzeko:

- ORM teknologietako mapaketaren konfigurazio-metadatuak eraiki.
- CRUD (Create, Read, Update eta Delete) eragiketak egikaritu `EntityManager` APIa erabilita.
- Biltegiratutako aplikazioaren datuak bilatu eta eskuratu JPQL (Java Persistence Query Language) lengoaia erabilita.

Espezifikazio berriarekin bean erako osagai bat inplementatzea nabari erraztu da. Orain, ez da `Home` erako interfazerik eraikiko `EJBHome` interfazea hedatuta, eta programatzaileak bezeroaren deia zerbitzarian kudeatzeko sortu behar zuen interfazea, `EJBObject` interfazea hedatuta, desagertu egin da. EJB 3an interfazeak POJI (Plain Old Java Interface) erako interfazeak izango dira eta beana eraikitzeko klasea POJO (Plain Old Java Object[67]) erako objektu arrunta izango da. Demagun gure `KaixoPatxi` ezaguna programatu nahi dugula beanak erabilita. EJB 2 erabiliko bagenu, hainbat interfaze eta salbuespen-egoera kontrolatu beharko genituzke irteera estandarretik ditxosozko agurra ateratzeko. EJB 3 espezifikazioarekin, hona hemen programatu behar duzun guztia:

```
public interface KaixoPatxi {
public void kaixoPatxi(String izena);
  }
                                    oOo
import javax.ejb.Stateless;
@Stateless
public class KaixoPatxiBean implements KaixoPatxi {
public void kaixoPatxi(String izena) {
System.out.println("Kaixo " + izena + " oraingoan, EJB 3 eran");
        }
  }
```

Osagaia erabiliko duen bezero-aplikazioa honako hau izango da:

```
import javax.ejb.EJB;

public class KaixoPatxiBezeroa {
      @EJB
        private static KaixoPatxi kaixo;

       public  static void main(String[] args) {

       kaixo.kaixoPatxi("Patziku Armendariz");
       System.out.println("-- EJB osagaia ondo egikaritu da --");
        }
}
```

67. http://en.wikipedia.org/wiki/POJO

Kontu egin ezen aurreko konplexutasuna utzita, berriro, gure betiko Java *zaharrera* itzuli garela. `@Stateless` ezaugarri berria kenduta, beste guztia ezaguna eta arrunta zaigu. Argi dago programazio-eredua erabat sinplifikatu dela alde mekanikoa eta arrotza baztertuz. `@Stateless` oharkizuna erabilita, beanari buruzko hainbat informazio adieraziko du programatzaileak, semantikoki oso aberatsa izango dena, osagaiaren ezaugarriak adierazteko.

Bestalde, Hibernate eta antzeko teknologien eragina handia izan da iraunkortasun berria antolatzeko. Oharkizunak erabilita, objektuen eta taula erlazionalen arteko mapaketa dotore adierazita gelditzen da irakurgarria oso bihurtuz programazioaren arlo hori. Hona hemen, nola geldituko litzatekeen EJB 3 erabilita aurreko orrietan erabili den adibidearen domeinuaren ereduko aldiko eskaeraren iturburu-kodea:

```
import java.io.*;
import java.util.*;

import javax.persistence.*;


@SuppressWarnings("serial")
@Entity(access = AccessType.FIELD)
@Table(name = "ALDIKO_ESKAERA")

public class AldikoEskaera implements Serializable {


private int bertsioa;



    @Id(generate = GeneratorType.AUTO)
    private int id; // Hibernate

    private int egoera = AldikoEskaera.NEW;

    @Column(name = "BANATZE_TIME")
    private Date banatzeData;

    @ManyToOne
    @JoinColumn(name = "DENDA_ID")
    private Denda denda;

    @OneToMany(cascade = {  CascadeType.ALL } )
    @JoinColumn(name = "ALDIKO_ESKAERA_ID")

@org.hibernate.annotations.IndexColumn(name="ARTIKULUAK_INDIZEA")

@org.hibernate.annotations.Cascade(org.hibernate.annotations.CascadeTy
pe.DELETE_ORPHAN)
```

```
    private List<AldikoEskaeraArtikuluak> artikuluakAE = new
ArrayList<AldikoEskaeraArtikuluak>();

    @ManyToOne(cascade = CascadeType.PERSIST, targetEntity =
AbstractBeherapenaInpl.class)
    @JoinColumn(name = "beherapen_id")
    private Beherapena beherapena;

    @Embedded
    @AttributeOverrides( {
            @AttributeOverride(name = "kale1", column = @Column(name =
"BANATZE_KALE1")),
            @AttributeOverride(name = "kale2", column = @Column(name =
"BANATZE_KALE2")),
            @AttributeOverride(name = "hiri", column = @Column(name =
"BANATZE_HIRI")),
            @AttributeOverride(name = "egoera", column = @Column(name
= "BANATZE_EGOERA")),
            @AttributeOverride(name = "postaKodea", column =
@Column(name = "BANATZE_POSTAKODEA")) })
    private Helbidea banatzeHelbidea;


    public AldikoEskaera() {

    }

    /**
     * Aldatu kopuruak
     *
     * @param kopuruak
     * @exception InvalidAldikoEskaeraEgoeraException
     */
    public void eguneratuKopuruak(int[] kopuruak)
            throws InvalidAldikoEskaeraEgoeraException {
        if (getEgoera() != AldikoEskaera.DENDA_AUKERATUTA
                && getEgoera() != AldikoEskaera.PREST_BIDALTZEKO
                && getEgoera() != AldikoEskaera.PREST_KOKATU)
            throw new InvalidAldikoEskaeraEgoeraException();

        List<Artikulu> artikuluak = denda
                .getArtikuluak();

        if (artikuluak.size() != kopuruak.length)
            throw new InvalidAldikoEskaeraEgoeraException(
                    "Ez dute neurri bera. Eduki behar: "
                            + artikuluak.size()
                            + "beste hau du: "
                            + kopuruak.length);

        artikuluakAE.clear();
```

```
        Iterator it = artikuluak.iterator();

        for (int i = 0; i < kopuruak.length; i++) {
            int kopuru = kopuruak[ i] ;
            Artikulu Artikulu = (Artikulu) it.next();
            if (kopuru > 0) {
                artikuluakAE
                        .add(new AldikoEskaeraArtikuluak(
                                kopuru, Artikulu));
            }

        }

        setEgoera(egiaztatu() ? AldikoEskaera.PREST_BIDALTZEKO
                : AldikoEskaera.DENDA_AUKERATUTA);
    }

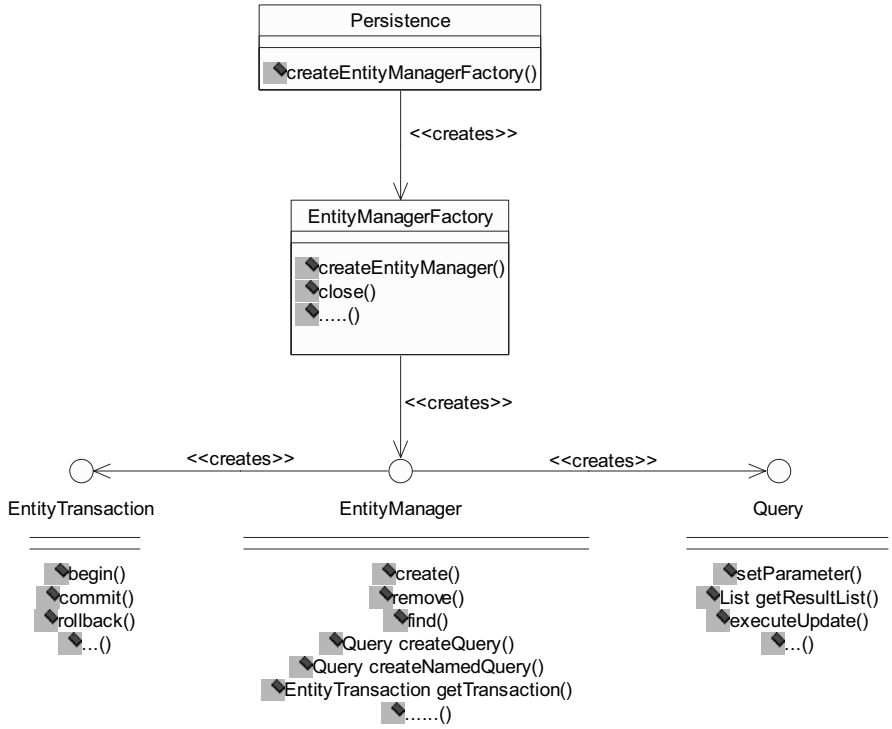
......}
```

Adierazi diren oharkizunek argi uzten dute nola eta norekin gauzatuko den mapaketa. Gure kasuan, `@Table` oharkizunak dio mapaketa `ALDIKO_ESKAERA` taularekin gauzatuko dela. Nola gauzatuko den mapaketa hori, `AldikoEskaera` klasearen barruan adierazi diren oharkizunek azaltzen dute. `@Id` oharkizunak non dagoen gako nagusia adierazten du eta EJB edukiontziari ohartarazten dio berak sortu behar duela. `@ManyToOne` eta `@OneToMany` oharkizunek, aldiko eskaerak izango dituen erlazioak adierazteko balio dute. Kontu egin ezen `@OneToMany`-rekin eskaeraren artikuluak kontrolatzen direla eta `@ManyToOne`-rekin eskaeraren denda eta aplikatu den beherapena. Klasearen eremuak erabilita, aldiko eskaeraren artikuluen kopuruak eguneratzen ditu `eguneratuKopuruak()` metodoak.

JDO edota Hibernate teknologien eragina ez da programazioaren arlora bakarrik mugatu espezifikazio berriaren iraunkortasunean, JPA (Java Persistence API) API berriaren arkitektura ere aipatutako teknologietan oinarritu da. **4.9. irudia**k lagunduko digu azaldutakoa argitzen.

Aplikazioa egikaritzean gauzatuko diren konexioak, transakzioak eta galderak irudian agertzen diren interfazeekin egituratuko dira. Hasteko, `Persistence` klasearen metodoekin, konexioen faktoria eraikiko da. Hori egin ondoren, `EntityManager` motako objektuaren metodoek entitate-bean motako objektuak eraiki, ezabatu eta bilatuko dituzte eta galderak antolatuko dituzte, Hibernate teknologian `Session` motako objektuaren metodoek eta JDOn `PersistenceManager` motako objektuaren metodoek egikaritzen zituzten bezala.

Ildo beretik, `Query` interfazea, JDOren edo Hibernateren `Query` interfazearen baliokidea da. Aplikazioak `Query` erako galdera bat eraiki nahi duenean, `EntityManager`-en faktoria-metodoak (`createQuery()` eta `createNamedQuery()`) egikarituko ditu. `Query` interfazearen metodoekin galderaren ezaugarriak kontrolatuko ditu eta emaitzak eskuratu.



4.9. irudia. EJB 3 espezifikazioaren iraunkortasunaren arkitektura nagusia.

`EntityTransaction` motako objektuaren metodoek transakzioak kudeatzeko balio dute eta `EntityManager`-en metodoak erabilita kudeatzen ditu aplikazioak. Interfaze hori, JDO edo Hibernateren `Transaction` interfazearen baliokidea da.

Iraunkortasun berrian, osagaien arteko herentzia, arazo gabe programatu daiteke `@Inheritance` oharkizuna erabilita. Demagun, aldiko eskaerei beherapen mota desberdinak aplikatu nahi zaizkiela eta hori egituratzeko, domeinuaren Klase Diagramak herentzia aplikatzea eskatzen duela. Ebazpena gauzatzen hasteko, interfaze bat eta klase abstraktu bat antolatzen dira lehendabizi. Lehenengoarekin, oinarrizko funtzionalitatea adierazten da eta, bigarrenarekin, mapaketaren nondik norakoak:

```
import java.io.*;

public interface Beherapena extends Serializable {
    public double getBanatzeBeherapen(AldikoEskaera eskaera);

    public double getBanatzeZati(AldikoEskaera eskaera);

    public String getGako();
```

```
    public int getId();
}

/*
@Entity(access = AccessType.FIELD)
@Table(name = "BEHERAPENA")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE,
discriminatorType = DiscriminatorType.STRING)
@DiscriminatorColumn(name = "BEHERAPENA_MOTA", nullable = true)

public abstract class Beherapena implements Serializable {

    @Id(generate = GeneratorType.AUTO)
    private int id;

    private String gako;

    public Beherapena() {
    }

    private Beherapena(String gako) {
        this.gako = gako;
    }

    public String getGako() {
        return gako;
    }

    public abstract double getBanatzeBeherapen(
            AldikoEskaera aldikoEskaera);

    public abstract double getBanatzeZati(
            AldikoEskaera aldikoEskaera);

}
```

Bi osagai horiekin, oinarrizko azpiegitura finkatuta gelditzen da eta orain gelditzen dena da: interfazea inplementatzea eta oinarrizko inplementazio hori hedatzea beherapen mota bakoitzaren ezaugarriak adierazteko. Lehenengo eragiketa, `AbstractBeherapenaInpl` klase abstraktuak gauzatuko du:

```
import javax.persistence.*;

@Entity(access = AccessType.FIELD)
@Table(name = "BEHERAPENA")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE,
discriminatorType = DiscriminatorType.STRING)
@DiscriminatorColumn(name = "BEHERAPENA_MOTA", nullable = true)

public abstract class AbstractBeherapenaInpl implements Beherapena
{
```

```
    @Id(generate = GeneratorType.AUTO)
    private int id = -1;

    private String gako;

    protected AbstractBeherapenaImpl() {

    }

    protected AbstractBeherapenaInpl(String gako) {
        this.gako = gako;
    }

    public String getGako() {
        return gako;
    }

    public int getId() {
        return id;
    }
}
```

Kontu egin ezen `@DiscriminatorColumn` oharkizunean adierazten dela beherapen mota eta `BEHERAPEN_MOTA` zutabeak biltegiratuko duela. Beherapen mota bakoitzaren semantika berezia klase berri batean adieraziko da eta, hutsetik ez garenez hasiko, `AbstractBeherapenaInpl` klase sortu berriaren ezaugarriak heredatuko dira:

```
import javax.persistence.*;

@Entity (access=AccessType.FIELD)
@Inheritance(strategy=InheritanceType.SINGLE_TABLE,
        discriminatorType = DiscriminatorType.STRING,
discriminatorValue="GARRAIO_MERKE")

public class GarraioBeherapena extends AbstractBeherapenaInpl {

    private double kopuruTxiki;

    public GarraioBeherapena() {
    }

    public GarraioBeherapena(String gako, double kopuruTxiki) {
              super(gako);
              this.kopuruTxiki = kopuruTxiki;
      }

       protected double getKopuruTxiki() {
              return kopuruTxiki;
       }
```

```
    public double getBanatzeBeherapen(AldikoEskaera eskaera) {
        double guztira = eskaera.getGuztira();
        if (guztira < getKopuruTxiki())
          return 0;
        else
          return eskaera.getBanatzeBalio();
    }

    public double getBanatzeZati(AldikoEskaera eskaera) {
        return 0;
    }
}
```

Aurreko orrialdeetan azaldu den legez, espezifikazio berrian eraiki den APIak zeharo estandarizatu eta erraztu du entitate-bean motako osagaien iraunkortasuna. Gainera, JPA APIaren egitura banatuta dagoenez, ebazpen bat baino gehiago aztertu eta aukeratu daitezke aplikazioa inplementatzean. Adibidez, posible da, BEA WebLogic edukiontzian, Hibernate teknologia txertatzea iraunkortasuna kudeatzeko, iraunkortasun-hornitzaile horrekin lan egin nahi bada aplikazioan eta ez, WebLogic berak dakarren Kodo inplementazioarekin. Agur, beraz, neurrira egindakoari eta ongi etorri *pret-a-porter*-ari.

Amaitzeko, iraunkortasunaren arlo hau adibide txiki batekin garatuko da zeinak programatu den aldiko eskaera eta azaldu den JPA APIa elkarlanean jarriko dituen. Horretarako, aipatutako aldiko eskaerarentzat gordailu bat diseinatu eta programatuko da, aldiko eskaerak eraiki eta bilatuko dituena:

```
import java.util.*;

import javax.ejb.*;
import javax.persistence.*;

@Stateless
public class EJB3AldikoEskaeraGordailu implements
        AldikoEskaeraGordailu {

    @PersistenceContext
    private EntityManager entityManager;

    public EJB3AldikoEskaeraGordailu() {
    }

    public EJB3AldikoEskaeraGordailu(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    public AldikoEskaera eraikiAldikoEskaera() {
        AldikoEskaera aldikoE = new AldikoEskaera();
        entityManager.persist(aldikoE);
```

```
        return aldikoE;
    }

    public AldikoEskaera bilatuAldikoEskaera(String
aldikoEskaeraId) {
        return (AldikoEskaera) entityManager.find(
                AldikoEskaera.class, new Integer(aldikoEskaeraId));
    }

    public AldikoEskaera bilatuEraikiAldikoEskaera(
            String aldikoEskaeraId) {
        if (aldikoEskaeraId == null)
            return eraikiAldikoEskaera();
        return bilatuAldikoEskaera(aldikoEskaeraId);
    }


    public AldikoEskaera eraikiAldikoEskaera(Helbidea
banatzeHelbidea,
            Date banatzeData, Denda denda,
            List artikuluak) {
        return null;
    }  }
```

Diseinatu den osagaia egoera gabeko beana da, `EntityManager` motako objektua erabiliko duena iraunkortasuna bermatzeko. Hori guztia adierazteko, nahikoa da `@Stateless` eta `@PersistenceContext` oharkizunak erabiltzea. Lehenengoarekin, osagaia, sesio-beana dela adierazten da eta, gainera, egoera gabekoa. Bigarrenarekin, `EntityManager` motako osagaia injektatzen zaio beanari, iraunkortasuna txukun eta arazo barik kudeatzeko.

EJB 2 espezifikazioan oso zaila zen probak bideratutako garapen mota erabiltzea, proba bakoitzaren ezarpena, edukiontziaren barnean gauzatzen zelako. Espezifikazio berriarekin, gauzak zeharo aldatu dira, entitate-beanak, edukiontziaren kanpoan egikaritu baitaitezke. Horrela, posible da JUnit erako probak diseinatzea JDOrekin edota Hibernate teknologiarekin egiten den bezala. Hona hemen adibide txiki bat:

```
import java.util.*;

import javax.persistence.*;

import junit.framework.*;
import org.hibernate.ejb.*;

public class EJB3AldikoEskaeraGordailuTests extends TestCase {

       private EntityManager em;

       private EntityTransaction transaction;
       private EntityManagerFactory emf;
```

```
	protected String aldikoEskId;
	protected String dendaId;

	private AldikoEskaera aldikoE;

	@Override
	protected void setUp() throws Exception {
		super.setUp();
		Properties props = new Properties();

		props.setProperty(Persistence.PERSISTENCE_PROVIDER,
				HibernatePersistence.class.getName());

		emf = Persistence.createEntityManagerFactory("pia", props);
	}

	@Override
	protected void tearDown() throws Exception {
		super.tearDown();
		if (emf != null)
			emf.close();
	}

	protected void doWithTransaction(TxnCallback cb) throws
Throwable {

		em =
emf.createEntityManager(PersistenceContextType.EXTENDED);

		transaction = em.getTransaction();
		try {
			transaction.begin();
			cb.execute();
			transaction.commit();
		} finally {
			if (transaction != null && transaction.isActive())
				transaction.rollback();
			if (em != null)
				em.close();
		} }

	private void hasieratuTestData() throws Throwable {
		doWithTransaction(new TxnCallback() {

			public void execute() throws Exception {

			    aldikoE = new AldikoEskaera();
			    em.persist(aldikoE);
			    aldikoEskId = aldikoE.getId();

			    Denda denda = DendaNagusi.eginDenda();
			    for (BanatzeKodea bk : denda.getZerbitzuArea())
					em.persist(bk);
```

```
                        em.persist(denda);

                        dendaId = denda.getId();

                    }
                });
        }

        private void eguneratuBanatzeHelbidea() throws Throwable {
                doWithTransaction(new TxnCallback() {

                    public void execute() throws Exception {

                        DendaGordailu dg = new EJB3DendaGordailu(em);
                        aldikoE = bilatuAldikoEskaera();
                        assertEquals(AldikoEskaeraEgoeraGakoak.ADOS,
aldikoE.eguneratuBanatzeInfo(
                                dg, DendaTestData.getHELBIDE1(),
                    DendaTestData.eginOndoBanatzeData(), false));

                    }
                });
        }


        public void test() throws Throwable {

                hasieratuTestData();

                eguneratuBanatzeHelbidea();
          .........

        }  }
```

Proba horrekin, `AldikoEskaera` osagaiaren portaera zehatza aztertu nahi da eta, horretarako, proba bat diseinatu da banatze-helbidea eguneratuko duena. Lehenengo eta behin, `Properties` motako objektua eraikiko du `setUp()` metodoak, iraunkortasun-hornitzailea adierazteko. Hasieraketa amaitzeko, `EntityManagerFactory` motako objektua eraikiko da, `Persistence` objektuaren `createEntityManagerFactory`() metodo estatikoa egikarituta. Proba amaitzeko, `EntityManagerFactory` motako objektua itxi egiten du `tearDown()` metodoak, faktoriaren `close()` metodoa erabilita.

`doWithTransaction()` metodoak `EntityManager` osagaia eraikitzen du faktoriaren `createEntityManager()` metodoa egikarituta. Azpiegitura osoa prestatuta, transakzioa jartzen du martxan metodoak. Gero, callback-a egikaritzen du eta, amaitzeko, commit-a gauzatzen du. Arazoak aurkitzen badira prozesuan, rollback-a egikaritzen du.

Probaren azpiegitura orokorra azalduta goazen probaren muina azaltzera:

- `hasieratuTestData()` metodoak `AldikoEskaera` eta `Denda` motako osagaiak eraiki eta biltegiratuko ditu, `EntityManager` objektua erabilita. Horrela, proba gauzatzeko behar duen informazioa, biltegiratuta gelditzen da.
- `eguneratuBanatzeHelbidea()` metodoak eguneratu nahi den banatze-informazioa eguneratzen du, `AldikoEskaera` osagaiaren `eguneratuBanatzeInfo()` metodoa egikarituta. `assertEquals()` metodoak ziurtatzen du proba ondo gauzatu dela.

Orain arte aztertu denarekin, azaldu da nola eraiki eta erabili aplikazioaren domeinuko osagaiak. Espezifikazio berriak zeharo aldatu du iraunkortasunaren trataera, estandarizatuz eta erraztuz osagaien kudeaketa. Goazen orain, aplikazioaren domeinuaren logika antolatzera.

Aplikazioaren domeinuaren logikan aldaketa kontzeptual gutxi aurkituko dira. Aurreko espezifikazio zaharrean legez, domeinuaren logika, sesio-beanak erabilita antolatuko da. Aurreko orrietan egin den bezala, hemen ere, Façade patroia erabiliko da aurkezpen mailaren eta domeinuaren logikaren arteko komunikazioa eratzeko. Komunikazio hori egituratzeko, jakin badakigu, hiru urrats ezagun erabiliko direla:

1. Erabiltzailearen interfaze grafikoak eraiki aurkezpen maila diseinatuta.
2. Aplikazioaren domeinuko entitateetara atzitu sesio-beanaren metodoak programatuta.
3. Aurkezpen mailak behar duen informazioa antolatu `ResultFaktoria` motako objektua diseinatuta.

Lehenengo bi urratsak, aurreko beste atal batean azaldu direnez eta ezagunak zaizkigunez, ez dira azalduko. Goazen, beraz, hirugarren urratsa azaltzera lehen erabili den adibidearekin, hau da, eskaera egin prozesuarekin. Lehen egin den bezala, orain ere, klase bat diseinatu eta programatu behar du programatzaileak, `Result` erako objektuak eraikiko dituena. Objektu mota horiek, aplikazioaren DTOak izango dira eta aurkezpen mailari bidaliko zaizkio era egituratuan. Horrela, aurkezpen mailak ez du arduratu behar objektuen grafoa berreraikitzeaz domeinuaren logikak bidaltzen dionean informazioa, eta, zuzenean erabil dezake informazioa interfaze grafikoa elikatzeko. Hona hemen eskaera egin prozesuak erabiliko duen `ResultFaktoria` motako osagaia, aipatutako informazioa egituratzeko:

```
import java.util.*;

import javax.ejb.*;

@Stateless
public class EJB3EskaeraEginFacadeResultFaktoria
        implements EskaeraEginFacadeResultFaktoria {

    public EJB3EskaeraEginFacadeResultFaktoria() {

    }

    public EskaeraEginFacadeResult egin(int egoeraGako,
            AldikoEskaera aldikoEskaera, List dendak) {
        hasieratuAldikoEskaera(aldikoEskaera);
        return new EskaeraEginFacadeResult(egoeraGako,
                aldikoEskaera, dendak);
    }

    public EskaeraEginFacadeResult egin(int egoeraGako,
            AldikoEskaera aldikoEskaera) {
        hasieratuAldikoEskaera(aldikoEskaera);
        return new EskaeraEginFacadeResult(egoeraGako,
                aldikoEskaera);
    }

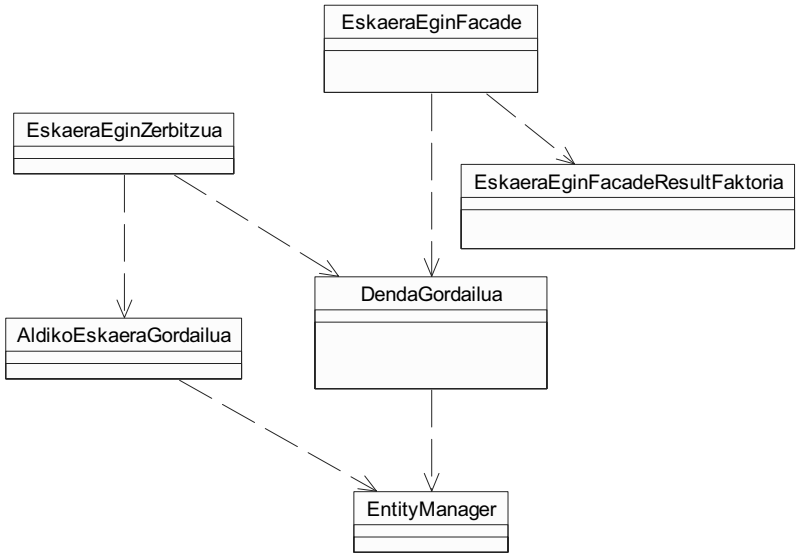
    private void hasieratuAldikoEskaera(
            AldikoEskaera aldikoEskaera) {
        Denda denda = aldikoEskaera
                .getDenda();
        if (denda != null) {
            Artikulu artikulu = denda
                    .getArtikuluak().get(0);
            artikulu.getIzena();
        }
        List<AldikoEskaeraArtikulu> artikuluak = aldikoEskaera
                .getArtikuluak();
        if (artikuluak != null && !artikuluak.isEmpty()) {
                                artikuluak.get(0);
        } }  }
```

Lehendabizi, `egin()` erako metodoak erabilita, ziurtatu egiten du osagaiak ezen behar dituen datuak ondo eskuratzen direla `hasieratuAldikoEskaera()` metodoa egikarituta. Bukatzeko, `EskaeraEginFacadeResult` motako objektu bat itzultzen dio metodo bakoitzak bezeroari. Horrela, behar duen informazioa egituratuta eta objektu bakarrean jasotzen du bezeroak.

Ohiko egoeretan, informazio mota bat baino gehiago kudeatu behar ditu Façade patroiak, hala nola `ResultFaktoria` motakoak, domeinuaren zerbitzuak edota gordailuak. Horietako bakoitza, bere aldetik, beste osagai batzuekin jar daiteke harremanetan, eta objektuz osatutako grafo konplexuak eratu. Adibidez,

aztertzen ari garen eskaera egin prozesuak **4.10. irudia**n azaltzen den grafoa erabiltzen du Façade patroiaren informazioa antolatzeko. Argi dago patroia ondo egituratu nahi bada, osagaiak bildu eta elkarrekin harremanetan jarri behar dituela. Hiru estrategia erabil daitezke osagaiak biltzeko eta atzitzeko:

1. EJBren baliabide hornitzailearen (Dependency Injection) teknika erabili osagaiak txertatzeko toki egokietan.
2. Spring teknologia erabili bitartekari-lanak egiteko, hau da, osagaiak eraikitzeko eta antolatzeko, eta gero, EJB teknologian integratu sortutako egitura.
3. Spring teknologia erabili lan gehiena egiteko. Azken egoera honetan EJB teknologia bezero arrunta bihurtuko da Façade patroia inplementatzen duen sesio-beanak, zuzenean deituko dio Spring teknologiaren bean-faktoriari.



4.10. irudia. Adibidearen informazioa antolatzeko klase nagusiak.

Lehenengo estrategiarekin, nahikoa da `@EJB` oharkizunak erabiltzea baliabideak injektatu nahi direnean osagaietan. Hala, aztertzen ari garen Façade patroia, estrategia hori erabilita inplementatu nahi bada, honako iturburu-kodearekin nahikoa izango da osagaiak txertatzeko sesio-beanetan:

```
@Stateless
public class EskaeraEginFacadeInpl implements EskaeraEginFacade {

    @EJB
    private DendaGordailu dendaGordailu;

    @EJB
    private EskaeraEginFacadeResultFaktoria resultFaktoria;
```

```
    @EJB
    private EskaeraEginZerbitzua zerbitzua;

    public EskaeraEginFacadeInpl(EskaeraEginZerbitzua zerbitzua,
            DendaGordailu dendaGordailu,
            EskaeraEginFacadeResultFaktoria resultFaktoria) {
        this.zerbitzua = zerbitzua;
        this.dendaGordailu = dendaGordailu;
        this.resultFaktoria = resultFaktoria;
    }
..................
    }

import java.util.*;

import javax.annotation.*;
import javax.ejb.*;


@Stateless
public class EskaeraEginZerbitzuInpl implements
        EskaeraEginZerbitzua {

    @EJB
    private AldikoEskaeraGordailu aldikoEskaeraGordailu;

    @EJB
    private DendaGordailu dendaGordailu;

    private EskaeraEginZerbitzuInpl() {
    }

    public EskaeraEginZerbitzuInpl(
            AldikoEskaeraGordailu aldikoEskaeraGordailu,
            DendaGordailu dendaGordailu) {
        this.aldikoEskaeraGordailu = aldikoEskaeraGordailu;
        this.dendaGordailu = dendaGordailu;
    }

    //
    AldikoEskaera bilatuEraikiAldikoEskaera(
            String aldikoEskaeraId) {
        if (aldikoEskaeraId == null)
            return aldikoEskaeraGordailu
                    .eraikiAldikoEskaera();
        else
            return aldikoEskaeraGordailu
                    .bilatuAldikoEskaera(aldikoEskaeraId);
    }
..................
    }
```

Bistan da, ebazpena, sinplea eta trinkoa dela. Oharkizun bakoitzak EJB edukiontziari ohartarazten dio sesio-beanarekin hasieratu behar duela eremua.

Lehenengo estrategia interesgarria da EJB edukiontzia nahikoa bada aplikazioa inplementatzeko, hau da, baliabide guztiak EJB edukiontziak dituenean. Alabaina, batzuetan, Spring teknologiaren baliabideez elikatu nahi da aplikazioa, kalitatea eta eraginkortasuna ziurtatzeko. Egoera horretan, Spring eta EJB teknologiek ondo integratuta egin behar dute lan eta hori da, hain zuzen, bigarren estrategiarekin aztertuko dena.

Integrazioa gauzatzeko hiru eragiketa egikarituko dira:

1. Aplikazioa hasieratzen denean, Spring teknologiak sortutako beanak ikusgai utzi, JNDI teknologia erabilita.
2. Oharkizunak jarri sesio-beanetan, EJB edukiontzia ohartarazteko ezen Spring teknologiak sortutako beanak injektatu behar dituela sesio-beanetan.
3. `EntityManager` motako objektua injektatu aplikazioaren gordailuetan.

Lehenengo eragiketa gauzatzeko, `Reference` motako objektua eraikitzen da eta `bind()` metodoa erabilita, izen batekin txertatzen da JNDI zuhaitzean. Hona hemen adibide txiki bat:

```
Context ctx = new InitialContext();
Reference reference = new Reference(
                EskaeraEginZerbitzua.class.getName(),
                new StringRefAddr("beanName", "EskaeraEginZerbitzua"),
                SpringObjectFactory.class.getName(),
                null);
  ctx.bind("EskaeraEginZerbitzua",reference);
```

Kontu egin ezen metodo eraikitzaileari, klase nagusia, Spring teknologiaren beana eta JNDI objektu-faktoria pasatzen zaizkiola, `Reference` motako objektua eraikitzeko.

JNDI inplementazioak erreferentzia hori aurkitzen duenean, `SpringObjectFactory` motako objektuaren `getObjectInstance()` metodoa egikaritzen du Spring beana eraikitzeko, Spring teknologiaren bean-faktoria erabilita:

```
import java.util.*;

import javax.naming.*;
import javax.naming.spi.*;

import org.apache.commons.logging.*;

public class SpringObjectFactory implements ObjectFactory {
```

```
       private Log logger = LogFactory.getLog(getClass());

       public Object getObjectInstance(Object reference, Name name,
                     Context nameCtx, Hashtable<?, ?> environment)
throws Exception {
              String beanName = (String) ((Reference)
reference).get(0).getContent();
              logger.debug("Norbait bean-ekin ari da: " + beanName);
              logger.debug("Erabili: " + "ejb3-spring-context.xml");
              try {
                     return TheBeanFactory.getBean("ejb3-spring-
context.xml",
                                   "spring-context.xml", beanName,
Object.class);
              } catch (RuntimeException e) {
                     logger.error(e);
                     throw e;
              } catch (Exception e) {
                     logger.error(e);
                     throw e;
              }  }   }
```

Bigarren eragiketa aztertzeko, demagun, lehenago azaldu den eskaera egin prozesuaren Façade patroiaren inplementazio berria programatzen dela bigarren estrategia hori erabilita:

```
@Stateless
@Depends("jboss.j2ee:service=EJB3,type=service,
name=SpringBeanReferenceInitializerLocal")

@PersistenceContext({
@PersistenceContext(name="EntityManager") })

public class EskaeraEginFacadeIntegraDepenInjectInpl
     implements
     EskaeraEginFacadeIntegraDepenInject {

     @Resource(name= "DendaGordailu")
     private DendaGordailu dendaGordailu;

       @Resource(name= "EskaeraEginFacadeResultFaktoria")
     private EskaeraEginFacadeResultFaktoria resultFaktoria;

       @Resource(name= "EskaeraEginZerbitzua")
     private EskaeraEginZerbitzua zerbitzua;
......
}
```

`@Resource` oharkizun bakoitzak EJB edukiontzia ohartarazten du adieraziz osagaia injektatu behar duela sesio-beanaren eremu hori hasieratzeko. `name` parametroaren balioa, JNDI izena da osagaia bilatzeko, JNDI zuhaitzean.

`@Depends` oharkizunak ziurtatzen du osagaien arteko mendekotasunak ondo egikarituko direla, hau da, JNDI erreferentziak aurretik eratuko direla eta, gero, EJB osagaien ezarpenak.

`@PersistenceContext` oharkizunarekin, JNDI izena eransten zaio `EntityManager` objektuari, hirugarren eta azken eragiketa arazorik gabe egikaritzeko.

Aplikazioaren gordailuetan, azken eragiketa gauzatu behar du programatzaileak, `EntityManager` motako objektuak injektatzeko. Hori egiteko, Spring teknologiaren bean definizioak programatu behar ditu eta han, Springen `FactoryBean` baliabidea erabili, JNDI objektuak bilatzeko:

```
<beans>

.....
<bean id="EntityManager"
    class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name= "jndiName">
   <value>java:comp.ejb3/env/EntityManager</value>
   </property>

   <property name="lookupOnStartup">
     <value>false</value>
   </property>

   <property name="proxyInterface">
     <value>javax.persistence.EntityManager</value>
   </property>
 </bean>

<bean id="AldikoEskaeraGordailuaInpl"
    class="EJB3AldikoEskaeraGordailua">
    <constructor-arg ref="EntityManager"/>
 </bean>
....
</beans>
```

Hiru eragiketak egikaritzen direnean aplikazioan, Spring teknologiak osagaiak eraikitzen ditu eta haien arteko mendekotasunak kudeatzen ditu. EJB edukiontziak, bere aldetik, eraikitako osagaiak aplikazioaren beanetan injektatzen ditu.

Hirugarren estrategiarekin, sesio-beanak, zuzen-zuzenean deitzen dio Spring teknologiaren bean-faktoriari behar dituen osagaiak instantziatzeko. Estrategia horrekin sesio-bean bakarra, Façade patroia inplementatzen duen beana izango da. Beste guztiak, POJO objektuak dira, aplikazioan txertatuko direnak.

Hona hemen gure adibidearen inplementazioa, azken estrategia hori erabilita:

```
@Stateless
public class EskaeraEginFacadeInplSpring implements
EskaeraEginFacade {

    private DendaGordailu dendaGordailu;

    private EskaeraEginFacadeResultFaktoria resultFaktoria;

    private EskaeraEginZerbitzua zerbitzua;

    public EskaeraEginFacadeInplSpring() {
            }

   @PostConstruct
   public void createOsagaiak (
     EntityManager entityManager)  {

       this.dendaGordailu =
(DendaGordailu)TheBeanFactory.getBean("DendaGordailu",
      DendaGordailu.class);

       this.resultFaktoria =
(EskaeraEginFacadeResultFaktoria)TheBeanFactory.
getBean("EskaeraEginFacadeResultFaktoria",
      EskaeraEginFacadeResultFaktoria.class);

       this.zerbitzua =
(EskaeraEginZerbitzua)TheBeanFactory.
getBean("EskaeraEginZerbitzua",
      EskaeraEginZerbitzua.class);
}
```

`createOsagaiak()` metodoak Spring teknologiaren bean-faktoriari deitzen dio sesio-beanak behar dituen osagaiak eraikitzeko. Arazoak ekiditeko, `@PostConstruct` oharkizunaren bidez, EJB edukiontziak badaki metodo hori egikaritu behar duela beana instantziatu ondoren. Horrela ziurtatuta gelditzen da, behar dituen hasieraketak ondo egikaritzen direla.

4.6. OBJEKTUETARA ZUZENDUTAKO IRAUNKORTASUNA: DB4O

Objektuetara zuzendutako garapenak beti izan du arazo larri bat iraunkortasunarekin. Aplikazioa garatzeko erabiltzen den programazio-ingurunea objektuetan oinarritzen da baina aplikazio horretako objektuei iraunkortasuna aplikatu nahi zaienean eredu erlazionalean biltegiratzen dira.

Datuak ez direnean konplexuak edota objektuen egiturak ez direnean sakonak, datu-bihurketa horrek ez du arazo larririk sortzen eta ingurune biak, elkarrekin bizi daitezke, tentsio txikiak sortuz, iraunkortasunaren kudeaketarekin. Egoerarik onenean, programatzaileak kuboak txertatu behar ditu zilindroetan.

Desdoitze-arazoak areagotu egiten dira aplikazioak objektu konplexuekin lan egin behar duenean. Biltegiratze-ordua heltzen denean, irtenbide tabulatua aurkitzea objektu horiei, ez da erraza izaten. Batzuetan, bilduma bat du objektuak bere barnean elementu iraunkor bezala, beste batzuetan, beste objektu bat hedatzen du. Egoera horietan, datuak, nabigazio bidez eskuratzen dira eta ez bilaketa bidez. Hau da, iraunkortasuna eredu erlazionala erabilita gauzatzen denean, mapaketa egikaritu behar du ingurune horien artean.

Programatzaileak jostun-lanetan aritu behar du objektuak biltegiratu edota berreskuratu nahi dituenean. Iraunkortasunaren gardentasun ezak arazo erantsia dakarkio berez konplexua den aplikazioari. Programatzaileak baliabide eta teknika berriak erabili eta menderatu beharko ditu arazoari aurre egiteko. Gauak esna pasatzen ditu objektuetara zuzendutako datu-base horretan amesten. Zero-administrazio eta gardena, sumatu ere ez duena egiten martxan dagoenean baina txintxo-txintxo biltegiratzen dituena objektuak eta, are txintxoago, eskuratu. Baina gau batean, ezusteko zoragarri batekin esnatzen du patuak, satoriraino eramanez. Hona hemen eskatutako eta amestutako jostailutxo berria: db4o[68].

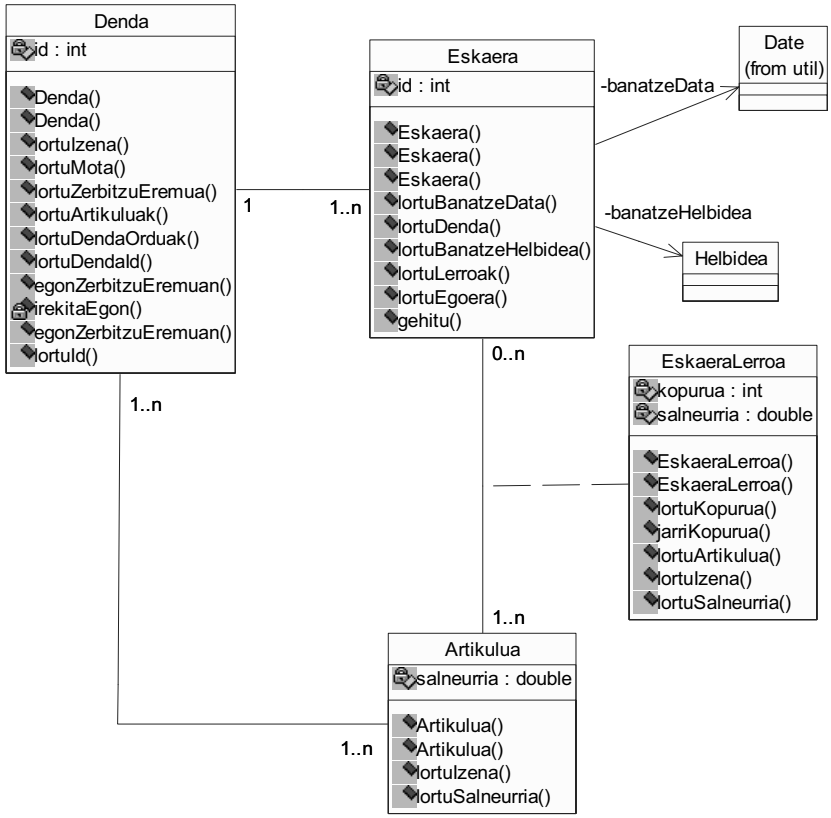
db4o teknologia erabilita eraikitzen diren datu-baseak, objektuetan oinarritzen dira erabat. Beraz, aplikazioarekin sortu behar duen elkarrizketa, erabat gardena eta zuzena da, ez baitu inongo datu-bihurketarik gauzatu behar. Ingurune artean eraiki behar duen elkarrizketa objektuak erabilita eratzen da eta ez da inongo mapaketarik egikarituko iraunkortasuna ziurtatzeko.

Objektuetara zuzendutako db4o teknologiaren ezaugarri nagusiak aztertzeko, aurreko orrietan eta teknologia desberdinekin aztertu den aplikazio informatiko bera erabiliko da atal honetan.

Gogoratu ezen aplikazio informatikoak bezeroaren eskaerak kudeatzen zituela denda egokia aukeratuta, eskaeraren artikuluak hornitzeko. **4.11. irudia**k islatzen du nola dauden diseinatuta aplikazioaren domeinuaren Klase Diagramaren klase nagusiak.

Kontu egin ezen klaseek, bildumak eta klase-egitura konplexuak erabiltzen dituztela domeinuaren betekizunak antolatzeko.

68. http:// www.db4o.com



4.11. irudia. Aplikazioaren domeinuaren Klase Diagrama nagusia.

Jakinda nola eratuko ditugun klaseak, has gaitezen aplikazioaren iturburu-kodea programatzen:

```
import java.util.*;
import java.io.File;
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import java.util.Date;
import com.db4o.inside.query.*;
import com.db4o.query.*;


public class Nagusi {
    public static void main(String[] args) {

        new File("C:/froga-5-2.yap").delete();
       Db4o.configure().messageLevel(0); // 0=silent, 4=loud
       ObjectContainer db = Db4o.openFile("C:/froga-5-2.yap");
```

```
    try {
     DendaOrdua d1,d2,d3;
     Hornitzailea ho1;
     Artikulua a1,a2,a3;
     Helbidea h1,h2,hEskaera;
     Denda dendaNagusia, denda1, denda2;
     String bezeroa = new String("Joxepo Matrakas");

     Eskaera eskaera;
     Date banatzeData = new Date(2007,4,14);

     d1 = new DendaOrdua(1,8,30,17,30);
     d2=  new DendaOrdua(2,9,30,18,30);
     d3=  new DendaOrdua(3,8,00,19,30);

     db.set(d1);
     db.set(d2);
     db.set(d3);

     ho1 = new Hornitzailea("HHHHHHH","Txikia");

     a1=  new Artikulua("AAAAAAAAA",35,ho1);
     a2=  new Artikulua("BBBBBBBBB",100,ho1);
     a3=  new Artikulua("DDDDDDDDD",200,ho1);

         db.set(a1);
         db.set(a3);
         db.set(a2);
         ObjectSet emaitza = (ObjectSet) db.get(new Artikulua());
         listArtikuluak(emaitza);  // eskuratu artikuluak

         emaitza.reset(); // reset ObjectSet-aren kurtsorea
         listEmaitza(emaitza);   // eskuratu artikuluak Object eran
         emaitza.reset();

h1 = new Helbidea("Kale1","Kale2","Bilbo","Bizkaia","1111");
h2 = new Helbidea("Kale10","Kale20","Lekeitio","Bizkaia","2222");
db.set(h1);
db.set(h2);
hEskaera = new Helbidea("KaleE0","KaleE1","Ea","Bizkaia","3333");
db.set(hEskaera);

    List artikuluak  = new ArrayList();
    List artikuluak1 = new ArrayList();
    List artikuluak2 = new ArrayList();

    Set dendaOrduak = new HashSet();
    Set helbideak   = new HashSet();
```

```
        artikuluak.add(a1);
        artikuluak.add(a2);
        artikuluak.add(a3);

        artikuluak1.add(a1);

        artikuluak2.add(a1);
        artikuluak2.add(a2);

        dendaOrduak.add(d1);
        dendaOrduak.add(d2);
        dendaOrduak.add(d3);

        helbideak.add(h1);
        helbideak.add(h2);

    dendaNagusia =
      new Denda("DDDDD","aaa",helbideak,dendaOrduak,artikuluak);

    denda1 =
      new Denda("D1111","aaa",helbideak,dendaOrduak,artikuluak1);

    denda2 =
      new Denda("D2222","aaa",helbideak,dendaOrduak,artikuluak2);


       db.set(dendaNagusia);
       db.set(denda1);
       db.set(denda2);

      eskaera = new Eskaera(hEskaera,banatzeData,dendaNagusia,
bezeroa);
      eskaera.gehitu(a1,10);
      eskaera.gehitu(a2,100);
      eskaera.gehitu(a3,1000);

       db.set(eskaera);

      // Gehitu entzulea galderari –optimizazioa aztertzeko

((com.db4o.YapStream)db).getNativeQueryHandler().addListener(new
Db4oQueryExecutionListener() {
            public void notifyQueryExecuted(NQOptimizationInfo info) {
                System.out.println(info.toString());
                System.out.println();
            }
        });


    System.out.println("--- QUERY BY EXAMPLE(QBE) ADIBIDEA ---");
    System.out.println("----------------------");
```

```
       Denda dendaProba = new Denda();
       dendaProba.gehitu(artikuluak);
       emaitza = db.get(dendaProba);
       listDenda(emaitza);

      System.out.println("--- NATIVE QUERY(NQ) ADIBIDEA ---");
      System.out.println("--------------------");

     ObjectSet artikuluakNQ = (ObjectSet)db.query(new
Predicate<Artikulua>() {
            public boolean match(Artikulua artikulua) {
                return artikulua.lortuSalneurria() > 40;
            }
        });

      listArtikuluak(artikuluakNQ);  // eskuratu artikuluak
      artikuluakNQ.reset();

         // Prestatu Comparator
         Comparator<Artikulua> artikuluCmp=
                 new Comparator<Artikulua>() {
             public int compare(Artikulua art1, Artikulua art2) {
             Double art11 = new Double(art1.lortuSalneurria());
             Double art22 = new Double(art2.lortuSalneurria());
         return art11.compareTo(art22);
             }
         };

       artikuluakNQ = (ObjectSet)db.query(new Predicate<Artikulua>()
{
            public boolean match(Artikulua artikulua) {
                return true;
            }
        }, artikuluCmp);


      listArtikuluak(artikuluakNQ);  // eskuratu artikuluak
      artikuluakNQ.reset();


           // SODA galderak eratzen:
          // -------------
          System.out.println("--  SODA GALDERAK   --");

         Query galdera1 = db.query();
         galdera1.constrain(Artikulua.class);
         galdera1.descend("izena").constrain("BBBBBBBBB");
         ObjectSet emaitza1 = galdera1.execute();
 System.out.println("------------------------");
         listArtikuluak(emaitza1);
```

```
        Query galdera2 = db.query();
        galdera2.constrain(Artikulua.class);
        Constraint lehenConstr=
galdera2.descend("salneurria").constrain(200);// DD artikulua

galdera2.descend("izena").constrain("BBBBBBBBB").or(lehenConstr);
        ObjectSet emaitza2 = galdera2.execute();
System.out.println("-----------------------");
        listArtikuluak(emaitza2);

        Query galdera3 = db.query();
        galdera3.constrain(Artikulua.class);
        galdera3.descend("izena").orderDescending();
        ObjectSet emaitza3 = galdera3.execute();
System.out.println("-----------------------");
        listArtikuluak(emaitza3);



        } finally {
            db.close();
        }
            }

    public static void listEmaitza(ObjectSet emaitza) {
        while (emaitza.hasNext()) {
            System.out.println(emaitza.next());
        }
System.out.println("-----------------------");
    }

    public static void listArtikuluak(ObjectSet emaitza) {
        while (emaitza.hasNext()) {
             Artikulua artikulu = (Artikulua)emaitza.next();
    System.out.print("Izena:  " + artikulu.lortuIzena());
    System.out.println("   Salneurria:  " +
artikulu.lortuSalneurria());
        }
System.out.println("-----------------------");
    }

   public static void listDenda(ObjectSet emaitza) {
        while (emaitza.hasNext()) {
            System.out.println(emaitza.next());
        }
 System.out.println("-----------------------");
    }
}
```

Iturburu-kodean[69] ikusten den legez, programatzaileak ez du inongo kode *arrotzik* gehitu behar iraunkortasuna inplementatzeko aplikazioan. Nahikoa da `ObjectContainer` motako objektu bat antolatzea objektuak biltegiratzeko eta eskuratzeko datu-basetik. Aipatutako eragiketak arazo barik gauzatuko dituzte `set()` eta `get()` metodoek.

Amaitzeko, aplikazioa erabat inplementatuta uzteko, domeinuko klaseak programatu behar ditu programatzaileak. Hona hemen klase horien iturburu-kodea:

```
import java.io.*;
import java.util.*;

public class Eskaera
                  {

       public static String EGINDA = "EGINDA";

       public static String ONARTUTA = "ONARTUTA";

       private static final String BAZTERTUTA = "BAZTERTUTA";

       public static String BIDALITA = "BIDALITA";

       private int id;

       private Date banatzeData;

       private Denda denda;

       private List lerroak = new ArrayList();

       private Helbidea banatzeHelbidea;

       private String egoera = EGINDA;

       private String bezeroa;


       public Eskaera() {

       }

       public Eskaera(int id) {
                      this.id = id;
       }

       public Eskaera(Helbidea banatzeHelbidea, Date banatzeData,
                     Denda denda, String bezeroa) {
              this.banatzeHelbidea = banatzeHelbidea;
              this.banatzeData = banatzeData;
```

69. Irakurle, lasai kodearen zati bat edo beste, tutik ere ez baduzu ulertzen, geroxeago, poliki-poliki eta zehatz-mehatz azalduko da.

```
            this.denda = denda;
           this.bezeroa = bezeroa;
            this.lerroak = new ArrayList();
      }

      public Date lortuBanatzeData() {
            return banatzeData;
                                  }

      public Denda lortuDenda() {
            return denda;
      }

      public Helbidea lortuBanatzeHelbidea() {
            return banatzeHelbidea;
      }

      public List lortuLerroak() {
            return lerroak;
      }

      public String lortuEgoera() {
            return egoera;
      }

      public void gehitu(Artikulua artikulua, int kopurua) {
            lerroak.add(new EskaeraLerroa(kopurua, artikulua));
      }  }
                              oOo

import java.io.*;

public class EskaeraLerroa
      {

       private Artikulua artikulua;
      private String izena;
       private int kopurua;
      private double salneurria;


       public EskaeraLerroa() {
       }

       public EskaeraLerroa(
             int kopurua,
             Artikulua artikulua) {
        this.artikulua = artikulua;
        this.kopurua = kopurua;
        this.izena = artikulua.lortuIzena();
        this.salneurria = artikulua.lortuSalneurria();
             }
```

```
       public int lortuKopurua() {
              return kopurua;
       }

       public void jarriKopurua(int kopurua) {
              this.kopurua = kopurua;
       }

       public Artikulua lortuArtikulua() {
              return artikulua;
       }

       public String lortuIzena() {
              return izena;
       }

       public double lortuSalneurria() {
              return salneurria;
       }  }
```

oOo

```
import java.io.*;

public class Helbidea {

    protected String kale1;

    protected String kale2;

    protected String hiri;

    protected String probintzi;

    protected String postaKodea;

    public Helbidea() {
    }

    public boolean equals(Object object) {
        if (object == null)
            return false;
        if (!(object instanceof Helbidea))
            return false;

        Helbidea beste = (Helbidea) object;

        return ValueObjectUtil.safeEquals(kale1, beste.kale1)
                && ValueObjectUtil.safeEquals(kale2,
                        beste.kale2)
                && ValueObjectUtil.safeEquals(hiri, beste.hiri)
                && ValueObjectUtil.safeEquals(probintzi,
beste.probintzi)
```

```
                && ValueObjectUtil.safeEquals(postaKodea,
beste.postaKodea);
    }

    public int hashCode() {
        return ValueObjectUtil.safeHash(kale1)
                ^ ValueObjectUtil.safeHash(kale2)
                ^ ValueObjectUtil.safeHash(hiri)
                ^ ValueObjectUtil.safeHash(probintzi)
                ^ ValueObjectUtil.safeHash(postaKodea);
    }

    public Helbidea(String kale1, String kale2, String hiri,
            String probintzi, String postaKodea) {
        this.kale1 = kale1;
        this.kale2 = kale2;
        this.hiri = hiri;
        this.probintzi = probintzi;
        this.postaKodea = postaKodea;
    }

    public String lortuKalea1() {
        return kale1;
    }

    public void ipiniKalea1(String kale1) {
        this.kale1 = kale1;
    }

    public String lortuKalea2() {
        return kale2;
    }

    public void ipiniKalea2(String kale2) {
        this.kale2 = kale2;
    }

    public String lortuHiria() {
        return hiri;
    }

    public void ipiniHiria(String hiri) {
        this.hiri = hiri;
    }


    public String lortuPostaKodea() {
        return postaKodea;
    }

    public void ipiniPostaKodea(String postaKodea) {
        this.postaKodea = postaKodea;
    }
```

```
    public boolean isNull() {
        return kale1 == null && kale2 == null && hiri == null
                && probintzi == null && postaKodea == null;
    }   }
```

oOo

```
import java.util.Date;
import java.util.*;

public class Denda {

       private int id;

       private String izena;

       private Set zerbitzuEremua;

       private List artikuluak;

       private Set dendaOrduak;

       private String mota;

       public Denda() {

       }

       public Denda(String izena, String mota, Set zerbitzuEremua,
                     Set irekitzeOrdua, List artikuluak) {
              this.izena = izena;
              this.mota = mota;
              this.zerbitzuEremua = zerbitzuEremua;
              this.dendaOrduak = irekitzeOrdua;
              this.artikuluak = artikuluak;
       }

       public String lortuIzena() {
              return izena;
       }

       public String lortuMota() {
              return mota;
       }

       public Set lortuZerbitzuEremua() {
              return zerbitzuEremua;
       }

       public List lortuArtikuluak() {
              return artikuluak;
       }
```

```
      public void gehitu(List artikuluak) {
        this.artikuluak = artikuluak;
     }

       public Set lortuDendaOrduak() {
              return dendaOrduak;
       }

       public String lortuDendaId() {
              return Integer.toString(id);
       }

       public boolean egonZerbitzuEremuan(Helbidea helbidea, Date
banatzeData) {
              return egonZerbitzuEremuan(helbidea) &&
irekitaEgon(banatzeData);
       }

       private boolean irekitaEgon(Date banatzeData) {
              for (Iterator it = dendaOrduak.iterator(); it.hasNext();) {
                     DendaOrdua tarte = (DendaOrdua) it.next();
                     if (tarte.irekitaEgon(banatzeData))
                            return true;
              }

              return false;
       }

       public boolean egonZerbitzuEremuan(Helbidea helbidea) {
              boolean emaitza =
lortuZerbitzuEremua().contains(helbidea.lortuPostaKodea());
              return emaitza;
       }

       public String lortuId() {
              return Integer.toString(id);
       }

      public String toString() {
      return "Denda:  "+izena+" ===== "+mota;
    }    }
                                  oOo
import java.util.*;
import java.util.Date;

public class DendaOrdua {

       private int id = -1;

       private int astekoEguna;

       private int irekiOrdua;
```

```
	private int irekiMinutua;

	private int itxiOrdua;

	private int itxiMinutua;

	public DendaOrdua() {
	}

	public DendaOrdua(int astekoEguna, int irekiOrdua, int
irekiMinutua,
			int itxiOrdua, int itxiMinutua) {
		this.astekoEguna = astekoEguna;
		this.irekiOrdua = irekiOrdua;
		this.irekiMinutua = irekiMinutua;
		this.itxiOrdua = itxiOrdua;
		this.itxiMinutua = itxiMinutua;
	}

	public boolean equals(Object x) {
		if (x == null)
			return false;
		if (!(x instanceof DendaOrdua))
			return false;

		DendaOrdua beste = (DendaOrdua) x;

		return astekoEguna == beste.astekoEguna && irekiOrdua ==
beste.irekiOrdua
				&& irekiMinutua == beste.irekiMinutua
				&& itxiOrdua == beste.itxiOrdua
				&& itxiMinutua == beste.itxiMinutua;
	}

	public int hashCode() {
		return astekoEguna ^ irekiOrdua ^ irekiMinutua ^ itxiOrdua
^ itxiMinutua;
	}

	public int lortuItxiMinutua() {
		return itxiMinutua;
	}

	public int lortuAstekoEguna() {
		return astekoEguna;
	}

	public int lortuIrekiOrdua() {
		return irekiOrdua;
	}

	public int lortuIrekiMinutua() {
		return irekiMinutua;
	}
```

```
       public int lortuItxiOrdua() {
              return itxiOrdua;
       }

       public boolean irekitaEgon(Date data) {
              Calendar c = Calendar.getInstance();
              c.setTime(data);
              int eguna = c.get(Calendar.DAY_OF_WEEK);
              int ordua = c.get(Calendar.HOUR_OF_DAY);
              int minutua = c.get(Calendar.MINUTE);
              boolean result = eguna == lortuAstekoEguna()
                        && (ordua > irekiOrdua || (ordua == irekiOrdua
&& minutua <= irekiMinutua))
                        && (ordua < itxiOrdua || (ordua == itxiOrdua
&& minutua < itxiMinutua));
              return result;
       }  }
                                    oOo
import java.io.*;

public class Artikulua {

    private String izena;
    private double salneurria;
    private Hornitzailea hornitzailea;

    public Artikulua() {
    }

    public Artikulua(String izena, double salneurria, Hornitzailea
hornitzailea) {
        this.izena = izena;
        this.salneurria = salneurria;
        this.hornitzailea = hornitzailea;
    }

    public String lortuIzena() {
        return izena;
    }

    public double lortuSalneurria() {
        return salneurria;
    }

   public Hornitzailea lortuHornitzailea() {
        return hornitzailea;
    }

  public String toString() {
      return "Artikulua:  "+izena+" ===== "+salneurria;
    }  }
                                    oOo
import java.io.*;

public class Hornitzailea {

    private String izena;
```

```
    private String fidagarritasuna;

    public Hornitzailea() {
    }

    public Hornitzailea(String izena, String fidagarritasuna) {
        this.izena = izena;
        this.fidagarritasuna = fidagarritasuna;
            }

    public String lortuIzena() {
        return izena;
    }

    public String lortuFidagarritasuna() {
        return fidagarritasuna;
    }  }
                                oOo
public class ValueObjectUtil {

       public static boolean safeEquals(String x, String y) {
           return (x == null || y == null) ? x == y : x.equals(y);
       }
       public static int safeHash(String x) {
           return x == null ? 0 : x.hashCode();
       }   }
```

Bistan da objektuekin lan egiteak hainbat bitartekari langabezian uzten dituela. Bilaketak, objektuen arteko nabigazioan oinarritzen dira eta gako nagusiaren garrantzia desagertu egiten da. Hala, bilaketak egiteko era, teknologia berri honetan, zeharo erraztuko da objektuekin egingo baita lan zuzenean, inongo datubihurketarik gabe.

Teknologiak hiru era ditu datu-baseari egingo zaizkion galderak kudeatzeko:

1. Adibide bidezko galdera (Query By Example).
2. Galdera natiboak (Native Queries).
3. Galdera-grafoan oinarritzen den galdera (Simple Object Data Access-SODA).

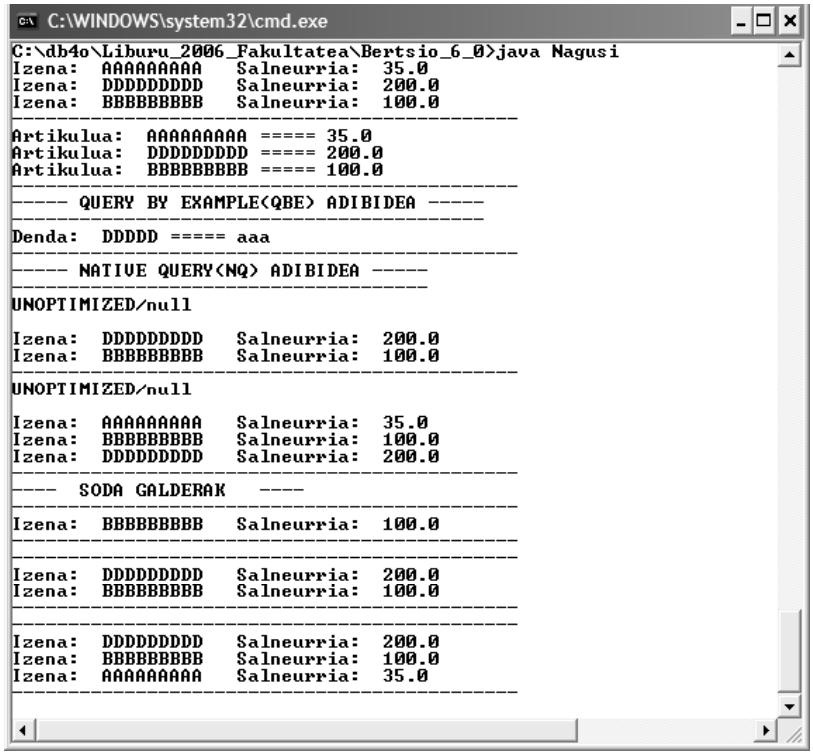
Lehen galdera mota oinarrizkoena da, baina askotan nahikoa izango da objektuak eskuratzeko. Galdera mota horretan, eskuratu nahi den objektuaren txantiloia eraikitzen da lehenengo, eta, klase horren instantzia sortuta, `get()` metodoari deitu baino lehen hasieratu behar diren eremuak hasieratzen dira txantiloian, eta, instantzia hori pasatzen zaio `get()` metodoari egin nahi den galdera egikaritzeko.

Mota horretako bi galdera ditu aztertzen ari garen aplikazioak bere iturburukodean:

```
 ObjectSet emaitza = (ObjectSet) db.get(new Artikulua());
 listArtikuluak(emaitza);  // eskuratu artikuluak
 ...........
System.out.println("--- QUERY BY EXAMPLE(QBE) ADIBIDEA ---");
System.out.println("----------------------");

 Denda dendaProba = new Denda();
 dendaProba.gehitu(artikuluak);
 emaitza = db.get(dendaProba);
 listDenda(emaitza);
```

Lehenengo galderarekin, datu-baseko artikulu guztiak eskuratzen ditu `get()` metodoak. db4o teknologiak `ObjectSet` motako objektuan biltegiratzen du eskuratutako objektu multzoa. Bigarren galderan, bilaketa birfindu egiten da zehaztuz nolakoa izango den hautagaien profila. **4.12. irudia**k islatzen du aplikazioak irteera estandarretik aterako duen informazioa.



4.12. irudia. Adibidearen emaitza irteera estandarretik.

Oinarrizko galderak egikaritu nahi direnean, hau da, bilatu nahi dena oso zehatza denean eta buruz buru jarraitzen dionean egituratu den txantiloiari, adibide bidezko galderak egokiak dira egoera horietarako. Arazoak, galdera konplexuekin sortzen dira, hau da, txantiloia ez denean nahikoa bilaketaren semantika adierazteko.

Halako egoeretan, galdera natiboen teknika egokiagoa eta malguagoa da arazoa konpontzeko. Jakina, malgutasunak konplexutasuna dakar galdera egiteko unean, eta objektu eta metodo berriak eraiki eta programatu behar ditu programatzaileak galdera mota horiek egikaritzeko. Zehatz-mehatz, `Predicate` motako objektua eraiki behar du eta bere `match()` metodoa programatu, bilaketaren irizpideak adierazteko. Amaitzeko, `query()` metodoari pasatzen zaio azpiegitura hori, galdera egikaritzeko. Hona hemen aztertzen ari garen aplikazioan erabiliko den adibidea:

```
System.out.println("--- NATIVE QUERY(NQ) ADIBIDEA ---");
System.out.println("--------------------");

     ObjectSet artikuluakNQ = (ObjectSet)db.query(new
Predicate<Artikulua>() {
            public boolean match(Artikulua artikulua) {
                return artikulua.lortuSalneurria() > 40;
            }
        });

     listArtikuluak(artikuluakNQ);  // eskuratu artikuluak
     artikuluakNQ.reset();
```

Kontu egin ezen aipatutako azpiegitura inner-klase anonimoa erabilita egituratzen dela. Galderaren emaitza irteera estandarretik ateratzen da, **4.12. irudia**n ikusten den legez.

Ur handiagotan murgilduta, datu-baseko artikuluak eskuratu nahi dira salneurriaren arabera sailkatuta, eta horretarako, galdera natiboa egikaritu nahi da lan hori gauzatzeko.

Oraingo honetan, `Comparator` motako objektua eraikiko da predikatuaren nondik norakoak zehazteko eta parametro bezala pasatuko zaio `query()` metodoari. Hala, predikatuak argi izango du zer den konparatu behar duena eta nora joan behar duen informazio bila konparaketa gauzatzeko. Hona hemen azaldutakoaren iturburu-kodea:

```
        // Prestatu Comparator
        Comparator<Artikulua> artikuluCmp=
                new Comparator<Artikulua>() {
            public int compare(Artikulua art1, Artikulua art2) {
            Double art11 = new Double(art1.lortuSalneurria());
            Double art22 = new Double(art2.lortuSalneurria());
        return art11.compareTo(art22);
            }
        };

      artikuluakNQ = (ObjectSet)db.query(new Predicate<Artikulua>()
{
            public boolean match(Artikulua artikulua) {
```

```
                return true;
            }
        }, artikuluCmp);

    listArtikuluak(artikuluakNQ);  // eskuratu artikuluak
    artikuluakNQ.reset();
```

db4o teknologiak onartzen duen hirugarren galdera mota, galdera-grafoan oinarritzen da. Galdera mota horretan, galderaren grafoa erabiltzen da galdera egituratzeko eta, galderaren egitura oso lotua dago domeinuaren objektuen ereduarekin. Horrela, galdera antolatzea erraza izaten da programatzailearentzat, domeinuaren ereduan oinarritzen delako orpoz orpo eta ez eredu erlazional *arrotzean*. Galderaren gardentasuna teknologiak bermatzen du, eta programatzailea ez da horretaz arduratzen.

Mota horretako galdera egikaritu nahi duenean, domeinuaren eredua izan behar du programatzaileak beti begi-bistan, bilatzen ari den informaziora heltzeko. Bilaketan, `Query` motako objektua egituratu behar du bilaketaren ezaugarriak adierazita. Honako adibide honek galdera mota hori erabiltzen du, informazioa eskuratzeko datu-basetik:

```
          // SODA galderak eratzen:
          // ------------
          System.out.println("--  SODA GALDERAK   --");

        Query galdera1 = db.query();
        galdera1.constrain(Artikulua.class);
        galdera1.descend("izena").constrain("BBBBBBBBB");
        ObjectSet emaitza1 = galdera1.execute();
 System.out.println("----------------------");
        listArtikuluak(emaitza1);
```

Bilaketaren norabidea `descend` ezaugarriak adierazten du. Bestalde, bidean aplikatu behar dituen murriztapenak, `constrain` ezaugarriek zehazten dituzte. Galdera egituratuta horrela, nahikoa da `execute()` metodoa egikaritzea galderaren emaitzak eskuratzeko `ObjectSet` motako objektuan.

Hona hemen beste galdera konplexu bat artikuluak eskuratzeko datu-basetik:

```
        Query galdera2 = db.query();
        galdera2.constrain(Artikulua.class);
        Constraint lehenConstr=
galdera2.descend("salneurria").constrain(200);// DD artikulua

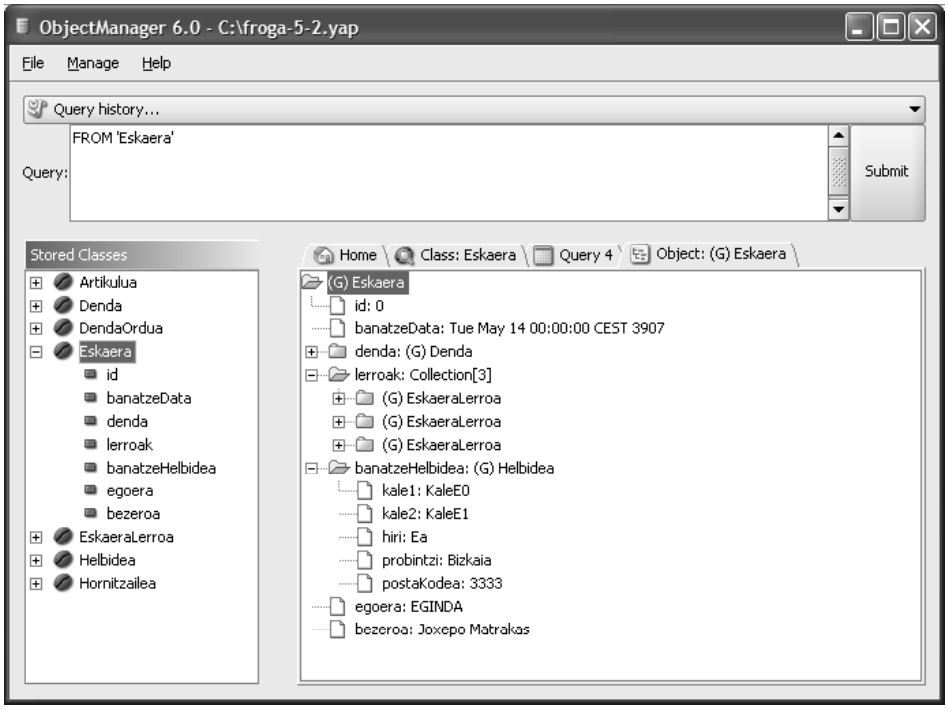
galdera2.descend("izena").constrain("BBBBBBBBB").or(lehenConstr);
        ObjectSet emaitza2 = galdera2.execute();
 System.out.println("----------------------");
        listArtikuluak(emaitza2);
```

Kontu egin ezen oraingo honetan `or` eragile logikoa erabiltzen dela galdera egituratzeko.

Amaitzeko atal hau, sailka ditzagun artikuluak izenaren arabera:

```
        Query galdera3 = db.query();
        galdera3.constrain(Artikulua.class);
        galdera3.descend("izena").orderDescending();
        ObjectSet emaitza3 = galdera3.execute();
System.out.println("----------------------");
        listArtikuluak(emaitza3);
```

Orain arte azaldu denarekin aplikazio informatikoan txertatzen ziren db4o teknologiaren baliabideak objektuetara zuzendutako datu-basea kudeatzeko. Datu-basearen edukiak zuzenean aztertu nahi badira, aplikazio informatikoa erabili gabe ObjectManager tresna (**4.13. irudia**) aurreratua eraiki dute db4o teknologiaren garatzaileek, datu-basera atzitzeko eta edukiak aztertzeko. ObjectManager 6.0 tresnaren interfaze grafikoaren ezaugarriak **4.13. irudia**n ikus daitezke. Datu-base administratzaile horrek pantailaratzen ditu biltegiratutako objektuak eta objektu horien egiturak eta balioak. `Query` atala erabilita, galderak egin ditzake erabiltzaileak, datu-basearen egoera aztertzeko zuzenean.



4.13. irudia. ObjectManager baliabidearen interfaze grafikoa.

4.7. LABURPENA

Hainbat erabaki hartu behar ditu programatzaileak, hiru mailako arkitektura inplementatu nahi duenean. Lehenengo eta behin, erabaki behar du ea, inplementazioak, ikuspegi prozedurala edo objektuetara zuzendutakoa izango duen. Badirudi, orain arte azaldutakoagatik, objektuetara zuzendutako ikuspegia litzatekeela onena eta egokiena. Zoritxarrez edo zorionez hori ez da horrela beti. Batzuetan, erakundearen eskarmentu txikiak, eta beste batzuetan, aplikazioaren tipologiak, baldintzatu egiten dute hautaketa eta ikuspegi prozedurala hautatzen da aplikazioa diseinatzeko eta inplementatzeko.

Ikuspegi prozedurala erabilita antolatzen denean domeinuaren logika, normala izaten da logika hori Transaction Script patroiarekin kontrolatzea. Horrela, erabilera-kasuei transakzio-gidoiak eraikitzen zaizkie transakzioen konplexutasuna kudeatzeko. Inplementazioaren kalitatea eta eraginkortasuna bermatuta gelditzen dira Spring eta iBATIS teknologiak erabilita.

iBATIS teknologia erabilita, iraunkortasunaren abstrakzio maila igo egiten du programatzaileak, lan mekanikoa eta errepikakorra teknologiaren kontrolpean utziz. Abstrakzio maila hori gehiago igo nahi badu, aplikazioaren konplexutasunak hala eskatzen diolako, Hibernate tresna aurreratua erabil dezake iraunkortasuna kudeatzeko. Hala, objektuen eta datu-basearen eskemen arteko mapaketa automatikoki egikarituko da, eta transakzioen kudeaketa ere, bere esku geldituko da.

Hiru mailako arkitekturetan, domeinuaren logikak, jakin badaki, ezker-eskuin dituela beste bi partaideak aiduru eta adi-adi. Alde batean, aplikazioaren informazio-sistemara atzituko du iraunkortasun mailak, hala eskatzen dionean domeinuaren logikak, eta bestean, nola gauzatu nahi duen komunikazioa interfaze grafikoekin eskatuko dio aurkezpen mailak domeinuaren logikari. EJB teknologia erabiltzen bada aplikazioa inplementatzeko, arazoa konponduta dago: sesio-beanek kapsulatuko dute domeinuaren logika. Hala, transakzioen eta segurtasunaren kudeaketa EJB edukiontziak tratatuko ditu.

Aplikazioaren diseinua eta inplementazioa POJO objektuak eraikita gauzatzen badira, domeinuaren logika kapsulatu egiten da Façade diseinu-patroi ezaguna erabilita. Ebazpena biribiltzeko, EJB edukiontziaren funtzionalitateak testuinguru horretara erakarriko dira aipatutako zerbitzuak egikaritzeko. Gaur egun, Spring teknologiak arazorik gabe betetzen eta, batzuetan, hedatzen ditu zerbitzu horiek.

Hiru mailako arkitekturan, iraunkortasun maila aztertu den bakoitzean, argi zegoen iraunkortasun hori datu-base erlazionalak erabilita egiten zela, jakina, mapaketa gauzatu ondoren. Behin eta berriz, programatzaileak programatuta edota automatikoki tresna aurreraturen bat erabilita, datu-baseko tauletara egokitzen ziren objektuak eta tupletatik jasotzen zen informazioa objektuetara eraldatzen zen, objektuak instantziatuz memorian.

Objektuetara zuzendutako db4o tresna aurreratuarekin aipatutako desdoitze-arazo horiek desagertu egiten dira, zuzenean biltegiratzen eta eskuratzen baitira objektuak, inongo datu-bihurketarik gabe.

4.8. OINARRIZKO BIBLIOGRAFIA

Bauer, C. eta King, G. (2006): *Java Persistence with Hibernate*, Manning.
Beck, K. (2003): *Test Driven Development: By Example*, Addison Wesley.
Begin, C.; Goodin, B. eta Meadors, L. (2007): *iBATIS in Action*, Manning.
Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P. eta Stal, M. (1996): *Pattern-Oriented Software Architecture. A System of Patterns*, Wiley&Sons.
Edlich, S.; Hörning, H.; Hörning, R. eta Paterson, J. (2006): *The Definitive Guide to db4o*, Apress.
Fowler, M. (2002): *Patterns of Enterprise Application Architecture*, Addison Wesley.
Panda, D.; Rahman, R. eta Lane, D. (2007): *EJB 3 in Action*, Manning.
Pugh, E. eta Gradecki, J. D. (2004): *Professional Hibernate*, Wrox.
Richardson, C. (2006): *POJOs in Action*, Manning
Sam-Bodden, B. (2006): *Beginning POJOs: Lightweight Java Web Development Using Plain Old Java Objects in Spring, Hibernate, and Tapestry*, Apress.

4.9. WEB GUNE INTERESGARRIAK

Zer dago bertan	Web gunea
J2EE diseinu-patroiak	http://java.sun.com/blueprints/corej2eepatterns/index.html
iBATIS teknologia	http://ibatis.apache.org/
Hibernate teknologia	http://www.hibernate.org/
EJB espezifikazioa(I)	http://java.sun.com/products/ejb/
EJB espezifikazioa(II)	http://java.sun.com/products/ejb/docs.html
EJB 3.0 espezifikazioaren JBoss bertsioa	http://docs.jboss.org/ejb3/app-server/tutorial/
Hibernateren tutoretza	http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernate
Hibernate/EJB	http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateVSEJB3
db4o teknologia	http://www.db4o.com/
db4o teknologiaren tutoretza	http://developer.db4o.com/Resources/view.aspx/Formula_One_Tutorial

5. Laugarren hurbilketa: zerbitzuetara zuzendutako arkitektura (web zerbitzuak)

Teknologia desberdinak garatu izan badira produktu informatiko bera eraikitzeko, gehienetan, produktuaren bertsioa ez da bateragarria izango beste teknologietan. Sarritan, arazoaren errua, erakundearen marketin sailean aurkitzen da eta ez teknologiaren eraginkortasunean. Ez dugu adibiderik jarriko, denoi adibide bera baitatorkigu burura.

Egoera horrek, jakina, ondorioak ditu: A plataformak A_1 teknologia erabilita P produktua eraiki badu, nekeza izango zaio B plataformaz eta B_1 teknologiaz baliatzen denari produktu hori erabiltzea edo hedatzea, P produktuaren eramangarritasun maila hutsaren hurrengoa baita.

Azter dezagun egoera bat gero eta gehiago azaltzen dena informatikaren arloan. Gizarte lehiakor honetan, gero eta gehiagotan, zerbitzuak eskaintzen dituen erakunde batek aurrera egin nahi badu, sare erara antolatu behar du bere eguneroko jarduna, hau da, sare birtual erraldoi batean kokatuta dago eta sarearen nodo baten lekua betetzen ari da. Sare horretan, nodo batek beste batekin harremanetan jarri behar badu sistema informatiko bat erabilita, argi dago bi sistemek bateragarriak izan behar dutela. Oraintsu arte, bateragarritasuna lortzeko erarik errazena plataforma eta teknologia berak erabiltzea izaten zen.

Demagun gure erakundeak bidaiak antolatzen dituela eta eskaintzen duen zerbitzua ahal denik eta integratuena izatea nahi duela, hau da, bidaiaren nondik norako gehienak berak antolatu nahi ditu zerbitzu integral bat eskainiz. Bidaiaren arlo gehienak erreserba-garaian zehaztuko dira, hala nola bidaiaren garraio mota, eta egonaldiko egoitzak eta jarduerak. Arlo batzuk, erakundeak berak antolatuko ditu zuzenean baina beste batzuk zeharka erreserbatuko ditu eta, horretarako, bere sistema informatikoa beste sistema informatiko batzuekin jarriko da harremanetan erreserbaren bideragarritasuna egiaztatzeko. Bestalde, jakin badaki, sistema informatiko horiek bakoitza garai batean garatu zirela eta bakoitzak bere plataforma eta teknologia erabili zituela. Zoritxarrez, sistema horietako batzuk ez dira bateragarriak erakundearen sistema informatikoarekin. Kasu horretan, zer egin? Zerbitzuetara zuzendutako arkitektura bat diseinatu.

Arkitektura mota horrek ikuspegi neutro eta agnostikoa erabiltzen du bere osagaiak antolatzeko. Sistemen arteko bateragarritasuna lortzeko, XML dokumentuak erabiltzen ditu iturburu nagusi gisa eta generizitatea aplikatzen saiatzen da sistemen arteko komunikazioa gauzatzeko. Hala, espezifikazio estandarrez baliatzen da zerbitzuak sortzeko, biltegiratzeko eta eskuratzeko. Prozesuak (**5.1. irudia**), oro har, zazpi urrats nagusi ditu:

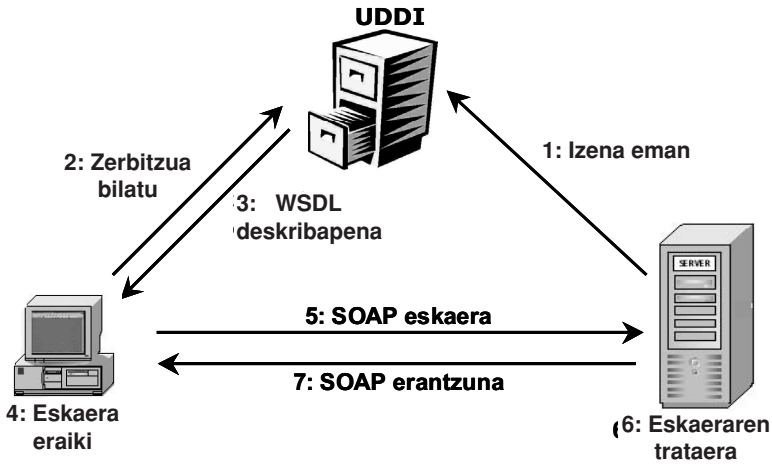
1. Zerbitzu hornitzaile batek web zerbitzu bat eraiki du eta argitara ematen du erregistratuz eta eskuragarri jarriz. Horretarako, UDDI (Universal, Description, Discovery and Integration) baliabidea erabiltzen du, zerbitzu eraiki berria erregistratzeko.

2. Zerbitzua erregistratu ondoren, zerbitzua erabili nahi duen bezeroak zerbitzua bilatzen du, UDDI erregistro-zerbitzua erabilita.

3. Zerbitzuaren deskribapen funtzionala WSDL (Web Services Description Language) dokumentua erabilita egituratzen da.

4. Bere nahietara egokitzen den web zerbitzu bat dagoela ohartarazi ondoren, bezeroak eskaera bat eraikitzen du zerbitzua eskatzeko hornitzaileari.

5, 6 eta 7. Eskaera ez da prozedura-dei bat bezala antolatzen, protokolo estandar bat erabiltzen da mezua eraikitzeko: SOAP (Simple Object Access Protocol)



5.1. irudia. Web zerbitzuen arkitektura orokorra.

Javaren teknologiak (**5.2 irudia**), aurkeztu dugun prozesua gauzatzeko, hainbat API eskaintzen du web zerbitzuen arloak kudeatzeko. Zehatz-mehatz, bost API daude web zerbitzuak eraikitzeko teknologia hori erabilita:

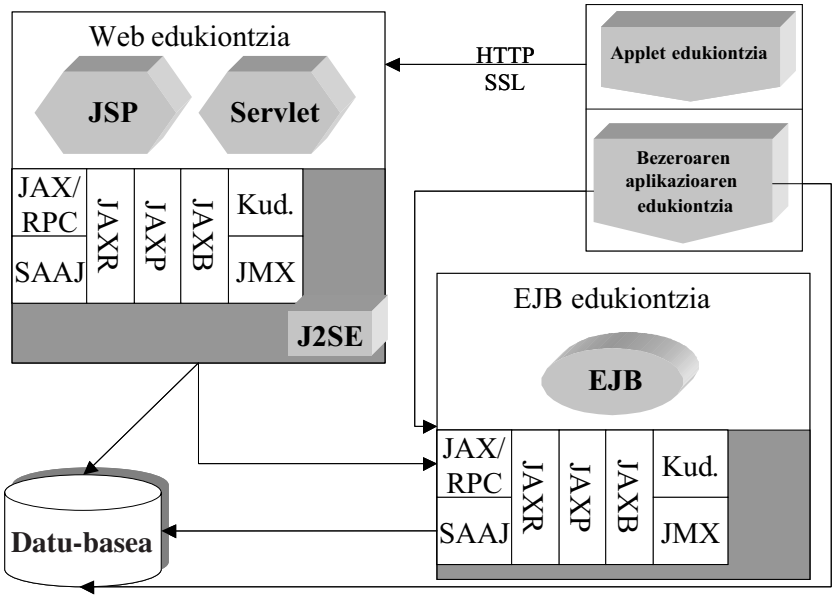
1. JAXP (Java API for XML Processing). Aplikazio informatikoak XML formatuan idatzitako datuak prozesatzeko balio du API honek.
2. JAXB (Java Architecture for XML Binding). APIak Java klaseak sortzen ditu XML eskemak erabilita.
3. SAAJ (SOAP with Attachments API for Java). Web zerbitzuaren mezuak, SOAP espezifikazioa erabilita antolatzen badira, API honek tratatzen ditu.
4. JAX-RPC (Java API for XML-based Remote Procedure Calls). Mezuak urruneko prozedura-deiak bezala tratatzen dira aplikazioan, eta APIak, bere ardurapean hartzen ditu urruneko dei horiek SOAP mezu bat gisa antolatzeko.
5. JAXR (Java API for XML Registries). API honek, XML erregistroak eraikitzeko eta Internet bidez aurkitzeko azpiegitura eskaintzen du.

Teknologia horretan web zerbitzu bat eraiki baino lehen, programatzaileak argi eduki behar du zein programazio-eredu aplikatuko duen zerbitzua sortzean. Lantzen ari garen teknologiak bi aukera eskaintzen dizkigu:

- Lehenengo eredua, urruneko prozedura-deietan oinarritzen da eta, gehienetan, proxy lokal batez baliatzen da zerbitzua eskatzeko.
- Bigarren ereduak, SOAP mezuarekin lan egiten du zuzenean eta SOAP mezuaren zatiak (`Envelope`, `Header`, `Part`...) objektu motekin kudeatzen ditu.

Programatzaileak web zerbitzua lehenengo ereduarekin atzitu nahi badu, nahitaez JAX-RPC APIaz baliatu beharko du urruneko prozedura-deia antolatzeko. Bezeroaren aldean, proxy lokal batekin irudikatuko du programatzaileak zerbitzua eta objektu lokal hori erabiliko du zerbitzuaren baliabideak eskuratzeko era erabat prozeduralean.

Bigarren programazio-eredua aplikatu nahi bada web zerbitzua kudeatzeko, SAAJ APIaz baliatuko da programatzailea. Eredu horretan, zuzenean prozesatzen du programatzaileak SOAP mezua, inongo bitartekaririk gabe.



5.2. irudia. Web zerbitzuak eraikitzeko Javaren teknologiaren APIak.

5.1. JAXP (Java API for XML Processing)

Aplikazio informatiko batek XML formatuan idatzitako datuak prozesatu nahi baditu eta aplikazio hori Javaren teknologia erabilita garatua izan bada, JAXP (Java API for XML Processing) APIaz baliatuko da XML dokumentua aztertzeko. JAXP APIaren baliabide nagusiak `java.xml.parsers` paketean daude adierazita. Pakete horrek faktoria klase bi dauzka XML dokumentuak tratatzeko: `SAXParserFactory` eta `DocumentBuilderFactory`.

JAXP paketeak bi aukera estandar eskaintzen dizkio programatzaileari XML dokumentuak prozesatzeko: SAX (Simple API for XML) eta DOM (Document Object Model) APIak. Lehenengoak, XML dokumentua zatika irakurtzen du eta egitura sintaktiko ezaguna aurkitzen duenean gertaera bat eraikitzen du eta egikaritzen ari den aplikazioari bidaltzen dio. Prozesua gauzatzeko, `ContentHandler` interfazearen metodoak erabiltzen ditu APIak. Adibidez, azterketan, "<" ikurrarekin aurkitzen bada, `startElement()` metodoari deitzen zaio, era berean, karakterezko datu bat badatorkio, `characters()` metodoari deitzen zaio eta "</" ikurrarekin aurkitzen bada, `endElement()` metodoaz baliatuko da. Programatzaileak esplizituki ez badu hala adierazten, metodoek ez dute ezer berezirik egiten. **5.3. taula**k jasotzen du lehenengo zutabean adierazita dagoen XML dokumentuak lerroz lerro izango duen trataera.

XML dokumentua	SAX deiak
<salneurriZerrenda> <liburua> <izena>Softwarearen Ingeniaritza</izena> <salneurria>23.50</salneurria> </liburua> </salneurriZerrenda>	startElement() startElement() startElement(), characters(), eta endElement() startElement(), characters(), eta endElement() endElement() endElement()

5.3. taula. XML dokumentuaren trataera.

Programatzaileak, funtzionalitatea itsatsi nahi badie aipatutako metodoei, azpiklase bat programatu beharko du eta azpiklase horretan birdefinitu metodoen funtzionalitatea. Demagun *Softwarearen Ingeniaritza* izeneko liburuaren salneurria eskuratu nahi dugula eta, informazio hori, XML dokumentu batean datorrela. Eragiketa hori gauzatzeko, klase bat diseinatu behar dugu, `DefaultHandler` klasea hedatzen duena. Klase horrek, oinarrizko inplementazioa itsasten dio lehenago aipatu dugun `ContentHandler` interfazeari. Gure kasuan, nahi dugun funtzionalitatea eskuratzeko, `startElement()` eta `characters()` metodoak birdefinituko ditugu:

```
public void startElement(..., String elementuIzena, ...){
      if(elementuIzena.equals("izena")){
                okIzena = true;
      } else if(elementuIzena.equals("salneurria") && okSoftIng ){
        okSalneurria = true;
        okIzena = false;
      }}

public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (okIzena && s.equals("Softwarearen Ingeniaritza")) {
       okSoftIng = true;
       okIzena = false;
       } else if (okSalneurria) {
       System.out.println("Softwarearen Ingeniaritza liburuaren
       salneurria : " + s);
       okSoftIng = false;
       okSalneurria = false;
       }}}
```

Bi metodoak elkarlanean aritzen dira liburuaren izenburua aurkitzeko eta izenburua aurkituta, salneurria eskuratzeko. Gogoratu salneurria, atzetik datorrela zuzen-zuzenean.

APIak `Softwarearen Ingeniaritza` elementua aurkitu duenean, honelako egikaritze-sekuentzia izango du prozesu nagusiak:

- `startElement()` metodoaren hurrengo deian: `okIzena` aldagaiak `true` balioa hartzen du.
- `characters()` metodoaren hurrengo deian: `okSoftIng` aldagaiak `true` balioa hartzen du.
- `startElement()` metodoaren hurrengo deian: `okSalneurria` aldagaiak `true` balioa hartzen du.
- `characters()` metodoaren hurrengo deian: eskuratutako salneurria idatzi egiten da irteera estandarrean.

Prozesua amaitzeko, faktoria bat eta `SAXParser` motako objektu bat eraiki behar ditugu XML dokumentua analizatzeko:

```
SAXParserFactory faktoria = SAXParserFactory.newInstance();
SAXParser saxParser = faktoria.newSAXParser();
saxParser.parse("salneurriZerrenda.xml", handler);
```

SAX analizatzaileak, `salneurriZerrenda.xml` fitxategia lerroz lerro aztertzen du eta egitura sintaktiko ezaguna aurkitzen duenean, `handler` klasearen metodo egokiari deitzen dio.

Analizatzaileak egiaztapenak gauzatu ditzake egitura sintaktikoak aztertzen ari den bitartean. XML dokumentuak DTDan (Document Type Definition) adierazten diren arauak[70] ondo betetzen dituela ziurtatzen du analizatzaileak. Faktoriari esplizituki adierazi behar dio programatzaileak egiaztapena gauzatu nahi dela `setValidating()` metodoa erabilita:

```
faktoria.setValidating(true);
```

`DOCTYPE` erazagupena erabiliko du XML dokumentuak adierazteko non bilatu eta nola deitzen den XML dokumentuari dagokion DTDa:

```
<!DOCTYPE SalneurriZerrenda SYSTEM "salneurriZerrenda.DTD">
```

Atal honen hasieran aipatu dugu ezen JAXP paketeak bi aukera eskaintzen dituela XML dokumentuak prozesatzeko: SAX eta DOM APIak. Bakoitzak, estrategiak eratzen ditu XML dokumentuak prozesatzeko. Aztertu berri dugu SAX APIak erabiltzen duena. Has gaitezen, beraz, arakatzen, nola antolatzen duen DOM APIak berea.

DOM paketea erabat neutrala eta berrerabilgarria da eta W3C DOM Working Group erakundearen espezifikazioak betetzen ditu orpoz orpo. DOM APIak XML dokumentua itxuraldatu egiten du objektu-zuhaitz gisa irudikatuz. DOM objektua

70. Aztertzen ari garen adibideari, honako DTD hau dagokio:

```
<!ELEMENT salneurriZerrenda (liburua)+>
<!ELEMENT liburua (izena, salneurria) >
<!ELEMENT izena (#PCDATA) >
<!ELEMENT salneurria (#PCDATA) >
```

eraiki ondoren, datu-egitura bat bezala erabil daiteke, hau da, zuhaitz-egitura arrunt baten aurrean egongo bagina bezala. Horrela, zuhaitzaren nodoak, inongo arazorik gabe, txertatu eta ezaba daitezke, `insert()` eta `remove()` metodoak erabilita.

Estrategia horrekin, ausazko atzipenak egin daitezke inongo arazorik gabe, XML dokumentuaren egituraren alde guztietara. Halaber, elementu berriak gehitu daitezke zuhaitzaren egitura handituz. SAX APIa erabiliko bagenu, bigarren eragiketa hori ezin izango litzateke gauzatu, APIak XML dokumentua irakurri egiten baitu bakarrik.

Demagun liburu berri bat gehitu nahi dugula liburuen zerrendan izenburua eta salneurriak biltegiratuz. Horretarako, `salneurriZerrenda.xml` fitxategia erabiliko dugu sarrera gisa, liburuen salneurriak memoriara ekartzeko eta DOM objektua eraikitzeko. DOM objektua eraiki ondoren, lehenengo, txertatu nahi ditugun elementuak eraikiko ditugu eta gero, zuhaitz-egituran txertatuko ditugu, `Node` objektu gisa.

SAX paketearekin egiten genuen bezala orain ere, lehenengo eta behin, faktoria bat sortu beharko dugu, `DocumentBuilder` motako objektu bat eraikitzeko. Objektu hori eraiki eta gero, nahikoa izango zaigu objektuaren `parse()` metodoa egikaritzea.

XML dokumentuaren DOM zuhaitzaren egitura eraikitzeko, gogoratu ezen DOM egiturak XML dokumentuaren egitura memorian irudikatzen duela objektuak eraikita, eta DOM objektuaren metodoak egikarituta eskura ditzakegula, aplikazio informatikoan erabiltzeko. Aipatutako faktoria eta `DocumentBuilder` motako objektuak eraikitzeko honako iturburu-kode hau erabiltzen da:

```
DocumentBuilderFactory faktoria =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document dokumentua = builder.parse("salneurriZerrenda.xml");
```

Iturburu-kode horrekin XML dokumentuaren DOM egitura eraikitzen da memorian. Orain gelditzen zaiguna da egitura aztertzea elementu berria txertatzeko. Gure kasuan erabakitzen dugu ezen, elementu berria, `Softwarearen Ingeniaritza` elementuaren aldamenean txertatuko dugula. Hori egiteko, lehenengo eta behin, liburuen zerrenda bat antolatzen dugu, eta gero, zerrenda aztertzen dugu, `Softwarearen Ingeniaritza` elementua aurkitu arte. Elementua aurkitu ondoren, `Node` motako objektu bat eraikitzen da liburu berrirako eta beste `Node` motako bi objektu, `izena` eta `salneurria` elementuetarako. Bai `izena` bai `salneurria`, karaktere motako datuak direnez, `Text` motako objektu bat eraikitzen da bakoitzerako eta elementuaren `Node` motako objektuari itsasten zaio objektuaren `appendChild()` metodoa egikarituta.

Azaldutako guztia iturburu-kodera itzuliko bagenu, honako kode hau edukiko genuke:

```
Node rootNode = dokumentua.getDocumentElement();
NodeList zerrenda = dokumentua.getElementsByTagName("liburua");

// Begizta zerrenda prozesatzeko.
for (int i=0; i < zerrenda.getLength(); i++) {
     Node thisLiburuNode = zerrenda.item(i);
     Node thisIzenaNode = thisLiburuNode.getFirstChild();
     if (thisIzenaNode == null) continue;
     if (thisIzenaNode.getFirstChild() == null) continue;
     if (! thisIzenaNode.getFirstChild() instanceof
             org.w3c.dom.Text) continue;

   String datua = thisIzenaNode.getFirstChild().getNodeValue();
   if (! datua.equals("Softwarearen Ingeniaritza")) continue;

// Aurkitu dugu Softwarearen Ingeniaritza elementua.
// Eraiki eta txertatu elementu berria.

   Node liburuNodeBerria = dokumentua.createElement("liburua");

   Node izenaNodeBerria = dokumentua.createElement("izena");
   Text tiNode = dokumentua.createTextNode("Javaren hastapenak");
   izenaNodeBerria.appendChild(tiNode);

   Node salneurriNodeBerria = dokumentua.createElement("salneurria");
   Text tsNode = dokumentua.createTextNode("24");
   salneurriNodeBerria.appendChild(tsNode);

   liburuNodeBerria.appendChild(izenaNodeBerria);
   liburuNodeBerria.appendChild(salneurriNodeBerria);
   rootNode.insertBefore(liburuNodeBerria, thisLiburuNode);
   break;
}
```

SAX paketearekin gertatzen zen bezala, hemen ere, XML dokumentuak DTDan adierazten diren arauak ondo betetzen dituen ala ez egiazta ditzake analizatzaileak. Faktoriari esplizituki adierazi behar dio programatzaileak ezen egiaztapena gauzatu nahi dela `setValidating()` metodoa egikarituta:

```
faktoria.setValidating(true);
```

DOCTYPE erazagupena erabiliko du XML dokumentuak adierazteko non bilatu eta nola deitzen den XML dokumentuari dagokion DTDa.

Aplikazioaren egikaritze-prozesua ondo joan bada, elementu berria DOM egituran txertatuko da arazorik gabe. Baina, programatzailea jakitun egongo da hasierako XML dokumentua zaharkitua gelditu zaiola eta ez badu ezer egiten,

beste aplikazio batek erabil dezakeela aldaketak jaso gabe. Horregatik, lehenbailehen, XML dokumentu berri bat eraiki behar du aldaketak islatzeko.

JAXP paketeak XLST (XSL[71] Transformations) APIa eskaintzen dio programatzaileari aipatutako arazoa konpontzeko. XSLT APIak W3C XSL Working Group erakundearen espezifikazioei jarraitzen die orpoz orpo eta formatuen arteko bihurketak gauzatzeko balio du.

Oraintsu aztertu dugun adibidean, XML dokumentu bat analizatu dugu eta DOM objektu bat eraiki dugu beraren egitura memorian jasotzeko. Hori eginda, elementu berri bat txertatu dugu egituran eta hor gelditu gara. Orain egokia litzateke, DOM objektu eguneratua erabilita, beste XML dokumentu bat eraikitzea aldaketak biltegiratzeko. Hori da, hain zuzen, jarraian gauzatuko duguna XSLT APIa erabilita. Zehatz-mehatz, `javax.xml.transform` paketeaz baliatuko gara aipatutako bihurketa egituratzeko.

Aurreko bi egoeretan bezala, hemen ere, lehenengo, faktoria bat eraikiko dugu, eta gero, faktoria hori erabiliko dugu `Transformer` motako objektu bat sortzeko:

```
TransformerFactory transFaktoria = TransformerFactory.newInstance();
Transformer transformer = transFaktoria.newTransformer();
```

`Tansformer` motako objektua eraiki ondoren, nahikoa zaigu objektu horren `transform()` metodoa egikaritzea bihurketa egokia gauzatzeko. Metodoa ondo egikaritzeko, lehenago, bi parametro prestatu behar ditugu honako hau adieraziz: (a) non bilatu behar duen metodoak iturburua bihurketa gauzatzeko eta (b) non utzi behar duen bihurketaren emaitza. Honako iturburu-kode honek aipatutako eragiketak gauzatzen ditu:

```
DOMSource iturburu = new DOMSource(dokumentua);

File XMLBerria = new File("XMLBerria.xml");
FileOutputStream os = new FileOutputStream(XMLBerria);
StreamResult emaitza = new StreamResult(os);
transformer.transform(iturburu, emaitza);
```

Iturburu-kodearen lehenengo lerroak `transform()` metodoaren lehenengo parametroa prestatzen du, eta bere azpiko hiru lerroak programatu dira adierazteko non eta nola gordetzen den bihurketaren emaitza.

Bihurketa biribildu nahi badugu eta XML dokumentu berria HTML dokumentu gisa biltegiratu nahi badugu, XSLT APIak di-da batean zertuko digu bihurketa.

71. `XSL= XML Stylesheet Language`

Bihurketa mota honek, lehen bezala, faktoria bat beharko du lehenengo eta behin. Faktoria eraiki ondoren, `Transformer` motako objektua sortu behar du baina orain `stylesheet` espezifikazioa adierazi behar zaio bihurketa gauzatzeko. Gure kasuan, erabiliko dugun `stylesheet` espezifikazioari esker, taula baten lerroetan txertatuko dira liburu-elementuak bihurketa gauzatzen den neurrian. Horretarako, honako `stylesheet` espezifikazioa erabiliko dugu:

```
<xsl:stylesheet version="1.0" xmlns:xsl ="http: //www.w3.org/ 1999/
XSL/Transform">
   <xsl:template match="salneurriZerrenda ">
         <html><head>Liburu Salneurriak</head>
              <body>
                  <table>
                       <xsl:apply-templates />
                  </table>
              </body>
         </html>
   </xsl:template>
   <xsl:template match="izena">
       <tr><td>
           <xsl:apply-templates />
       </td></tr>
   </xsl:template>
   <xsl:template match="salneurria">
       <tr><td>
           <xsl:apply-templates />
       </td></tr>
   </xsl:template>
</xsl:stylesheet>
```

Azpiegitura guztia prestatu ondoren, hona hemen nola gelditzen den iturburu-kodea bihurketa gauzatzeko:

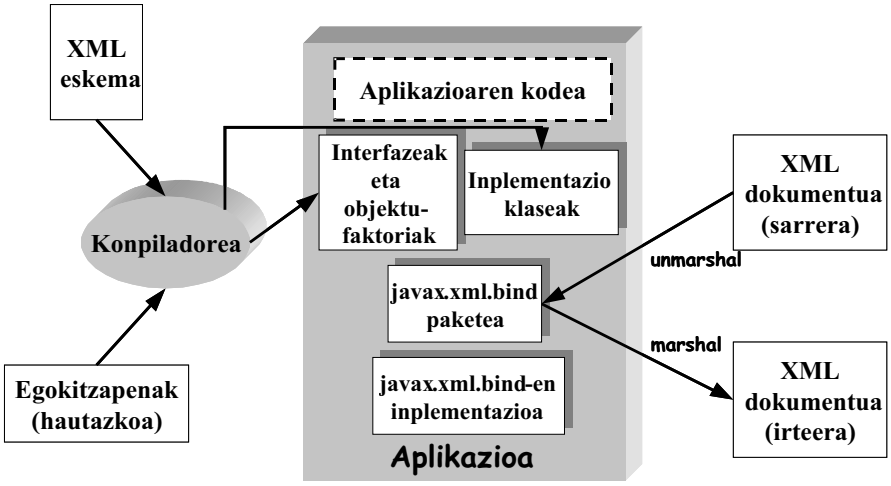
```
TransformerFactory tFaktoria = TransformerFactory.newInstance();
String stylesheet = "salneurriak.xsl";
String iturburuId = "XMLBerria.xml";
File HTMLsalneurriak = new File("HTMLSalneurriak.html");
FileOutputStream os = new FileOutputStream(HTMLsalneurriak);
Transformer transformer = tFaktoria.newTransformer(new
StreamSource(stylesheet));

transformer.transform(new StreamSource(iturburuId), new
StreamResult(os));
```

5.2. JAXB (Java Architecture for XML Binding)

JAXP paketea aztertu dugun bitartean, argi ikusi da programatzaileak Javaren teknologia ez ezik, XML teknologia ere behar bezala menderatu behar duela XML dokumentua prozesatzeko eta analizatzeko. Batzuetan, ordea, ez zaigu interesatzen

horrelako eszenategiaren aurrean egotea aplikazio informatikoa garatzeko. Nahiago genuke baliabide bat izatea XML eskemak eta dokumentuak zuzenean tratatzeko, arduratu gabe eskema eta dokumentu horien barne-egituraz. Egokia litzateke gure Arkadia desiragarri horretan, eskema horien eta dokumentu horien informazioa Java teknologiara itzuliko balitz lehenbailehen eta dokumentu horien informazioa prozesatu ondoren, berriro bihurtu, azkar eta arazorik gabe, XML dokumentu batera, hala nahi izanez gero. Zorionez, JAXB (Java Architecture for XML Binding) APIak buruhauste handirik gabe egiten du hori. JAXB APIak Java klaseak sortzen ditu XML eskemak erabilita. XML dokumentuaren zuhaitz-egitura, objektu bihurtzen du paketeak unmarshal eragiketa aplikatuta. Ildo beretik, alderantzizko bidea egin nahi bada, hau da, Java objektuz hornitutako zuhaitz-egitura XML dokumentu bihurtu, marshal eragiketa gauzatzen da bihurketa gauzatzeko.



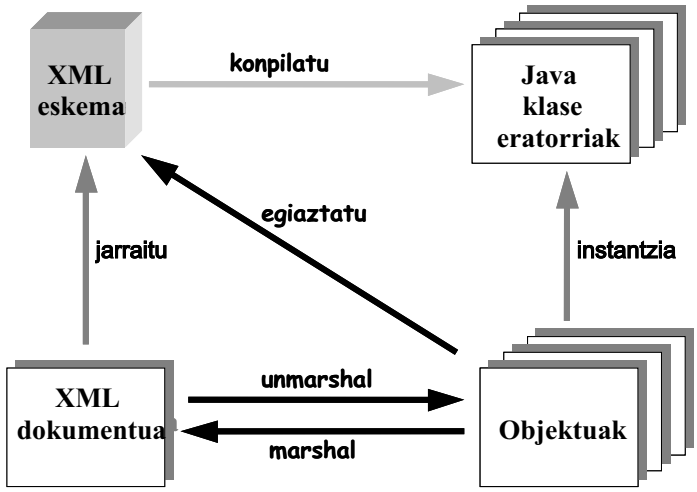
5.4. irudia. JAXB baliabidearen prozesu orokorra.

5.4. irudiak azaltzen du nola egikaritzen den prozesu orokorra. JAXB konpiladoreak XML eskemak eta egokitzapenak irakurtzen ditu eta JAXB klaseak eraikitzen ditu eskema horietan oinarrituta. Hori egin ondoren, sortutako klase guztiak eta aplikazioaren iturburu-kodea konpilatu egiten dira. Hori gauzatuta, aplikazioa egikaritzen da (**5.5. irudia**). Aplikazioak, une batean, XML dokumentu bat irakurriko du eta dokumentu horri, hasieran aipatu ditugun XML eskemaren arauak betetzen baditu, unmarshal eragiketa aplikatuko zaio. Horrenbestez, eskuragarri izango du aplikazioak objektu gisa XML dokumentuaren egitura.

Une horretan, XML dokumentuaren datu irudikatuak, objektu erara alda ditzake aplikazioak eta balio berriak sortu, konpiladoreak eraikitako interfazeak erabilita. Aldaketa horiek biltegiratu nahi baditugu XML dokumentu eran, marshal eragiketa gauzatu behar dugu XML dokumentu bat sortzeko.

Marshal eragiketa egikaritu baino lehen, objektuen edukiak egiazta ditzake aplikazioak, baieztatzeko objektuen egitura bat datorrela XML dokumentuaren XML eskemarekin. Unmarshal eragiketa egikaritu baino lehen ere, aplika daiteke egiaztapen-prozesua, ziurtatzeko, XML eskemaren arauak ondo betetzen dituela XML dokumentuak. Horra hor, beraz, JAXB APIaren ezaugarri nagusiak.

Aztertuta JAXB paketeak nola antolatzen dituen eragiketa nagusiak eta ikusita eragiketa bakoitzak erabiliko dituen baliabideak, jarrai dezagun paketearen azalpenarekin azterketaren azalpen maila igoz.



5.5. irudia. Aplikazioaren egikaritze-fluxua.

Bigarren zati honetan, Sun erakundeak web zerbitzuak aurkezteko prestatu duen tutoretza-eskuliburuan jasota dauden bizpahiru adibide egokitu eta aztertuko ditugu.

Lehenengo adibidean, XML dokumentu bat irakurriko dugu eta unmarshal eragiketa gauzatuta, dokumentuaren zuhaitz-egitura eskuratuko dugu eta objektu eran antolatuko dugu memorian. Urrats guztiak ondo egikaritu direla ziurtatzeko, eraiki dugun informazioa eskuratuko dugu `get()` metodoak erabilita, eta, azkenik, irteera estandarretik aterako dugu lortutako informazioa aplikazioaren erabiltzaileak ikus dezan.

Adibidearen iturburu-kodea aztertu baino lehen, hona hemen adibide guztietan erabiliko den XML dokumentua:

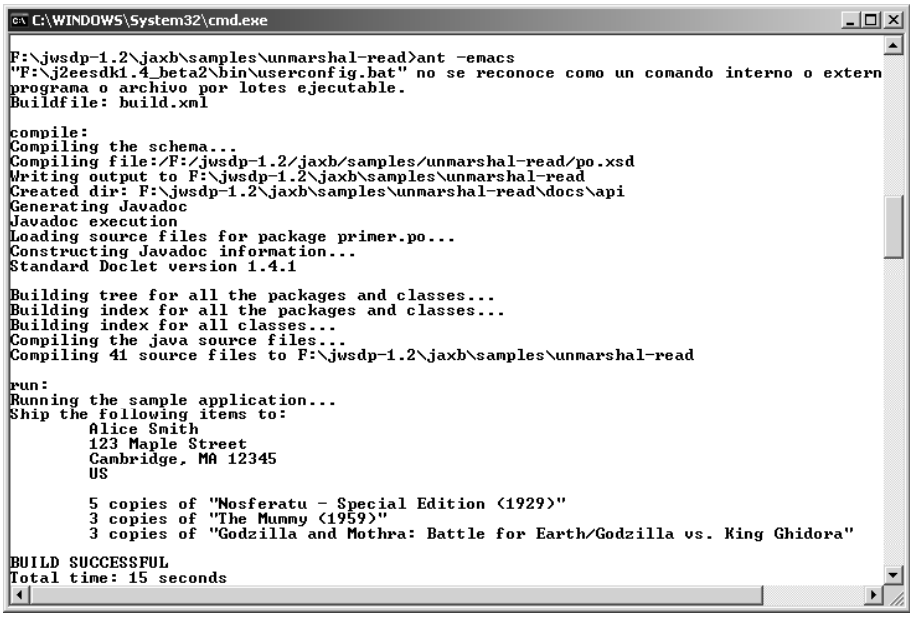
```
<?xml version="1.0" ?>
 <purchaseOrder orderDate="1999-10-20">
   <shipTo country="US">
     <name>Alice Smith</name>
     <street>123 Maple Street</street>
```

```
      <city>Cambridge</city>
      <state>MA</state>
      <zip>12345</zip>
    </shipTo>
  <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <city>Cambridge</city>
      <state>MA</state>
      <zip>12345</zip>
    </billTo>
  <items>
    <item partNum="242-NO">
      <productName>Nosferatu - Special Edition (1929)</productName>
      <quantity>5</quantity>
      <USPrice>19.99</USPrice>
    </item>
  <item partNum="242-MU">
      <productName>The Mummy (1959)</productName>
      <quantity>3</quantity>
      <USPrice>19.98</USPrice>
    </item>
  <item partNum="242-GZ">
      <productName>Godzilla and Mothra: Battle for Earth/Godzilla
        vs. King Ghidora</productName>
      <quantity>3</quantity>
      <USPrice>27.95</USPrice>
    </item>
   </items>
</purchaseOrder>
```

Dokumentuak, bezero hipotetiko baten eskaera bat jasotzen du adieraziz nora (`shipTo`) bidali behar diren eskaeraren artikuluak (`item`) eta nori (`billTo`) fakturatuko zaion eskaeraren zenbatekoa.

Garatu nahi dugun aplikazioak bi gauza eskuratuko ditu: (a) eskaeraren jasotzailearen datu pertsonalak eta (b) eskaeraren artikuluen informazioa. **5.6. irudia**k jasotzen du irteera estandarrean agertuko den informazioa aplikazioa egikarituta.

Adibidearen nondik norakoak argituta, has gaitezen aplikazioaren xehetasunak azaltzen. Horretarako, aplikazioaren iturburu-kodeak laguntza paregabea emango digu aipatutako urratsak nola betetzen diren arakatzeko:



5.6. irudia. Aplikazioaren emaitzak irteera estandarretik.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
public class Main {

    public static void main( String[] args ) {
        try {
            JAXBContext jc = JAXBContext.newInstance( “primer.po” );

            // Unmarshaller motako objektu bat eraiki
            Unmarshaller u = jc.createUnmarshaller();

      PurchaseOrder po =
      (PurchaseOrder)u.unmarshal( new FileInputStream( “po.xml” ) );

            // PurchaseOrder-en edukiak aztertu
            System.out.println( “Ship the following items to: “ );

            // ikuskatu helbidea(nora bidali behar den)
            USAddress address = po.getShipTo();
            displayAddress( address );
```

```
            // ikuskatu eskaeraren elementuak
            Items items = po.getItems();
            displayItems( items );

        } catch( JAXBException je ) {
            je.printStackTrace();
        } catch( IOException ioe ) {
            ioe.printStackTrace();  }}

    public static void displayAddress( USAddress address ) {
        // ikuskatu helbidea
        System.out.println( "\t" + address.getName() );
        System.out.println( "\t" + address.getStreet() );
        System.out.println( "\t" + address.getCity() +
                            ", " + address.getState() +
                            " "  + address.getZip() );
        System.out.println( "\t" + address.getCountry() + "\n"); }

    public static void displayItems( Items items ) {
        // items objektuak List motako zerrenda bat kudeatzen du
        List itemTypeList = items.getItem();


        // iteratzailearen edukiak aztertu
    for( Iterator iter = itemTypeList.iterator(); iter.hasNext(); ) {
            Items.ItemType item = (Items.ItemType)iter.next();
            System.out.println( "\t" + item.getQuantity() +
           " copies of \"" + item.getProductName() +
                               "\"" ); }}}
```

Lehenengo eta behin, `JAXBContext` motako objektu bat eraikitzen da JAXB konpiladoreak `primer.po` paketean sortu dituen klaseak kudeatzeko. Gogoratu klase horiek aplikazioa egikaritu baino lehen eraikitzen dituela konpiladoreak XML dokumentuaren XML eskema (`po.xsd`) erabilita.

Bigarren urratsean, unmarshal eragiketa gauzatzeko, `Unmarshaller` motako objektua eraikitzen da eta objektu horren `unmarshall()` metodoa egikarituta, XML dokumentua (`po.xml`) prozesatzen da. Emaitza `PurchaseOrder` motako aldagai bati esleitzen zaio. `PurchaseOrder` eta geroago agertzen diren `USAddress` eta `Items`, JAXB konpiladoreak eraiki dituen interfazeak dira, eta memorian eraikiko diren objektuak kudeatzeko erabiliko dira. Azter dezagun, adibidez, `PurchaseOrder` interfazea. Hona hemen, konpiladoreak eraiki duen iturburu-kodea:

```
//
// This file was generated by the JavaTM Architecture for XML
Binding(JAXB) Reference Implementation, v1.0.1-05/30/2003 05:06
AM(java_re)-fcs
// See <a
href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
```

```
// Any modifications to this file will be lost upon recompilation
of the source schema.
// Generated on: 2004.04.22 at 11:08:25 CEST
//

package primer.po;

/**
 * Java content class for purchaseOrder element declaration.
 *  <p>The following schema fragment specifies the expected content
 *  contained within this java content object.
 * <p>
 * <pre>
 * &lt;element name="purchaseOrder" type="{}PurchaseOrderType"/>
 * </pre>
 *
 */
public interface PurchaseOrder
    extends javax.xml.bind.Element, primer.po.PurchaseOrderType
{

}
```

oOo

```
//
// This file was generated by the JavaTM Architecture for XML
// Binding(JAXB) Reference Implementation, v1.0.1-05/30/2003 05:06
// AM(java_re)-fcs
// See <a href="http://java.sun.com/xml/jaxb">
// http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation
// of the source schema.
// Generated on: 2004.04.22 at 11:08:25 CEST
//

package primer.po;

/**
 * Java content class for PurchaseOrderType complex type.
 *  <p>The following schema fragment specifies the expected content
 * contained within this java content object.
 * <p>
 * <pre>
 * &lt;complexType name="PurchaseOrderType">
 *   &lt;complexContent>
 *     &lt;restriction
 *   base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="shipTo" type="{}USAddress"/>
 *         &lt;element name="billTo" type="{}USAddress"/>
```

```
 *          &lt;element ref="{}comment" minOccurs="0"/>
 *          &lt;element name="items" type="{}Items"/>
 *        &lt;/sequence>
 *        &lt;attribute name="orderDate"
 *    type="{http://www.w3.org/2001/XMLSchema}date" />
 *      &lt;/restriction>
 *    &lt;/complexContent>
 *  &lt;/complexType>
 * </pre>
 *
 */
public interface PurchaseOrderType {


    /**
     *
     * @return possible object is
     * {@link primer.po.Items}
     */
    primer.po.Items getItems();

    /**
     *
     * @param value allowed object is
     * {@link primer.po.Items}
     */
    void setItems(primer.po.Items value);

    /**
     *
     * @return possible object is
     * {@link java.util.Calendar}
     */
    java.util.Calendar getOrderDate();

    /**
     *
     * @param value allowed object is
     * {@link java.util.Calendar}
     */
    void setOrderDate(java.util.Calendar value);

    /**
     *
     * @return possible object is
     * {@link java.lang.String}
     */
    java.lang.String getComment();

    /**
     *
     * @param value allowed object is
```

```
     * {@link java.lang.String}
     */
    void setComment(java.lang.String value);

    /**
     *
     * @return possible object is
     * {@link primer.po.USAddress}
     */
    primer.po.USAddress getBillTo();

    /**
     *
     * @param value allowed object is
     * {@link primer.po.USAddress}
     */
    void setBillTo(primer.po.USAddress value);

    /**
     *
     * @return possible object is
     * {@link primer.po.USAddress}
     */
    primer.po.USAddress getShipTo();

    /**
     *
     * @param value allowed object is
     * {@link primer.po.USAddress}
     */
    void setShipTo(primer.po.USAddress value);

    }
```

Horiek interfazeak badira, non daude interfazeen inplementazioak? Zorionez, JAXB konpiladoreak, `primer.po` paketean eraiki ditu eta guk ez dugu ezertaz arduratu behar. Adibide gisa, hona hemen, `PurchaseOrderType` interfazearen inplementazioaren zati bat:

```
//
// This file was generated by the JavaTM Architecture for XML
// Binding(JAXB) Reference Implementation, v1.0.1-05/30/2003 05:06
// AM(java_re)-fcs
// See <a href="http://java.sun.com/xml/
// jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation
of // the source schema.
// Generated on: 2004.04.22 at 11:08:25 CEST
//

package primer.po.impl;
```

```
public class PurchaseOrderTypeImpl implements
primer.po.PurchaseOrderType, com.sun.xml.bind.JAXBObject,
primer.po.impl.runtime.UnmarshallableObject,
primer.po.impl.runtime.XMLSerializable,
primer.po.impl.runtime.ValidatableObject
{

    protected primer.po.Items _Items;
    protected java.util.Calendar _OrderDate;
    protected java.lang.String _Comment;
    protected primer.po.USAddress _BillTo;
    protected primer.po.USAddress _ShipTo;
. . . . . .

    public primer.po.Items getItems() {
        return _Items;
    }

    public void setItems(primer.po.Items value) {
        _Items = value;
    }
    public primer.po.USAddress getBillTo() {
        return _BillTo;
    }

    public void setBillTo(primer.po.USAddress value) {
        _BillTo = value;
    }

    public primer.po.USAddress getShipTo() {
        return _ShipTo;
    }

    public void setShipTo(primer.po.USAddress value) {
        _ShipTo = value;}
    . . . . . .

    }
```

Interfazeak eta inplementazioak aztertuta, argi ikusten da zein den JAXB konpiladoreak jarraitzen duen estrategia interfazeak eta inplementazioak eraikitzeko, alegia, XML dokumentuaren informazioa objektuetan egituratzen da eta metodo atzitzaileak (`get()` eta `set()`) eraikitzen dira informazioa eskuratzeko objektu horietatik.

Horregatik, gure adibidean eskaeraren jasotzailearen helbidea eskuratu eta ikusi nahi badugu, honako hau idatziko dugu:

```
USAddress address = po.getShipTo();
displayAddress( address );
```

Eskaeraren artikuluak eskuratu nahi baditugu, eragiketa bera gauzatuko da.

Bigarren adibidean, lehenengo eta behin, unmarshal eragiketa ezaguna egikarituko dugu XML dokumentuaren egitura-zuhaitza eraikitzeko. Hori egin ondoren, memorian dugun informazioa aldatu egingo dugu. Zehatz-mehatz fakturatu behar dugun pertsonaren informazioa aldatu egingo da. Bukatzeko, XML dokumentu bat eraikiko da memoriako informazioa erabilita. Horretarako, `Marshaller` motako objektu bat eraikiko da eta objektu horren `marshall()` metodoa egikarituko da XML dokumentu berria eraikitzeko. Gure adibidean, XML dokumentu berria, irteera estandarrean (**5.7. irudia**) bistaratuko du erabiltzaileak. Hona hemen aipatutako guztia gauzatzeko behar dugun iturburu-kodea:

```
public class Main {

    public static void main( String[] args ) {
        try {

            JAXBContext jc = JAXBContext.newInstance( "primer.po"
);

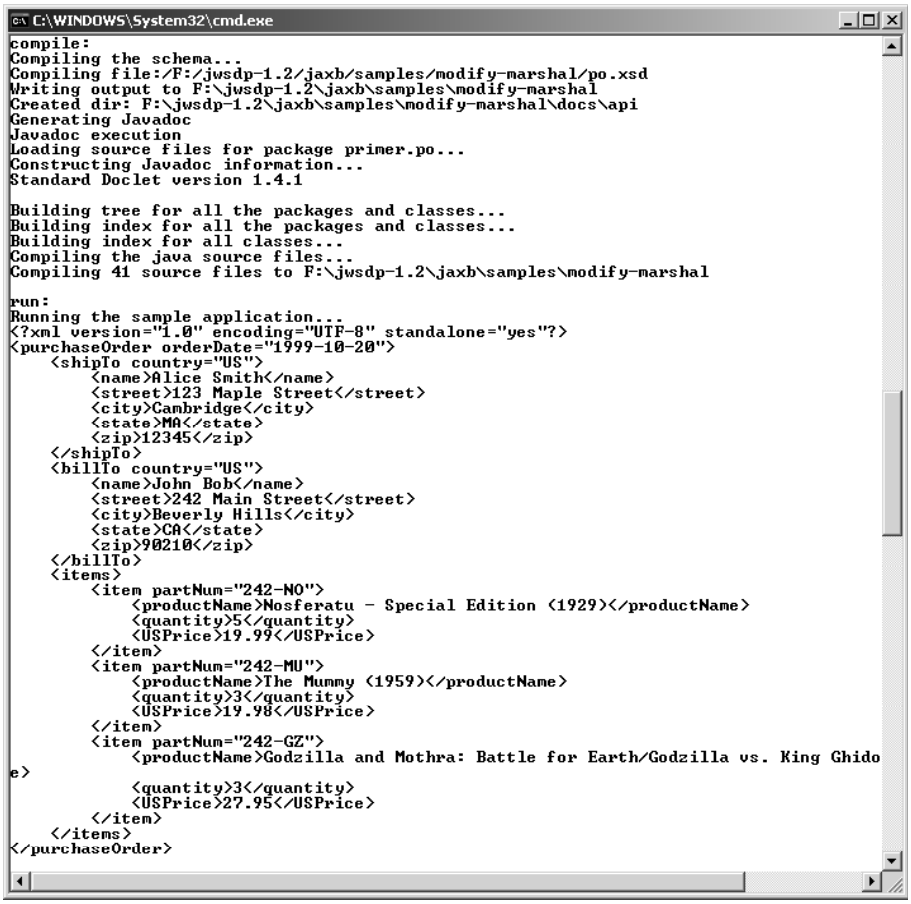
            // Unmarshaller objektu bat eraiki
            Unmarshaller u = jc.createUnmarshaller();

            PurchaseOrder po =
    (PurchaseOrder)u.unmarshal( new FileInputStream( "po.xml" ) );

                    // aldatu nora bidali behar den helbidea
                      USAddress address = po.getBillTo();
                         address.setName( "John Bob" );
                     address.setStreet( "242 Main Street" );
                       address.setCity( "Beverly Hills" );
                            address.setState( "CA" );
                  address.setZip( new BigDecimal( "90210" ) );

            // eraiki Marshaller motako objektu bat
// eta fitxategi batera bidali emaitza
            Marshaller m = jc.createMarshaller();
   m.setProperty( Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE );
            m.marshal( po, System.out );

        } catch( JAXBException je ) {
            je.printStackTrace();
        } catch( IOException ioe ) {
            ioe.printStackTrace();}}}
```



5.7. irudia. Aplikazioaren irteeraren emaitzen XML dokumentua.

Bukatzeko adibideen atal hau, JAXB konpiladoreak eraikitzen duen objektuen faktoriaren egitura aztertuko dugu adibide txiki bat landuz. Oraingoan, marshal eragiketa gauzatuko du adibideak aurretik inolako unmarshal eragiketarik egin gabe. Alegia, XML dokumentua eraikitzeko behar duen informazio guztia aplikazioak berak sortuko du egikaritzen denean. Horretarako, XML dokumentu bat irakurri ordez, `ObjectFactory` motako objektu bat eraikiko da eta objektu horren `createPurchaseOrder()` metodoa egikarituta, `PurchaseOrder` motako objektu bat eskuratuko da. Hemendik aurrera, bakarrik falta dugu, `PurchaseOrder` motako objektuaren `set()` metodoak egikaritzea. Horrela, XML dokumentua eraikitzeko behar dugun informazio guztia objektu eran antolatuko dugu eta amaieran, marshal eragiketa antolatuko dugu informazio hori guztia XML dokumentu (**5.8. irudia**) batera iraultzeko. Horretarako, `Marshaller` motako objektu bat eraikiko dugu eta, aurreko adibidean bezala, objektu horren `marshal()` metodoa egikarituko dugu. Hona hemen aplikazioaren iturburu-kodea adibide honetarako:

```
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Calendar;
import java.util.List;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import primer.po.*;

public class Main {

    public static void main( String[] args ) {
        try {
      JAXBContext jc = JAXBContext.newInstance( "primer.po" );

            // ObjectFactory instantzia eraiki

            ObjectFactory objFactory = new ObjectFactory();

            // eraiki PurchaseOrder bat hutsik
            PurchaseOrder po = objFactory.createPurchaseOrder();

            po.setOrderDate( Calendar.getInstance() );

            // eraiki USAddress(nora) motako objektu bat
            USAddress shipTo = createUSAddress( objFactory,
                              "Alice Smith","123 Maple Street",
                                "Cambridge","MA","12345" );

            // adierazi helbide horretara bidali nahi dela eskaera
            po.setShipTo( shipTo );

            // eraiki USAddress(nori) motako objektu bat
            USAddress billTo = createUSAddress( objFactory,
                                                "Robert Smith",
                                                "8 Oak Avenue",
                                                "Cambridge",
                                                "MA",
                                                "12345" );

            // adierazi helbide horri fakturatuko zaiola eskaera
            po.setBillTo( billTo );

            // eraiki Items motako objektua
            Items items = objFactory.createItems();

            List itemList = items.getItem();

            itemList.add( createItemType( objFactory,
                           "Nosferatu - Special Edition (1929)",
                                         new BigInteger( "5" ),
                                         new BigDecimal( "19.99" ),
```

```
                                        null,
                                        null,
                                        "242-NO" ) );
            itemList.add( createItemType( objFactory,
                                        "The Mummy (1959)",
                                        new BigInteger( "3" ),
                                        new BigDecimal( "19.98" ),
                                        null,
                                        null,
                                        "242-MU" ) );
            itemList.add( createItemType( objFactory,
   "Godzilla and Mothra: Battle for Earth/Godzilla vs. King Ghidora",
                                        new BigInteger( "3" ),
                                        new BigDecimal( "27.95" ),
                                        null,
                                        null,
                                        "242-GZ" ) );

            // adierazi eskaeraren elementuak
            po.setItems( items );

            // eraiki Marshaller motako objektua
            // eta marshal eragiketa gauzatu.
            // Emaitza irteera estandarretik ateratzen da
            Marshaller m = jc.createMarshaller();
  m.setProperty( Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE );
            m.marshal( po, System.out );

        } catch( JAXBException je ) {
            je.printStackTrace();  } }

    public static USAddress createUSAddress( ObjectFactory objFactory,
                                        String name, String street,
                                        String city, String state,
                                               String zip )
        throws JAXBException {

        // eraiki USAddress motako objektua
        USAddress address = objFactory.createUSAddress();

        // itsatsi propietateak objektu eraiki berriari
        address.setName( name );
        address.setStreet( street );
        address.setCity( city );
        address.setState( state );
        address.setZip( new BigDecimal( zip ) );

        return address;
    }

    public static Items.ItemType createItemType( ObjectFactory
objFactory,
                                                 String productName,
                                                 BigInteger quantity,
```

```
                                                BigDecimal price,
                                                String comment,
                                                Calendar shipDate,
                                                String partNum )
        throws JAXBException {

        // eraiki ItemType motako objektua
        Items.ItemType itemType = objFactory.createItemsItemType();

        // itsatsi propietateak objektu eraiki berriari
        itemType.setProductName( productName );
        itemType.setQuantity( quantity );
        itemType.setUSPrice( price );
        itemType.setComment( comment );
        itemType.setShipDate( shipDate );
        itemType.setPartNum( partNum );

        return itemType;
    }}
```

```
C:\WINDOWS\System32\cmd.exe
compile:
Compiling the schema...
Compiling file:/F:/jwsdp-1.2/jaxb/samples/create-marshal/po.xsd
Writing output to F:\jwsdp-1.2\jaxb\samples\create-marshal
Created dir: F:\jwsdp-1.2\jaxb\samples\create-marshal\docs\api
Generating Javadoc
Javadoc execution
Loading source files for package primer.po...
Constructing Javadoc information...
Standard Doclet version 1.4.1

Building tree for all the packages and classes...
Building index for all the packages and classes...
Building index for all classes...
Compiling the java source files...
Compiling 41 source files to F:\jwsdp-1.2\jaxb\samples\create-marshal

run:
Running the sample application...
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<purchaseOrder orderDate="2004-04-22+01:00">
    <shipTo>
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Cambridge</city>
        <state>MA</state>
        <zip>12345</zip>
    </shipTo>
    <billTo>
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Cambridge</city>
        <state>MA</state>
        <zip>12345</zip>
    </billTo>
    <items>
        <item partNum="242-NO">
            <productName>Nosferatu - Special Edition (1929)</productName>
            <quantity>5</quantity>
            <USPrice>19.99</USPrice>
        </item>
        <item partNum="242-MU">
            <productName>The Mummy (1959)</productName>
            <quantity>3</quantity>
            <USPrice>19.98</USPrice>
        </item>
        <item partNum="242-GZ">
            <productName>Godzilla and Mothra: Battle for Earth/Godzilla vs. King Ghido
e>
            <quantity>3</quantity>
            <USPrice>27.95</USPrice>
        </item>
    </items>
</purchaseOrder>
```

5.8. irudia. Aplikazioaren irteera `marshal` eragiketa egikarituta.

5.3. SAAJ (SOAP with Attachments API for Java)

Aurreko ataletan aztertu dugu ezen Javaren teknologiak APIak dauzkala XML dokumentuak analizatzeko eta prozesatzeko. API horiek arazorik gabe erabilita, XML dokumentuaren informazioa eskuratu eta egokitu dezake programatzaileak bere aplikazio informatikoan integratzeko. Halaber, ikusi dugunez, memoriako informazioa berriz XML dokumentu eran antolatu nahi badugu, APIek baliabideak eskaintzen dizkigute eragiketa hori gauzatzeko. Horrela, aplikazioen artean, notazio neutro eta agnostiko bat erabili nahi edo behar dugunean, XML dokumentuena izan daiteke aukera taxuzko bat.

Ados jarrita XMLren notazio neutroa ondo datorkigula aplikazioen komunikazioa antolatzeko, egin dezagun jauzi kualitatiboa eta jar gaitezen beste egoera berri batean.

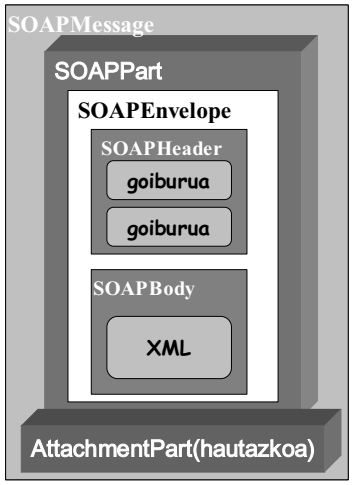
Egoera berri horretan, plataformen arteko agnostizismoa aldarrikatuko dugu eta XXI. mendeko sermolari postmodernoak bihurtuko gara. Gure liburu zuriak maitasuna, elkarrizketa eta elkartasun kosmiko eta unibertsalak bultzatuko ditu. Biodibertsitateak on egiten baitu han eta hemen.

Gure betebehar agnostikoa aurrera eramateko, protokolo neutroa aukeratu behar dugu, hori bai, XML dokumentuak erabiltzen dituena euskarri gisa. SOAP (Simple Object Access Protocol) protokoloaren espezifikazioa primeran datorkigu aldarrikatzen ari garen ebanjelizazioari ekiteko.

Orain arte ikusitakoak ikusita, mezu bat sarean zehar bidali nahi genuenean, ez zegoen arazo handirik mezu hori prozesatzeko baldin bezeroa eta zerbitzaria plataforma bereko teknologiarekin eraikita bazeuden. Aplikazioa programatzeko era plataformaren teknologiarekin lotzen zen, eta ondo lotu ere, zeren gehienetan sortzen zen softwarea ez baitzen inondik ere bateragarria eta eramangarria beste plataformetara.

SOAP espezifikazioak egoera erabat itxuraldatzen du. Plataformen arteko ika-mikak baztertu eta mezuaren formatua hartzen du aintzakotzat. Horrela, edozein plataformak (.NET, EJB...) beste baten baliabideetara atzitu nahi badu, aski da espezifikazioaren inplementazioak izatea atzipena gauzatzeko. Plataforma batek dei bat luzatu nahi dionean beste plataforma bati, aski izango du mezua SOAP erara (**5.9. irudia**) antolatzea besteak ulertzeko. Bigarren plataformak gauza bera egingo du deiaren erantzuna antolatzeko.

5.9. irudian ikusten den legez, SOAP mezuak bi zati nagusi ditu: `SOAPPart` eta `AttachmentPart`. Bigarrena hautazkoa da eta gehienetan ez da erabiltzen. `SOAPPart` zatiaren azpizati nagusia `SOAPEnvelope` zatia da eta bertan daude mezuaren goiburuak (`SOAPHeader`) eta mezuaren gorputza (`SOAPBody`). Azpimarratu behar da ezen zati itsatsiekin gertatzen zen bezala, mezuaren goiburuak hautazkoak direla eta programatzaileak ezaba ditzakeela mezua bidali baino lehen.



5.9. irudia. SOAP mezuaren egitura orokorra.

Arestian adierazi dugunez, plataforma bakoitzak bere APIak eduki behar ditu SOAP mezuak prozesatzeko, eta erantzuna memorian eraiki ondoren, erantzun hori, SOAP mezu gisa antolatzeko. Javaren teknologiak SAAJ (SOAP with Attachments API for Java) APIa erabiltzen du aipatutako eragiketak gauzatzeko. APIak faktoria erako objektuak erabiltzen ditu SOAP mezuaren bi eragiketa nagusiak egituratzeko: konexioa sortu eta mezua antolatu.

Lehenengoko eragiketa eratzeko, konexioa sortzeko alegia, `SOAPConnectionFactory` motako objektua instantziatuko da lehenengo eta behin, eta gero, objektu hori erabili `SOAPConnection` motako objektua eraikitzeko. Konexioa eratu eta gero, konexio hori mezuak bidaltzeko erabili nahi badu, eraiki duen azken objektuaren `call()` metodoa egikarituko du mezua helbide egokira bidaltzeko. Sortzen den konexioa puntutik puntura erakoa denez, eskaera-erantzuna estrategia baliatzen du mezuak prozesatzeko. Orain arte azaldutakoa iturburu-kodera ekarriko bagenu, honako kode hau beharko genuke konexioa sortzeko eta deia gauzatzeko:

```
SOAPConnectionFactory soapConnectionFactory =
          SOAPConnectionFactory.newInstance();
SOAPConnection connection =
          soapConnectionFactory.createConnection();
. . . . . .
URL endpoint = new
      URL("http://www.balioZerbitzua.com/zerbitzua");
SOAPMessage erantzuna = connection.call(mezua, endpoint);
```

Jakina, prozesu guztia biribiltzeko, protagonista nagusiaren nondik norakoak azaltzea falta dugu. Orain, SOAP mezua egituratzeko, prozesuak ildo beretik jarraituko du: lehenengo eta behin, SOAP faktoria bat eraiki, `SOAPFactory`

klasearen `newInstance()` metodo estatikoa egikarituta. Gero, mezua eraikitzen hasten gara, eta horretarako, `MessageFactory` klasearen `newInstance()` metodo estatikoa egikaritzen dugu mezuen faktoria bat eraikitzeko. Faktoria eraiki ondoren, objektuaren `createMessage()` metodoa egikaritzen dugu `SOAPMessage` motako objektu bat sortzeko. Hona hemen prozesu guztia, iturburu-kodera iraulita:

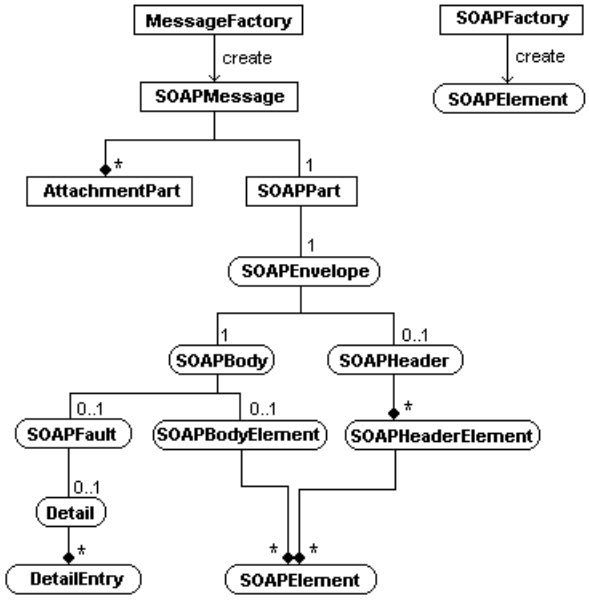
```
SOAPFactory soapFactory = SOAPFactory.newInstance();

    MessageFactory faktoria = MessageFactory.newInstance();
    SOAPMessage mezua = faktoria.createMessage();

        SOAPHeader goiburua = mezua.getSOAPHeader();
        SOAPBody gorputza = mezua.getSOAPBody();

        goiburua.detachNode();
```

Mezua eraiki ondoren, mezuaren zatietara atzitu nahi badugu, `get()` metodoak erabiltzen dira zati zehatz horietara atzitzeko. Atzipena, urratsez urrats bete daiteke mezuaren egitura kontuan izanda, hau da, lehenengo, mezuaren `SOAPPart` eskuratu, gero, zati hori erabili mezuaren `SOAPEnvelope` zatia lortzeko, eta, azkenik, zati hori erabili, mezuaren `SOAPHeader` eta `SOAPBody` zatiak antolatzeko. Badago, baina, beste bide zuzenago bat zatiak eskuratzeko eta hori da, hain zuzen, erabili duguna goiko adibidean.



5.10. irudia. SOAP mezua antolatzeko klase nagusien egituraketa.

SOAP mezuei buruzko lerroak berriz irakurtzen badituzu, irakurle, ageriagerian gelditzen da programatzaileak argi eduki behar duela mezuaren egitura nolakoa den. **5.10. irudia**k azaltzen du nola egituratzen diren SOAP mezua antolatzeko behar diren klaseak. Kontu egin ezen klase gehienak SOAP espezifikazioaren alderdi guztiak hertsiki betetzeko diseinatu direla.

Orain arte egindakoari gainbegiratu bat ematen badiogu, berehala konturatzen gara bi eragiketa nagusi gauzatu ditugula: konexioa sortu eta SOAP mezua eraiki, oraingoz edukirik gabe. Prozesuaren barne-portaera aztertuko bagenu, orain arte egindakoak honelako XML dokumentua eraikiko luke SOAP mezuarentzat:

```
<SOAP-ENV:Envelope
 xmlns[72]:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    . . . . . .
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Osa dezagun, beraz, mezua, elementuak eta ezaugarri berriak itsatsita mezuaren gorputzari. Horretarako, adibide bat garatuko dugu agian noizbait egia izan daitekeena. Demagun urruneko zerbitzari batek zerbitzu bat eskaintzen duela bertara SOAP mezu egokia bidaltzen baduzu. Urruneko zerbitzariak hainbat enpresa komertzialen ranking bat kudeatzen du eta zerrenda hori eguneratu egiten da egunero balio berriekin. Guri Euskaltelen balioa igo edo jaitsi den jakitea interesatzen zaigu eta, horretarako, aplikazio txiki bat eraikiko dugu balio hori eskuratzeko.

Konexioak eta mezua eraikitzeko, goian aipatu ditugun urratsak egikarituko lituzke aplikazioak inongo aldaketarik gabe, baina oraingoan, mezuaren gorputza, informazioarekin bete beharko litzateke, SOAP mezu egokia eratzeko:

```
Name gorputzarenIzena = soapFactory.createName(
  "EskuratuAzkenBalioa", "m", "http://www.balioZerbitzua.com");
SOAPBodyElement gorputzElementua =
          gorputza.addBodyElement(gorputzarenIzena);

Name izena = soapFactory.createName("ezaugarria");
      SOAPElement ezaugarria =
          gorputzElementua.addChildElement(izena);
      ezaugarria.addTextNode("Euskaltel");
```

Iturburu-kodean ikusten den legez, mezuaren gorputzari elementu bat itsasten diogu, `addBodyElement()` metodoa egikarituta. Lehenago, metodoak behar duen parametroa eraikitzen dugu, `SOAPFactory` motako objektuaren `createName()` metodoa egikarituta. Metodoak hiru parametro behar ditu: gorputzaren izen lokala, izenen espazioan erabiliko den aurrizkia eta aplikatuko den URIa.

72. **xmlns**-k XML namespace adierazten du eta bere balioak zehazten du zein izango den `Envelope` elementuari asoziatuko zaion URIa (Uniform Resource Identifier).

Atzetik datozen hiru lerroekin, `<ezaugarria>` elementua eraikitzen dugu lehenengo, eta gero, elementu horri "`Euskaltel`" `String` motako balioa itsasten diogu testu-nodo gisa, ezaugarriaren `addTextNode()` metodoa egikarituta. **5.11. irudia**k SOAP mezu osoaren egitura jasotzen du, adieraziz, mezuaren elementu bakoitza norekin erlazionatzen den SAAJ APIaren barnean.



5.11. irudia. SOAP mezuaren egitura.

Amaitzeko, hona hemen aplikazio osoaren iturburu-kodea:

```
import javax.xml.soap.*;
import java.util.*;
import java.net.URL;

public class Eskaera {
    public static void main(String[] args)  {
        try {
            SOAPConnectionFactory soapConnectionFactory =
                SOAPConnectionFactory.newInstance();
            SOAPConnection connection =
                soapConnectionFactory.createConnection();
       SOAPFactory soapFactory = SOAPFactory.newInstance();

        MessageFactory faktoria = MessageFactory.newInstance();
        SOAPMessage mezua = faktoria.createMessage();

            SOAPHeader goiburua = mezua.getSOAPHeader();
            SOAPBody gorputza = mezua.getSOAPBody();
            goiburua.detachNode();

        Name gorputzarenIzena = soapFactory.createName(
        "EskuratuAzkenBalioa", "m", "http://www.balioZerbitzua.com");
            SOAPBodyElement gorputzElementua =
                gorputza.addBodyElement(gorputzarenIzena);

        Name izena = soapFactory.createName("ezaugarria");
        SOAPElement ezaugarria =
            gorputzElementua.addChildElement(izena);
        ezaugarria.addTextNode("Euskaltel");
```

```
       URL endpoint = new
URL("http://www.balioZerbitzua.com/zerbitzua");
       SOAPMessage erantzuna = connection.call(mezua, endpoint);
            connection.close();

            SOAPBody soapBody = erantzuna.getSOAPBody();
            Iterator iterator =
                soapBody.getChildElements(gorputzarenIzena);
            SOAPBodyElement gorputzElementua =
                (SOAPBodyElement)iterator.next();
            String azkenBalioa = gorputzElementua.getValue();

       System.out.print("Euskaltelen azken balioa honako hau da: ");
            System.out.println(azkenBalioa);

        } catch (Exception ex) {
            ex.printStackTrace();    }}}
```

Iturburu-kodean ikusten den legez, deia gauzatu baino lehen, URL bat eraikitzen du aplikazioak konexioaren `call()` metodoari adierazteko nora bidali behar duen SOAP mezua.

Aplikazioaren azken lerroetan, jaso den erantzunaren gorputza eskuratzen da, `getSOAPBody()` metodoa egikarituta, eta, bukatzeko, iteratzaile bat erabilita, gorputzaren elementuak lortzen dira. Gure adibidean, elementu bakarra dugunez, `next()` metodoarekin eskuratzen da eta `getValue()` metodoa egikarituta, bere `String` balioa lortzen da.

Demagun aztertu dugun itxurazko adibidea erreal bihurtu nahi dugula, hau da, benetan gauzatu nahi dugula atzipena, Interneteko helbide erreal bat erabilita. Hori egiteko, ez dugu urrunera joan beharrik paraje aurkitzen baita ebazpena. Nahikoa da Sun erakundearen tutoretza-eskuliburua erabiltzea eta hango adibide bat hartzea. Zehatz-mehatz, `MyUddiPing` izeneko aplikazioa hartuko dugu:

```
public class MyUddiPing {

    public static void main(String[] args) {
        try {

            if (args.length != 2)  {
                System.err.println("Argument required: " +
                    "-Dbusiness-name=<name>");
                System.exit(1);
            }

            // uddi.properties fitxategian datoz
            // aplikazioak erabiliko dituen propietateak,

            Properties myprops = new Properties();
            myprops.load(new FileInputStream(args[0]));

            Properties props = System.getProperties();
```

```
Enumeration enum = myprops.propertyNames();
while (enum.hasMoreElements()) {
    String s = (String) enum.nextElement();
    props.setProperty(s, myprops.getProperty(s));
}

// Eraiki konexioa eta mezuen faktoria
SOAPConnectionFactory soapConnectionFactory =
    SOAPConnectionFactory.newInstance();
SOAPConnection connection =
    soapConnectionFactory.createConnection();
MessageFactory messageFactory =
    MessageFactory.newInstance();

// Mezua eraiki
SOAPMessage message =
    messageFactory.createMessage();

SOAPHeader header = message.getSOAPHeader();
SOAPBody body = message.getSOAPBody();
header.detachNode();

// SOAP faktoria bat eraiki

SOAPFactory soapFactory =
    SOAPFactory.newInstance();
SOAPBodyElement findBusiness =
    body.addBodyElement(soapFactory.createName(
  "find_business", "", "urn:uddi-org:api_v2"));
findBusiness.addAttribute(soapFactory.createName(
    "generic"), "2.0");
findBusiness.addAttribute(soapFactory.createName(
    "maxRows"), "100");
SOAPElement businessName =
    findBusiness.addChildElement(
        soapFactory.createName("name"));
businessName.addTextNode(args[ 1]);

message.saveChanges();

// Ikuskatu bidaliko dugun mezua
System.out.println("\n-- Request Message --\n");
message.writeTo(System.out);

// Eskuratu fitxategitik mezua bidali behar dugun tokia
URL endpoint = new URL(
    System.getProperties().getProperty("URL"));

// Bidali mezua eta jaso erantzuna
SOAPMessage reply =
    connection.call(message, endpoint);

System.out.println("\n\nReceived reply from: " +
    endpoint);
System.out.println("\n-- Reply Message --\n");
```

```
// Ikuskatu erantzuna
reply.writeTo(System.out);

// Analizatu erantzuna irakurgarri uzteko
SOAPBody replyBody = reply.getSOAPBody();

System.out.println("\n\nContent extracted from " +
    "the reply message:\n");

Iterator businessListIterator =
    replyBody.getChildElements(
        soapFactory.createName("businessList",
            "", "urn:uddi-org:api_v2"));


SOAPBodyElement businessList =
    (SOAPBodyElement)businessListIterator.next();

/*
 *  Eskuratu elementuak.
 *  Galderan jarri dugun baldintza betetzen
 *  duten elementuak izango dira
 */
Iterator businessInfosIterator =
    businessList.getChildElements(
        soapFactory.createName("businessInfos",
            "", "urn:uddi-org:api_v2"));

SOAPElement businessInfos =
    (SOAPElement)businessInfosIterator.next();

/*
 * Erakundearen izena eta
 * agian, deskripzioa eskuratzen dira
 */
Iterator businessInfoIterator =
    businessInfos.getChildElements(
        soapFactory.createName("businessInfo",
            "", "urn:uddi-org:api_v2"));

if (! businessInfoIterator.hasNext()) {
    System.out.println("No businesses found " +
   "matching the name \"" + args[ 1]  + "\".");

} else {
    while (businessInfoIterator.hasNext()) {
        SOAPElement businessInfo = (SOAPElement)
            businessInfoIterator.next();
        // Eskuratu izena eta  deskripzioa

        Iterator nameIterator =
            businessInfo.getChildElements(
                soapFactory.createName("name",
                    "", "urn:uddi-org:api_v2"));
```

```
                    while (nameIterator.hasNext()) {
                        businessName =
                            (SOAPElement)nameIterator.next();
                        System.out.println("Company name: " +
                            businessName.getValue());
                          }
                          Iterator descriptionIterator =
                              businessInfo.getChildElements(
                                  soapFactory.createName(
                                      "description", "",
                                      "urn:uddi-org:api_v2"));
                          while (descriptionIterator.hasNext()) {
                        SOAPElement businessDescription =
                                  (SOAPElement)
                                  descriptionIterator.next();
                              System.out.println("Description: " +
                                  businessDescription.getValue());
                          }
                          System.out.println("");
                      }}

            // Konexioa itxi
            connection.close();

        } catch (Exception ex) {
            ex.printStackTrace();  }}}
```

Aplikazioak, adibidean egin dugun bezala, konexioa eta SOAP mezua eraikitzen ditu eta urruneko zerbitzariari bidaltzen dio. Urruneko zerbitzariaren informazioa `properties` motako fitxategitik irakurtzen da mezua bidali (`endpoint`) baino lehen:

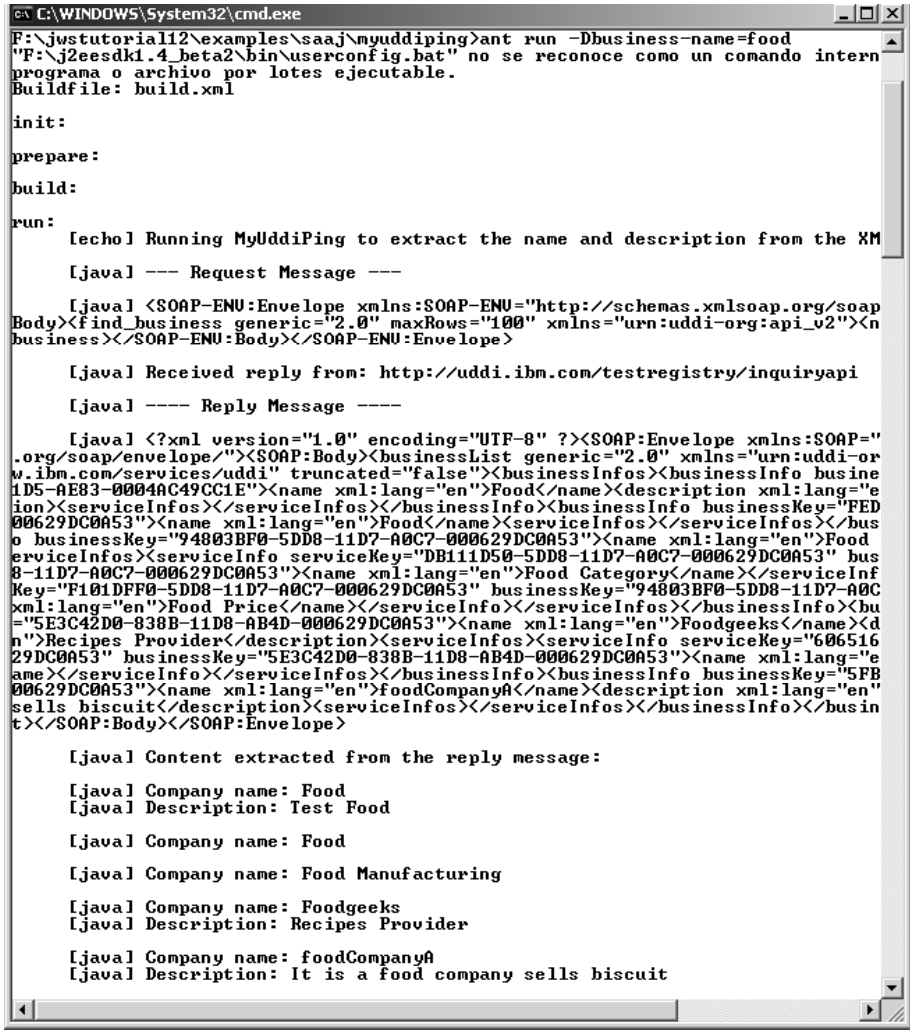
```
URL:http://uddi.ibm.com/testregistry/inquiryapi
http.proxyHost:
http.proxyPort:8080
```

Halaber, irteera estandarretik ateratzen du aplikazioak SOAP mezua, bidali baino lehen:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
                    <SOAP-ENV:Body>
<find_business generic="2.0" maxRows="100" xmlns="urn:uddi-org:api_v2">
                              <name>food</name>
                        </find_business>
                    </SOAP-ENV:Body>
              </SOAP-ENV:Envelope>
```

Zerbitzariak bidaltzen duen erantzuna XML dokumentu gordina da, gero, aplikazioak, irakurgarri bihurtuko duena. Zehatz-mehatz, SOAP mezuak, urruneko zerbitzariari, zerbitzu zehatza eskatzen dio: bilatzeko `food` izenarekin agertzen diren konpainia guztiak eta eskuratzeko bakoitzaren izena eta deskribapena.

Erantzuna jaso ondoren, nork bidali dion erantzuna eta erantzunaren informazio gordina irteera estandarretik ateratzen du aplikazioak berriz. SOAP mezuaren informazioa zatika prozesatzen du aplikazioaren azken zatiak eta mezuaren informazio adierazgarria irteera estandarretik ateratzen du. **5.12. irudia**k, aplikazioa egikarituta, irteera estandarretik ateratzen den informazioa jasotzen du.



5.12. irudia. Irteera estandarretik ateratzen den informazioa.

Atal honetan orain arte aztertu ditugun adibideetan, SOAP mezua exekuzio-garaian eraikitzen zen SAAJ APIaren objektu desberdinen `add()`, `get()`, `create()` eta `newInstance()` metodoak exekutatuta. Batzuetan, ostera, mezuaren gorputza XML dokumentu batean dugu biltegiratuta eta exekuzio-garaian irakurriko dugu, DOM paketea erabilita, DOM erako objektua eraikitzeko. DOM

objektua eraiki ondoren, nahikoa zaigu, `SOAPBody` motako objektuaren `addDocument()` metodoa exekutatzea `SOAPBodyElement` motako objektua eraikitzeko. Hona hemen DOM objektua eta SOAP mezua eraikitzeko beharko dugun iturburu-kodea:

```
static Document dokumentua= null;
DocumentBuilderFactory faktoria =
      DocumentBuilderFactory.newInstance();
  faktoria.setNamespaceAware(true);
  try {
     DocumentBuilder builder =
         faktoria.newDocumentBuilder();
         dokumentua = builder.parse( new File(args[0]) );

. . . . . .
MessageFactory faktoria = MessageFactory.newInstance();
SOAPMessage mezua = faktoria.createMessage();

SOAPHeader goiburua = mezua.getSOAPHeader();
SOAPBody gorputza = mezua.getSOAPBody();
      goiburua.detachNode();

// Gehitu DOM dokumentua mezuaren gorputzari
SOAPBodyElement dokElementua =
          gorputza.addDocument(dokumentua);
```

Azken estrategia honekin argi ikusten da XML dokumentuak ez daukala SOAP mezu osoaren informazioa, gorputzarena baizik. Mezu osoa XML dokumentuan islatu nahi badugu, estrategia zertxobait aldatu behar dugu:

```
static DOMSource domSource;
DocumentBuilderFactory faktoria =
      DocumentBuilderFactory.newInstance();
  faktoria.setNamespaceAware(true);
  try {
     DocumentBuilder builder =
         faktoria.newDocumentBuilder();
Document dokumentua = builder.parse( new File(args[0]) );
domSource = new DOMSource(dokumentua);
. . . . .
MessageFactory faktoria = MessageFactory.newInstance();
SOAPMessage mezua = faktoria.createMessage();

// Eskuratu mezuaren SOAPPart zatia
// eta finkatu bere edukia domSource balioarekin
 SOAPPart soapPart = mezua.getSOAPPart();
 soapPart.setContent(domSource);
```

Hau da, DOM paketearekin eraikitako `Document` motako objektua sarrerako parametro gisa erabiliko da `DOMSource` klasearen metodo eraikitzailean,

`DOMSource` motako objektua eraikitzeko. Gero, `DOMSource` motako objektu eraiki berria, SOAP mezuaren `SOAPPart` zatiari gehituko dio aplikazioaren exekuzioak.

Bukatzeko SAAJ APIari buruzko atal hau, azter dezagun nola eransten dion APIak informazioa SOAP mezuari gorputzaren bukaeran. Batzuetan, SOAP mezuan, informazio itsatsia bidali behar dugu XML ez den formatu batean. Horretarako, SOAP mezuari, `AttachmentPart` motako objektua itsatsiko zaio. Orain arte egin dugun bezala, hemen ere, `AttachmentPart` motako objektua eraikiko da eta `set()` metodoak erabilita edukiz osatuko da itsatsi nahi den zati hori. Bukatzeko, mezuaren `addAttachmentPart()` metodoa egikarituta, mezuan txertatzen da zati berria. Hona hemen, adibide txiki bat testu-fitxategiaren informazioa eransteko mezuari:

```
FileReader fr = null;
BufferedReader br = null;
String lerroa = "";

// Eraiki AttachmentPart motako objektua
AttachmentPart attachment1 =
message.createAttachmentPart();

 fr = new FileReader(new File(args[ 0] ));
 br = new BufferedReader(fr);

 String stringEdukia = "";
      lerroa = br.readLine();
      while (lerroa != null) {
          stringEdukia = stringEdukia.concat(lerroa);
          stringEdukia = stringEdukia.concat("\n");
          lerroa = br.readLine();
      }
      attachment1.setContent(stringEdukia, "text/plain");
      attachment1.setContentId("attached_text");
      mezua.addAttachmentPart(attachment1);
```

Iturburu-kodean ikusten den legez, parametro gisa datorkio testu-fitxategiaren izena aplikazio nagusiari, eta programak, fitxategia irakurri eta `String` motako objektua betetzen du fitxategiaren informazioarekin. Edukiak prest daudenean, `AttachmentPart` motako objektuaren `setContent()` motako metodoak egikaritzen dira erantsi nahi den zatia edukiz betetzeko. Azkenik, erantsi nahi den zatia mezuan txertatzen da, `addAttachmentPart()` metodoa egikarituta.

5.4. JAX-RPC (Java API for XML-based Remote Procedure Calls)

Orain arte aztertutakoarekin, jakin badakigu, web zerbitzuen testuinguru neutro eta agnostikoa antolatzeko, bi baliabide nagusi erabiltzen direla: XML dokumentuak eta SOAP mezuak. Teknologia horiek erabilita, zeharo errazten da plataformen arteko harremanak eratzea, inork ez baitu amore eman behar zerbitzu bat eskatzeko

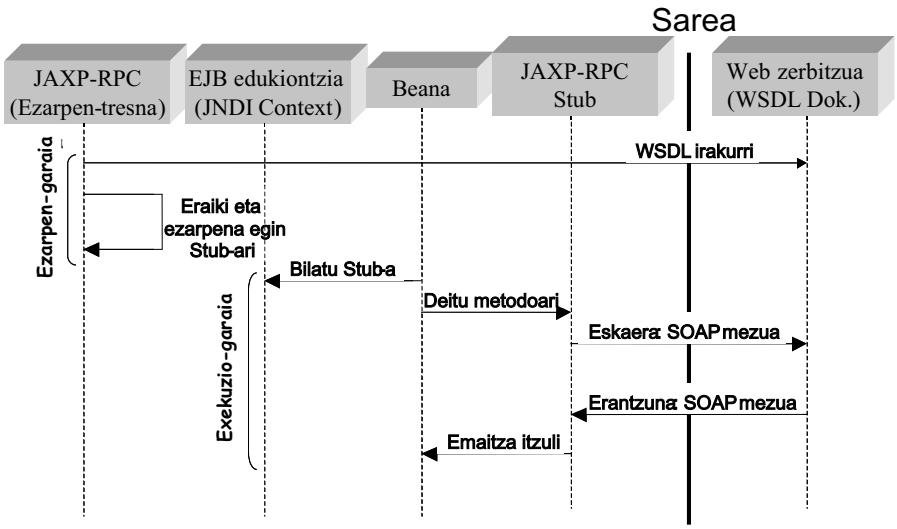
plataforma batek beste batekin elkarrizketa gauzatu nahi duenean. Testuinguru horretan, bezeroak, SOAP mezua eraikitzen du eta zerbitzariari bidaltzen dio zerbitzua eskatuz. Zerbitzariak, bere aldetik, SOAP mezua jaso eta egikaritu egiten du. Bukatzeko, zerbitzariak, erantzuna eraikitzen du SOAP mezu gisa eta bezeroari bidaltzen dio. Konturatu ezen testuinguru horretan, bezeroa J2EE teknologia erabiltzen egon daitekeela eta zerbitzariak zerbitzua, agian, .NET teknologia erabilita eraiki duela.

SOAP mezuaren estrategia erabiltzen bada zerbitzua bilatzeko eta egikaritzeko, argi dago, aplikazioa garatzen ari den programatzaileak berak eraiki eta kudeatu behar duela SOAP mezuaren egitura, aplikazioaren iturburu-kodea erabiliz. Egoera zertxobait aldatzen da SOAP mezuaren kudeaketa beste era batera antolatzen badugu. Bigarren estrategia horrekin, programatzaileak, ez du SOAP mezua zuzenean kudeatzen, Javaren azpiegitura teknologikoari pasatzen dio ardura hori. Gauzak errazteko, gainera, zerbitzuaren proxy lokal batekin lan egingo du programatzaileak zuzenean inongo formatu-bihurketarik eratu gabe. RMI teknologia menderatzen badu programatzaileak, ez du inongo arazorik izango zerbitzua eraikitzeko bigarren estrategia horrekin.

Estrategia horretan, bezeroak urruneko zerbitzariarekin harremanetan jarri nahi badu zerbitzua eskatzeko, urruneko prozedura-deia eraiki behar du aplikazio informatikoan. Javaren teknologiaren JAX-RPC (Java API for XML-based Remote Procedure Calls) APIa arduratuko da urruneko dei hori SOAP mezu gisa antolatzeaz, hau da, deia sarean zehar SOAP mezu gisa bidaliko da, baina programatzailearentzat erabat gardena izango da nola gauzatzen den lan hori. Hark, beti, azpiegitura teknologikoarekin, prozedura-dei mailan hitz egingo du eta ez du arduratu behar nola egituratzen den deia SOAP mezu erara.

Programatzaileak zerbitzuaren proxy lokal bat eraikiko du eta hori erabiliko du urruneko zerbitzariarekin harremanetan jartzeko. Kontu egin ezen estrategia bera erabiltzen zuela RMIk urruneko objektuetara atzitzeko. JAX-RPC APIarekin, berria, SOAP mezuan kokatzen da. Mezuak geruza neutro eta agnostikoa itsasten dio urruneko prozedura-deiari plataformen arteko komunikazioa eratzeko.

Web zerbitzua taxuz garatu nahi bada, aipatutako prozesuaren eragiketak une egokietan egikaritu behar dira. Eragiketa batzuk (**5.13. irudia**) ezarpen-garaian egikarituko dira eta beste batzuk exekuzio-garaian. Gogoratu ezen aplikazioak exekuzio-garaian zerbitzuaren proxy lokalarekin lan egingo duela, baina lehenago, proxy hori eraiki egin beharko da ezarpen-garaian. JAX-RPC teknologiak stub erara eraikiko du proxy-a eta, aplikazioak, exekuzio-garaian erabiliko du, SOAP mezuak bidaltzeko eta jasotzeko.



5.13. irudia. Prozesuaren exekuzio-fluxua.

Azter dezagun, bada, azaletik azaldu dugun prozesua nola gauzatzen den zehatz-mehatz. Web zerbitzua eraiki nahi badugu JAX-RPC teknologia erabilita, hiru urrats nagusi gauzatu behar ditugu:

1. Zerbitzariaren endpoint interfazea idatzi eta bere inplementazioa sortu.
2. Zerbitzuak behar dituen fitxategi osagarriak eraiki, konpilatu eta paketatu `war` motako fitxategi batean.
3. Ezarpena gauzatu bigarren urratsean eraikitako `war` motako fitxategiarekin.

Lehen urratsean adierazten dena betetzeko, RMI teknologiarekin egiten genuen eragiketa bera gauzatuko da, hau da, interfazea eraiki `Remote` interfazea hedatzen duena eta klasea diseinatu interfaze hori inplementatzen duena. Har dezagun gure betiko `Kaixo` adibidearen bertsio anglosaxoia eta eraiki dezagun web zerbitzua `String` motako objektua itzultzen duena. Hona hemen zerbitzuak erabiliko duen endpoint interfazea, urruneko prozedura-deiak zehazteko:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloIF extends Remote {
    public String sayHello(String s) throws RemoteException;
}
```

Interfazeak, urruneko zerbitzuak onartuko dituen prozedura-deien signatura zehazten du. Kontu egin, hemen ere, RMI teknologiarekin gertatzen zen bezala, `RemoteException` motako salbuespena altxa dezakeela metodo bakoitzak.

Lehen urratsa bukatzeko, hona hemen interfazeak izango duen inplementazioa:

```
public class HelloImpl implements HelloIF {

    public String mezua = "Kaixo ";

    public String sayHello(String s) {
        return mezua + s;
    }}
```

Bigarren urratsean, zerbitzua eraikitzen da lehen urratseko bi fitxategiak konpilatuta eta `wscompile` tresna erabilita. Tresnak erabiliko den zerbitzua definitzen du, `model.gz` fitxategia eraikita:

```
wscompile -define -d build -nd build
-classpath build config-interface.xml -model build/model.gz
```

Tresnak `config-interface.xml` XML dokumentua irakurtzen du zerbitzuaren nondik norakoak zehazteko:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration   xmlns="http://java.sun.com/xml/ns/jax-
rpc/ri/config">
                              <service   name="MyHelloService"
targetNamespace="urn:Foo"            typeNamespace="urn:Foo"
packageName="helloservice">
      <interface name="helloservice.HelloIF" />
      </service>
      </configuration>
```

XML dokumentu erako konfigurazio-fitxategiak zerbitzuaren izena adierazten du eta baita izenen espazioa, zerbitzuaren klaseak kokatuko diren paketearen izena eta `endpoint` interfazearen izena ere.

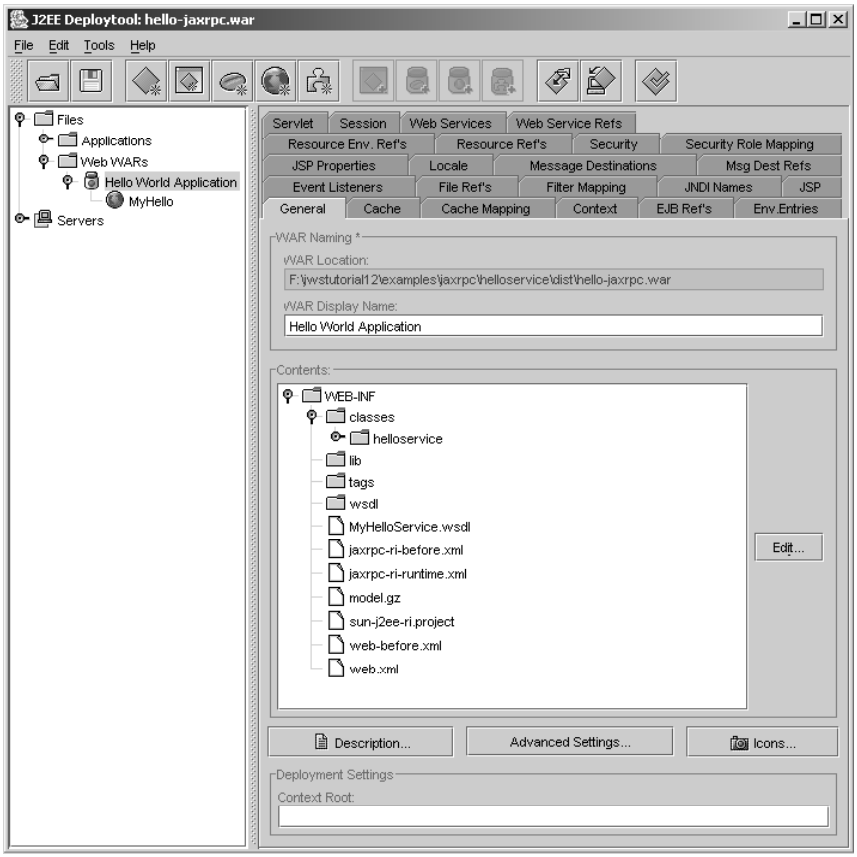
Fitxategi guztiak sortu ondoren, `war` motako fitxategia eraikitzen da `wsdeploy` tresna erabilita:

```
wsdeploy -o dist/hello-jaxrpc.war dist/hello-jaxrpc-portable.war
```

Tresnak WSDL motako fitxategia eraikitzen du, eta `hello-jaxrpc-portable.war` fitxategia erabilita, azken `hello-jaxrpc.war` fitxategia eraikitzen du (**5.14. irudia**).

Azken urratsa gauzatzeko, nahikoa zaigu `war` motako fitxategi berrriarekin zerbitzuaren ezarpena gauzatzea. Ezarpena ondo egikaritu dela egiaztatzeko, honako URL hau erabiltzen da:

```
http://localhost:8080/hello-jaxrpc/hello
```



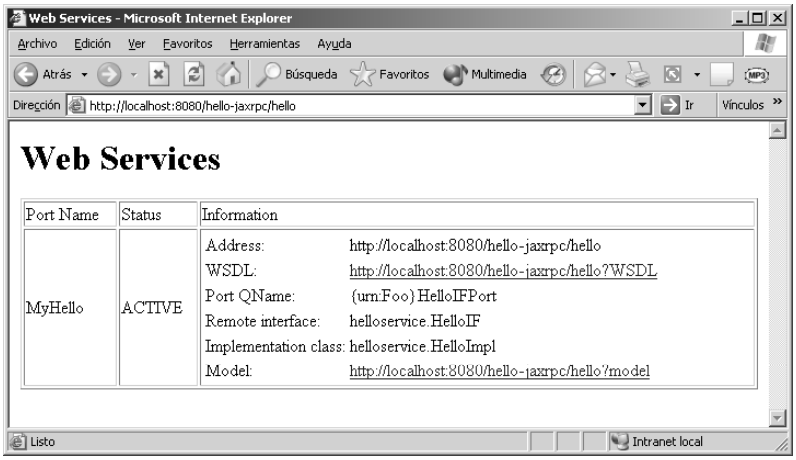
5.14. irudia. `war` motako baliabidearen ezaugarriak.

Dena ondo badoa, nabigatzaileak **5.15. irudia**n agertzen den orria bistaratuko du adieraziz web zerbitzuaren ataka-izena `MyHello` dela eta bere egoeraren balioa `ACTIVE`. `Information` zutabearen informazioak web zerbitzuaren zehaztasunak jasotzen ditu. **5.16. irudia**k, berriz, web zerbitzuaren WSDL informazioa adierazten du zerbitzuaren ezaugarriak argituz.

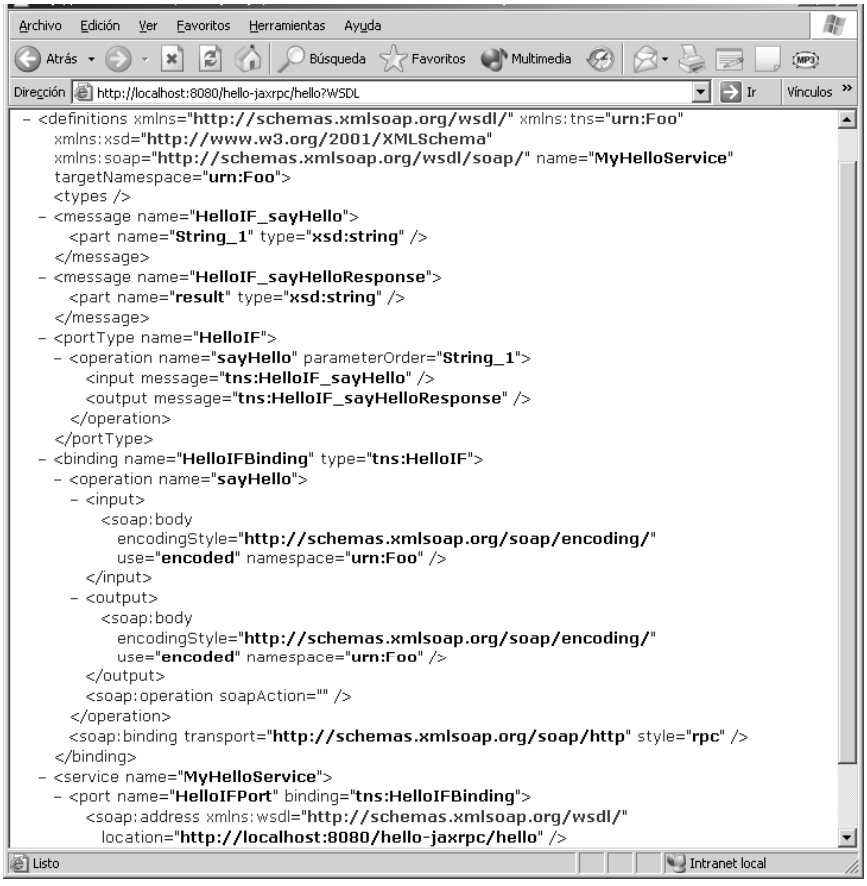
Hiru urratsak gauzatuta, web zerbitzua prest dago bezeroaren mezuak jasotzeko. Bezeroak zerbitzuaren urruneko prozedura atzitu nahi badu, hiru bide ditu:

1. Stub estatikoa erabili.
2. Proxy dinamikoa sortu.
3. Eskaera-interfaze dinamikoa (Dynamic Invocation Interfaze) eraiki.

Stub estatikoaren estrategia erabiltzen bada web zerbitzuaren urruneko prozedura atzitzeko, proxy lokal batekin ordezkatzen du aplikazioak urruneko zerbitzua eta, horretarako, `stub` erako objektu lokala erabiliko du. Aplikazioaren stub horiek ezarpen-garaian eraikitzen dira `wscompile` tresna erabilita.



5.15. irudia. Web zerbitzuaren ezaugarriak.



5.16. irudia. Web zerbitzuaren deskripzioa WSDL eran.

Hona hemen aplikazio txiki bat, stub estatikoaren estrategia erabiltzen duena urruneko zerbitzura atzitzeko:

```
import javax.xml.rpc.Stub;

public class HelloClient {

    private String endpointAddress;

    public static void main(String[] args) {

      System.out.println("Endpoint address = " + args[0]);
      try {
          Stub stub = createProxy();
        stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
               args[0]);
          HelloIF hello = (HelloIF)stub;
          System.out.println(hello.sayHello("Duke!"));
      } catch (Exception ex) {
          ex.printStackTrace();  }}

          private static Stub createProxy() {
        // MyHelloService_Impl(JX-RPC APIak eraikitzen du),
        // programatzaileak, zuzenean erabiltzen du.
        return (Stub)(new MyHelloService_Impl().getHelloIFPort());}}
```

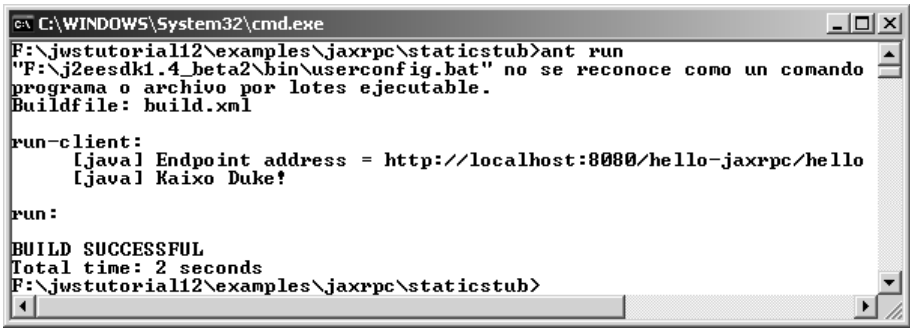
Lehendabizi, `Stub` motako objektua eraikitzen du aplikazioak, `createProxy()` metodo estatikoa egikarituta. Metodoak, ezarpen-garaian eraikitako `MyHelloService_Impl` klasearen `getHelloIFPort()` metodoa erabiltzen du `Stub` motako objektua sortzeko. Klase hori gardena da programatzailearentzat eta JX-RPC APIak eraikitzen du ezarpen-garaian, eta honako inplementazio partzial hau du:

```
public class MyHelloService_Impl extends BasicService
     implements MyHelloService
{

     public MyHelloService_Impl()
     {
         super(serviceName, new QName[] {
             ns1_HelloIFPort_QNAME }, (new
MyHelloService_SerializerRegistry()).getRegistry());
     }
. . . . . .

public HelloIF getHelloIFPort()
     {
         String as[] = new String[0];
         HandlerChainImpl handlerchainimpl = new
HandlerChainImpl(getHandlerRegistry().getHandlerChain(ns1_HelloIF
```

```
Port_QNAME));
         handlerchainimpl.setRoles(as);
         HelloIF_Stub helloif_stub = new
HelloIF_Stub(handlerchainimpl);
         try
  {
       helloif_stub._initialize(internalTypeRegistry);
  }
  catch(JAXRPCException jaxrpcexception)
  {
       throw jaxrpcexception;
  }
  catch(Exception exception)
  {
throw new JAXRPCException(exception.getMessage(), exception);
  }
  return helloif_stub;
  }. . . . . .}
```



5.17. irudia. Aplikazioaren emaitza irteera estandarrean.

`Stub` motako objektua eraiki ondoren, objektuaren `_setProperty()` metodoa egikaritzen da adierazteko stub objektuari zein izango den urruneko zerbitzuaren helbidea. Bukatzeko, `Stub` motako objektuari, datu-bihurketa aplikatzen zaio urruneko prozedura-deia egikaritu baino lehen. **5.17. irudia**k islatzen du aplikazioa egikarituta irteera estandarrean agertzen den informazioa.

Urruneko zerbitzuaren prozedura-deia egikaritu nahi badu bezeroak proxy dinamikoa erabilita, aplikazioak behar duen proxy-a deia gauzatzeko, exekuziogaraian eraikiko da. Horretarako, `ServiceFactory` motako objektuaren `createService()` metodoa egikaritzen da. Metodoak bi parametro eskatzen ditu zerbitzua eraikitzeko: WSDLren URLa eta `QName` motako objektu bat. Exekuziogaraian, zerbitzuarekiko informazioa, WSDLa erabilita eskuratzen du bezeroak. Gure adibidean, WSDL fitxategiaren URLa, hain zuzen, ezarpen-garaian zehaztu zena, WSDLra zuzentzen da:

```
http://localhost:8080/hello-jaxrpc/hello?WSDL
```

Zerbitzua eraiki eta gero, proxy-a sortzen da, `Service` motako objektuaren `getPort()` metodoa egikarituta. `Service` motako objektua, faktoria-objektu bat da proxy dinamikoak eraikitzeko. Hona hemen aplikazio txiki bat bezero mota hori inplementatzen duena:

```
import java.net.URL;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import dynamicproxy.HelloIF;

public class HelloClient {

    public static void main(String[] args) {
        try {

            String UrlString = args[0] + "?WSDL";
            String nameSpaceUri = "urn:Foo";
            String serviceName = "MyHelloService";
            String portName = "HelloIFPort";

            System.out.println("UrlString = " + UrlString);
            URL helloWsdlUrl = new URL(UrlString);

      ServiceFactory serviceFactory = ServiceFactory.newInstance();

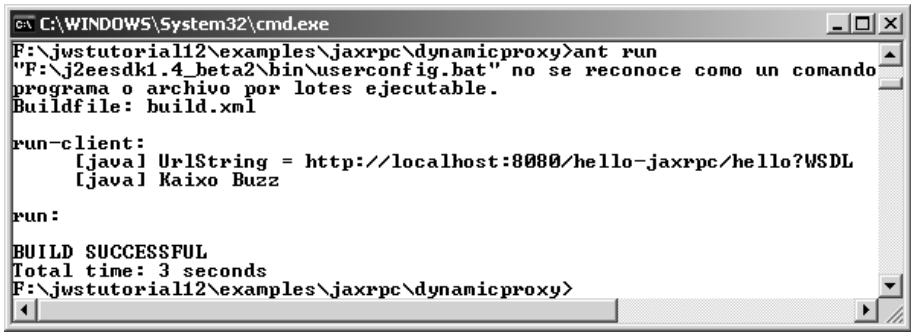
      Service helloService = serviceFactory.createService(helloWsdlUrl,
                new QName(nameSpaceUri, serviceName));

            dynamicproxy.HelloIF
                myProxy = (dynamicproxy.HelloIF)
            helloService.getPort(new QName(nameSpaceUri, portName),
                dynamicproxy.HelloIF.class);

            System.out.println(myProxy.sayHello("Buzz"));

        } catch (Exception ex) {
            ex.printStackTrace();   } } }
```

5.18. irudiak jasotzen ditu aplikazioak erabiltzen duen URLa eta urruneko prozedura egikarituta sortzen den informazioa.



5.18. irudia. Aplikazioa egikarituta sortzen den informazioa.

Amaitzeko, bezeroak urruneko zerbitzu batera atzitzeko erabil dezakeen hirugarren estrategia aztertuko dugu. Estrategia horrek ez ditu behar `wscompile` tresnak eraikitzen dituen klaseak, eta ia dena dinamikoki sortuko da exekuzio-garaian. Bezeroak, urruneko prozedura bati deitzeko, urruneko prozeduraren signatura edo zerbitzuaren izena sortuko ditu exekuzio-garaian. Hona hemen, aplikazio bat eskaera-interfaze dinamikoa (Dynamic Invocation Interface) estrategia erabiltzen duena urruneko zerbitzua egikaritzeko:

```
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

public class HelloClient {

    private static String qnameService = "MyHelloService";
    private static String qnamePort = "HelloIF";

    private static String BODY_NAMESPACE_VALUE =
        "urn:Foo";
    private static String ENCODING_STYLE_PROPERTY =
         "javax.xml.rpc.encodingstyle.namespace.uri";
    private static String NS_XSD =
        "http://www.w3.org/2001/XMLSchema";
    private static String URI_ENCODING =
         "http://schemas.xmlsoap.org/soap/encoding/";

    public static void main(String[] args) {

        System.out.println("Endpoint address = " + args[0]);

        try {
            ServiceFactory factory =
                ServiceFactory.newInstance();
```

```
            Service service =
                factory.createService(new QName(qnameService));

            QName port = new QName(qnamePort);

            Call call = service.createCall(port);
            call.setTargetEndpointAddress(args[0]);

            call.setProperty(Call.SOAPACTION_USE_PROPERTY,
                new Boolean(true));
            call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
            call.setProperty(ENCODING_STYLE_PROPERTY, URI_ENCODING);
            QName QNAME_TYPE_STRING = new QName(NS_XSD, "string");
            call.setReturnType(QNAME_TYPE_STRING);


            call.setOperationName(new QName(BODY_NAMESPACE_VALUE,
                "sayHello"));
            call.addParameter("String_1", QNAME_TYPE_STRING,
                ParameterMode.IN);
            String[] params = { "Murph!" };

            String result = (String)call.invoke(params);
            System.out.println(result);

        } catch (Exception ex) {
            ex.printStackTrace();  }}}
```

Kontu egin ezen estrategia horretan deia exekuzio-garaian eratzen dela `Call` motako objektua eraikita. Objektu hori eraiki baino lehen, `Service` motako objektua eraikiko da lehenengo, `ServiceFactory` motako objektuaren `createService()` metodoa egikarituta. Metodoak eskatzen duen parametro bakarra `QName` motako objektua da. Objektu horrek balio du zerbitzuaren izena adierazteko eta WSDL fitxategian `service` elementuan egoten da zehaztuta:

```
<service name = "MyHelloService">
```

Zerbitzua eraiki ondoren, `Call` motako objektua sortzen da `Service` motako objektuaren `createCall()` metodoa egikarituta. Zerbitzuaren urruneko prozeduraren eskaera dinamikoa onartzen du `Call` motako objektu horrek. Objektu mota hori eraikitzeko, `QName` motako objektua du `createCall()` metodoak.

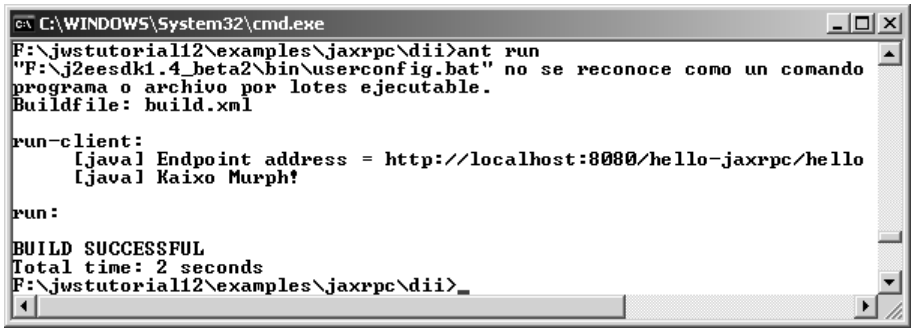
`QName` motako objektuak zerbitzuaren endpoint interfazea adierazteko balio du, eta WSDL fitxategian `portType` elementuan egoten da zehaztuta:

```
<portType name = "HelloIF">
```

Konturatu ezen gauzatuko den deiaren ezaugarriak poliki-poliki zatika ari garela eraikitzen. Hurrengo urratsean, `Call` motako objektuaren `setTargetEndpointAddress()` metodoa egikarituta, zerbitzuaren helbidea

adierazten da. Hori egin ondoren, `Call` motako objektuaren `set()` metodoak egikarituta, honako ezaugarri hauek adierazten dira: deiaren propietateak eta urruneko metodoaren izena, parametroak eta metodoak itzultzen duen datu mota.

Deiaren ezaugarri guztiak adierazi ondoren, `Call` motako objektuaren `invoke()` metodoa egikaritzen da urruneko zerbitzua eskatzeko. Dena ondo egikaritzen bada, irteera estandarrean, **5.19. irudia**n agertzen den informazioa ikusiko da.



5.19. irudia. Aplikazioa egikarituta irteera estandarretik ateratzen den informazioa.

Kontu egin ezen orain arte erabili dugun bezero mota aplikazio arrunta zela. Horren ordez, JSP orria erabili nahi badugu, hau da, web aplikazioa erabili nahi badugu bezeroaren aldean, ez dugu inongo aldaketarik egin behar urruneko zerbitzua erabiltzeko. Demagun gure urruneko zerbitzua stub estatikoa erabilita atzitu nahi dugula. Hona hemen hori egiteko egikaritu beharko ditugun JSP orriak:

```
<html>
<head><title>Kaixo</title></head>
<body bgcolor="white">
<img src="bart.gif">
<h2>Kaixo, nire izena Bart da. Zein da zurea?</h2>
<form method="get">
<input type="text" name="username" size="25">
<p></p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

<%
    String username = request.getParameter("username");
    if ( username != null && username.length() > 0 ) {
%>
    <%@include file="erantzuna.jsp" %>
<%
    }
%>
</body>
</html>
```

oOo

```
<%@ page import="javax.xml.rpc.Stub,javax.naming.*,webclient.*"
%>
<%
    String erantzuna = null;
    try {
Stub stub = (Stub)(new MyHelloService_Impl().getHelloIFPort());

  stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
         "http://localhost:8080/hello-jaxrpc/hello");
      HelloIF hello = (HelloIF)stub;
      erantzuna =
hello.sayHello(request.getParameter("username"));

    } catch (Exception ex) {
        resp = ex.toString();
    }
%>
<h2><font color="black"><%=erantzuna%>!</font></h2>
```



5.20. irudia. Urruneko zerbitzuak bidalitako erantzunaren JSP orria.

Lehenengo, ongi etorria eman eta testu-eremua bistaratzen du JSP orriak, erabiltzaileak bere izena sar dezan. Izena idatzi ondoren, bigarren JSP orria egikaritzen da urruneko zerbitzuaren prozedura egikaritzeko. Urruneko zerbitzura

atzitzeko, proxy lokalaren estrategia erabiliko da eta, horretarako, stub estatikoa eraikiko da iturburu-kodean ikusten den legez. Amaitzeko, urruneko zerbitzuak bidalitako erantzuna (**5.20. irudia**) JSP orriaren beheko aldean agertzen da.

5.5. JAXR (Java API for XML Registries)

Zerbitzu hornitzaileak, web zerbitzua sortu duenean, erregistratu egin behar du nonbait, web zerbitzu horren bezeroak aurki dezan. Javaren teknologiak JAXR (Java API for XML Registries) APIa eskaintzen dio programatzaileari erregistroak eraikitzeko eta Internet bidez bilatzeko. XML erregistroek espezifikazio estandarrak erabiltzen dituzte erregistroaren informazioa zehazteko; bi ezagunenak, ebXML (Electronic Business using eXtensible Markup Language) eta UDDI (Universal Description, Discovery and Integration) dira.

Beraz, XML erregistro batera atzitu nahi duenean bezeroak, zerbitzu-hornitzailearekin harremanetan jarriko da konexioa eraikita, eta konexioa sortu ondoren, eskaera bat osatuko du. Eskaerak, tankera askotakoak izan daitezke, batzuetan, erakundeak bilatu nahi izaten dira `findOrganizations()` metodoa erabilita; beste batzuetan, zerbitzuak dira bilatu nahi direnak `findServices()` metodoa egikarituta. Azkenik, `findServicesBindings()` metodoaren bidez zerbitzura atzitzeko behar duen informazioa eskuratzen da.

Azter dezagun zehatz-mehatz nola eraikitzen den konexioa JAXR APIa erabilita. Lehenago egin dugun bezala, hemen ere, lehenengo eta behin, faktoria bat eraikiko da, `ConnectionFactory` klasearen `newInstance()` metodoa egikarituta. Faktoria eraiki ondoren, konexioak izango dituen propietateak zehazten dira. Hori egiteko, `setProperties()` metodoa egikaritzen da, metodoan `Properties` motako objektua parametro gisa ipinita. Metodoa egikaritu ondoren, `createConnection()` metodoa egikaritzen da `Connection` motako objektua eraikitzeko. Hona hemen prozesu guztia iturburu-kodera itzulita:

```
/**
     * Establishes a connection to a registry.
     *
     * @param queryUrl     the URL of the query registry
     * @param publishUrl   the URL of the publish registry
     */
    public void konexioaOsatu(String queryUrl,String publishUrl) {

        /*
         * Specify proxy information in case you
         *  are going beyond your firewall.
         */
        ResourceBundle registryBundle =
           ResourceBundle.getBundle("JAXRExamples");
        String httpProxyHost =
registryBundle.getString("http.proxyHost");
```

```
        String httpProxyPort =
registryBundle.getString("http.proxyPort");

        /*
         * Define connection configuration properties.
         * For simple queries, you need the query URL.
         */
        Properties props = new Properties();
        props.setProperty("javax.xml.registry.queryManagerURL",
            queryUrl);
        props.setProperty("com.sun.xml.registry.http.proxyHost",
            httpProxyHost);
        props.setProperty("com.sun.xml.registry.http.proxyPort",
            httpProxyPort);
  try {
            // Create the connection, passing it the
            // configuration properties
            ConnectionFactory factory =
                ConnectionFactory.newInstance();
            factory.setProperties(props);
            connection = factory.createConnection();
            System.out.println("Created connection to registry");
        } catch (Exception e) {
            e.printStackTrace();
            if (connection != null) {
                try {
                    connection.close();
                } catch (JAXRException je) {}  }}}
```

Konexioa eraikita, eta ez bada salbuespenik altxatu, aplikazioa prest dago eskaerak gauzatzeko. Lehenengo eta behin, `RegistryService` motako objektua eraikiko da konexioaren `getRegistryService()` metodoa egikarituta. Objektu hori eraiki ondoren, `BusinessQueryManager` motako objektua sortzen da `RegistryService` motako objektuaren `getBusinessQueryManager()` metodoa egikarituta.

Bilaketa gauzatu baino lehen, bilaketa nola egituratuko den eta zer den bilatuko dena adierazten dira. Horretarako, `Collection` motako bi objektu eraikitzen dira, gero parametro gisa itsatsiko direnak `find()` motako metodoan. Bilaketaren emaitzak `BulkResponse` motako objektuan jasotzen dira. Objektu horren `getCollection()` metodoa egikaritzen bada, `Collection` motako objektuan eskuratzen dira emaitzak. Hona hemen, prozesu guztiaren iturburu-kodea:

```
/**
     * Searches for organizations containing a string and
     * displays data about them.
     *
     * @param qString      the string argument
     */
```

```
public void egikarituGaldera(String qString) {
    RegistryService rs = null;
    BusinessQueryManager bqm = null;

    try {
        // Get registry service and query manager
        rs = connection.getRegistryService();
        bqm = rs.getBusinessQueryManager();
        System.out.println("Got registry service and " +
            "query manager");

        // Define find qualifiers and name patterns
        Collection findQualifiers = new ArrayList();
        findQualifiers.add(FindQualifier.SORT_BY_NAME_DESC);
        Collection namePatterns = new ArrayList();
        namePatterns.add("%" + qString + "%");

        // Find using the name
        BulkResponse response =
            bqm.findOrganizations(findQualifiers,
                namePatterns, null, null, null, null);
        Collection orgs = response.getCollection();
```

Bilatutako erakundeen informazioa bistaratu nahi bada, aski da iteratzaile bat eraikitzea eta iteratzaile horren elementuak eskuratzea:

```
// Display information about the organizations found
      Iterator orgIter = orgs.iterator();
      if (!(orgIter.hasNext())) {
          System.out.println("No organizations found");
      } else while (orgIter.hasNext()) {
          Organization org = (Organization) orgIter.next();
          System.out.println("Org name: " + getName(org));
          System.out.println("Org description: " +
              getDescription(org));
          System.out.println("Org key id: " + getKey(org));
      .......
 /**
 * Returns the name value for a registry object.
 *
 * @param ro   a RegistryObject
 * @return           the String value
 */
private String getName(RegistryObject ro)
    throws JAXRException {

    try {
        return ro.getName().getValue();
    } catch (NullPointerException npe) {
        return "No Name";  }}
 /**
```

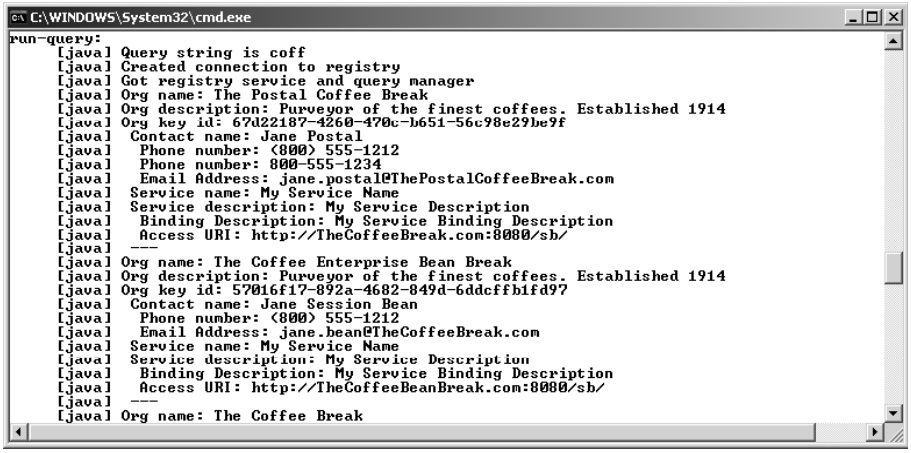
```
      * Returns the key id value for a registry object.
      *
      * @param ro   a RegistryObject
      * @return            the String value
      */
      private String getKey(RegistryObject ro)
        throws JAXRException {

        try {
            return ro.getKey().getId();
        } catch (NullPointerException npe) {
            return "No Key";}}
.......}
```

Gure adibideak egikaritzeko honako propietateen fitxategi hau erabili dugu:

```
## Egikaritu nahi diren <query> eta <publish> URLei
## kendu ohar-ikurra
## IBM:
#query.url=http://uddi.ibm.com/testregistry/inquiryapi
#publish.url=https://uddi.ibm.com/testregistry/publishapi
## Microsoft:
query.url=http://test.uddi.microsoft.com/inquire
publish.url=https://test.uddi.microsoft.com/publish
## Registry Server:
#query.url=http://localhost:8080/RegistryServer
#publish.url=http://localhost:8080/RegistryServer
## Adierazi erabiltzailea eta pasahitza
## Registry Server-entzat (testuser/testuser)
registry.username=testuser
registry.password=testuser
## HTTP eta HTTPS proxy host-a eta ataka
http.proxyHost=
http.proxyPort=8080
https.proxyHost=
https.proxyPort=8080
## Erabiltzen diren balioak
org.name=The Coffee Break
org.description=Purveyor of the finest coffees. Established 1912
person.name=Jane Doe
phone.number=(800) 555-1212
email.address=jane.doe@TheCoffeeBreak.com
classification.scheme=ntis-gov:naics
classification.name=Snack and Nonalcoholic Beverage Bars
classification.value=722213
service.name=My Service Name
service.description=My Service Description
svcbinding.description=My Service Binding Description
svcbinding.accessURI=http://TheCoffeeBreak.com:8080/sb/
..............
```

Ikusten den legez, IBM eta Microsoft erakundeen web guneetara bidera daiteke JAXR aplikazioa, bertako XML erregistroak arakatzeko. **5.21. irudia**n, Microsoft erakundearen web gunera joanda, zerbitzariak itzuli digun informazio partziala jasotzen da.



5.21. irudia. Microsoft web guneak itzultzen duen informazioa.

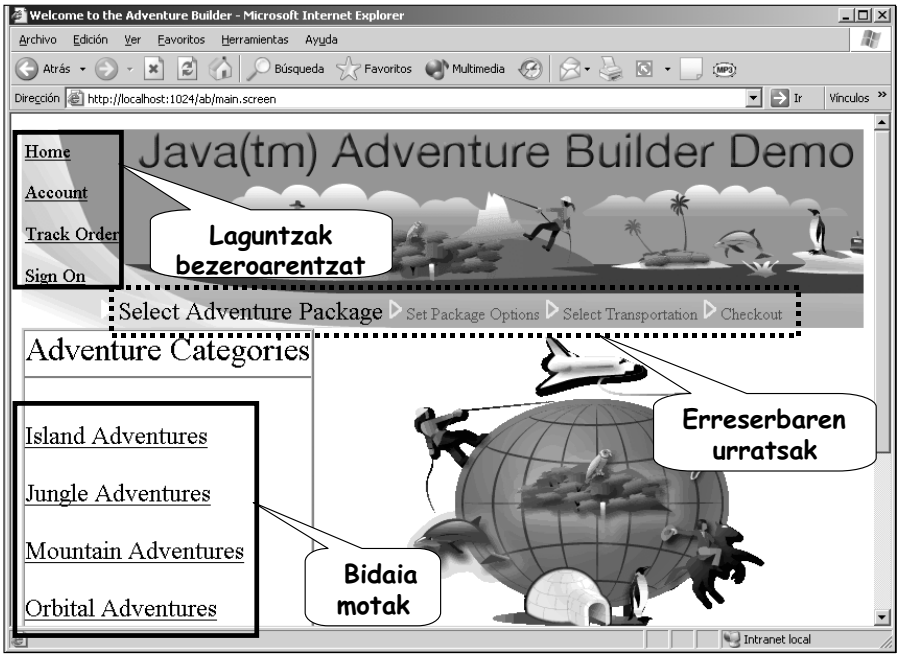
Kontu egin ezen erakundearen zerbitzuak eta zerbitzuen ezaugarriak eskuratu nahi baditugu, nahikoa dela `Organization` objektuaren `getServices()` metodoa egikaritzea zerbitzuak lortzeko, eta `Service` motako objektuaren `getServicesBindings()` metodoa egikaritzea, zerbitzuaren zehaztasunen berri jakiteko:

```
Iterator orgIter = orgs.iterator();
while (orgIter.hasNext()) {
   Organization org = (Organization) orgIter.next();
   Collection services = org.getServices();
   Iterator svcIter = services.iterator();
     while (svcIter.hasNext()) {
        Service svc = (Service) svcIter.next();
        Collection serviceBindings =
          svc.getServiceBindings();
        Iterator sbIter = serviceBindings.iterator();
        while (sbIter.hasNext()) {
          ServiceBinding sb =(ServiceBinding) sbIter.next();
}}}
```

5.6. ADIBIDE KONPLEXU BATEN DISEINUA ETA GARAPENA WEB ZERBITZUAK ERABILITA

Kapituluaren atarian egoera estandarra aurkeztu dugu web zerbitzuen beharra argudiatzen zuena. Egoerak, bidaia-agentzia baten prozesu orokorraren nondik norakoak azaltzen zituen, arrazoitzeko, egoera horietan web zerbitzuen aukera

egokia izan zitekeela, aplikazio informatikoa garatzeko erakunde horretan. Sun erakundeak ere hala pentsatu zuen eta Adventure Builder aplikazio kanonikoa[73] diseinatu, garatu eta eskuragarri jarri zuen, edozeinek aztertzeko nola egikaritzen diren Javaren teknologiak eskaintzen dituen APIak, adibide konplexu eta landu batean.



5.22. irudia. Adibidearen hasierako interfaze grafikoa.

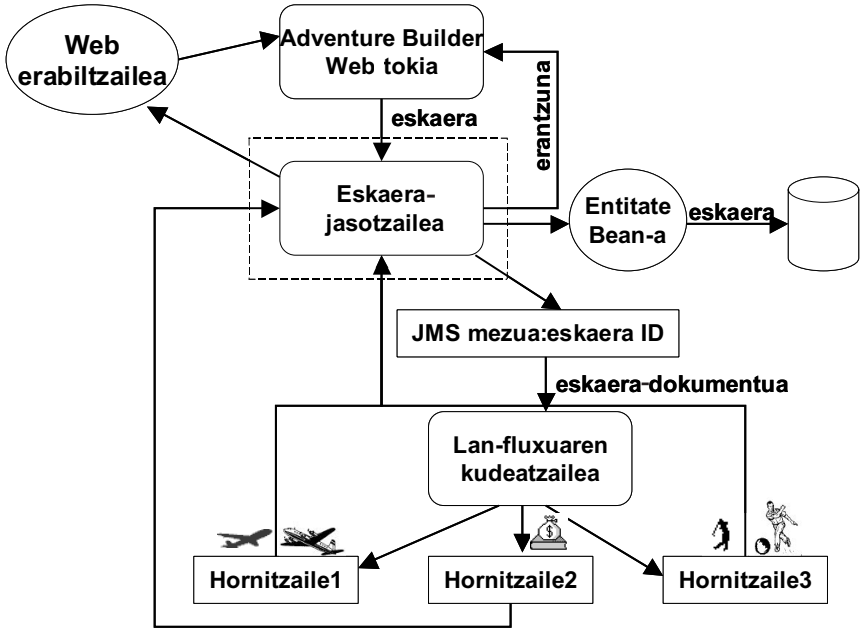
Bidaia-agentzia hipotetiko baten bidaia-erreserbak kudeatzen ditu adibideak web orriak erabilita. Hasierako web orriak (**5.22 irudia**) hiru atal ditu eta erreserbaren hiru funtzionalitate prozesatzeko erabiltzen dira:

1. Goiko aldean agertzen den atalaren bidez, erreserbaren bezeroak eta aplikazioaren administratzaileak bezeroaren kontua eta eskaeraren egoera azter ditzakete, jakiteko, nork eta nola gauzatu duen erreserba.
2. Leihoaren ezkerraldean, bidaia-agentziak eskaintzen dituen bidaia motak agertzen dira. Horietako bat aukeratu ondoren, erreserbaren esleipenaren prozesua martxan hasten da bezeroarentzat.
3. Bidaia mota aukeratu ondoren, leiho berria (**5.25. irudia**) irekitzen zaio bezeroari aurrekonfiguratutako bidaia-pakete bat aukeratu dezan. Une oro, aplikazioak, leihoaren erdialdean, erreserbaren zein urratsetan dagoen adierazten dio bezeroari, gogoratzeko, zer bete behar duen une horretan eta zenbat urrats gelditzen zaizkion bidaiaren erreserba bukatzeko.

73. `http://java.sun.com/developer/releases/adventure/`

Bidaia-agentziak zerbitzu integrala eskaini nahi duenez, bezeroak paketea aukeratzen duenean ez du bakarrik egonaldiko hotela erreserbatzen, horrez gain, egonaldiko jarduerak eta garraioa ere etxetik antolatzen ditu.

Jakina, zerbitzu horiek guztiak, bidaia-agentziak banaka erreserbatu behar ditu eta gainera, hornitzaileak erabilita. Hornitzaileek, bestalde, beren lan-fluxua izango dute, aspaldian finkatuta, erreserbak kudeatzeko. Eta zertxobait gehiago nahasteko zokondoak, erabiltzen duten aplikazio informatikoa, plataforma eta teknologia desberdinekin garatu dute.



5.23. irudia. Aplikazioaren lan-fluxua.

Argi dago, egoera horretan, bezero-zerbitzari arkitektura arrunta ez dela nahikoa aplikazioa diseinatzeko. Bezeroak, eskaera bat bidaltzen duenean bidaia-agentziaren web gunera (**5.23. irudia**), eskaera hori, eskaera prozesatuko duen moduluak hartzen du eta J2EE teknologia erabilita, entitate-bean bat eraikitzen du eta edukiontziak kudeatutako iraunkortasuna aplikatzen dio eskaeraren informazioa biltegiratzeko. Eskaera biltegiratu eta gero, eskaera jaso duen azpimoduluak, eskaera-dokumentu bat eraikitzen du eta lan-fluxua kudeatuko duen moduluari bidaltzen dio, erreserbak izango duen lan-fluxua antola dezan. Orduan hasten da prozesuko lanik gogorrena: lan-fluxuaren kudeatzaileak, SOAP mezuak eraiki eta hornitzaileei bidaltzen dizkie zerbitzu desberdinak eskatuz. Hornitzaileak bere aldetik, mezua jaso, prozesatu eta erantzuna bidaltzen du XML eta SOAP teknologiak erabilita. Erantzun guztiak jasota eta erreserbaren bideragarritasuna egiaztatuta, mezua bidaltzen dio aplikazioak bezeroari bere posta elektronikoa erabilita.

Aurkeztutako prozesu orokorra gauzatzeko, Eredu-Bista-Kontrol arkitektura modularra erabiltzen da. Bestalde, aplikazioaren zehaztasun funtzionalak tentuz diseinatu dira eta hainbat J2EE diseinu-patroiz (**5.24. irudia**) baliatzen da aplikazioa, eskaera funtzional desberdinak kudeatzeko. Patroi horien guztien portaera koordinatzeko Eredu-Bista-Kontrol patroi ezaguna antolatu da, informazio-fluxua bideratzeko aplikazioaren bazter guztietara.

Aplikazioak eskaera jasotzen duenean, kontrolatzaileari pasatu baino lehen, beharrezkoak diren aurretratamenduak egikaritzen ditu. Batez ere, segurtasun-neurriak aplikatzen ditu. Horretarako, XML fitxategi bat erabiltzen du jakiteko nolakoa izango den aplikatu behar duen segurtasun-politika. Hona hemen Adventure Builder aplikazioak erabiltzen duen aipatutako XML fitxategia:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<signon-config>

 <!– Form Sign On Page–>
 <signon-form-login-page>signon.screen</signon-form-login-page>

 <!– Error Page When Sign On fails–>
 <signon-form-error-page>signon_failed.screen</signon-form-error-page>

 <!– A Protected Resource–>
 <security-constraint>
   <web-resource-collection>
    <web-resource-name>Account information</web-resource-name>
    <url-pattern>customer.do</url-pattern>
   </web-resource-collection>
 </security-constraint>

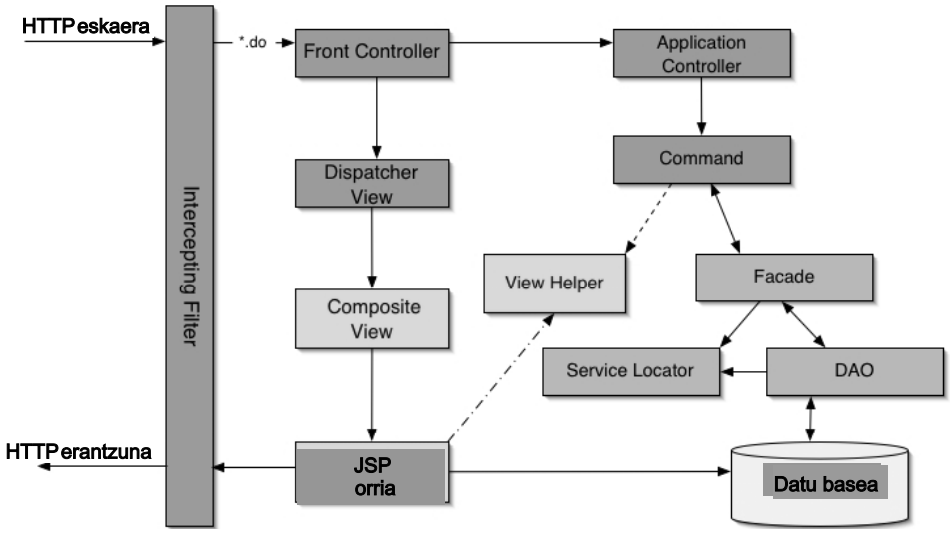
 <!– A Protected Resource–>
 <security-constraint>
   <web-resource-collection>
    <web-resource-name>Signon</web-resource-name>
    <url-pattern>signon_welcome.screen</url-pattern>
   </web-resource-collection>
 </security-constraint>

 <!– A Protected Resource–>
 <security-constraint>
   <web-resource-collection>
    <web-resource-name>Order Information Screen</web-resource-name>
    <url-pattern>enter_order_info.screen</url-pattern>
   </web-resource-collection>
 </security-constraint>

 <!– A Protected Resource–>
 <security-constraint>
   <web-resource-collection>
```

```
    <web-resource-name>Checkout</web-resource-name>
    <url-pattern>checkout.do</url-pattern>
   </web-resource-collection>
 </security-constraint>
</signon-config>
```

Eskaerak hasierako iragazki hori igarotzen badu, Front Controller patroia aktibatzen du eskaeraren premiak prozesatzeko. Patroi hori aplikazioaren servlet nagusi batek inplementatuko du eta eskaeraren prozesu guztiaren ardura, bere esku egongo da. Kontrolatzaileak, ikusita eskaera bat prozesatzea ez duela txantxetan hartu behar eta jakinda beste hainbat lan egin behar dituela, aplikazioaren kontrolatzaileari delegatzen dio eskaeraren prozesu zehatza.



5.24. irudia. Aplikazioak erabiliko dituen diseinu-patroiak.

Kanpotik jaso diren eskaerak mapeatu egiten dira egikarituko diren ekintzekin aplikazioan. Eragiketa hori gauzatzeko, XML fitxategi bat erabiltzen du aplikazioaren kontrolatzaileak:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mappings>
   <url-mapping url="cart.do" useFlowHandler="true">
    <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.CartHTMLAction</web
-action-class>
        <flow-handler
class="com.sun.j2ee.blueprints.adventure.web.handlers.CartFlowHandler"
>
            <handler-result result="SELECT_TRANSPORT"
screen="transportation.screen"/>
            <handler-result result="ENTER_PACKAGE_DETAILS"
```

```
screen="enter_package_details.screen" />
             <handler-result result="RESERVE_HOTEL"
screen="reserve_hotel.screen" />
            <handler-result result="PURCHASE_ACTIVITIES"
screen="purchase_activities.screen" />
            <handler-result result="CART" screen="cart.screen" />
           <handler-result result="CART-ACTIVITIES"
screen="cart.screen?tab=activities" />
            <handler-result result="CART-LODGING"
screen="cart.screen?tab=lodging" />
            <handler-result result="CART_TRANSPORT"
screen="cart.screen?tab=transportation" />
            <handler-result result="PURCHASE_MORE_ACTIVITIES"
screen="purchase_more_activities.screen" />
            <handler-result result="CANCEL" screen="main.screen" />
        </flow-handler>
   </url-mapping>

   <url-mapping url="searchtransport.do"
screen="available_transport.screen" >
        <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.TransportSearchHTM
LAction</web-action-class>
   </url-mapping>

   <url-mapping url="ordertracking.do"
screen="order_tracking_result.screen" >
        <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.OrderTrackingHTMLA
ction</web-action-class>
   </url-mapping>

   <url-mapping url="checkout.do" screen="checkout.screen" >
        <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.CheckoutHTMLAct
ion</web-action-class>
   </url-mapping>

   <url-mapping url="customer.do" screen="account.screen" >
        <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.CustomerHTMLAction
</web-action-class>
   </url-mapping>

   <url-mapping url="createsignon.do" screen="create_account.screen" >
        <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.CreateSignOnHTMLAc
tion</web-action-class>
   </url-mapping>
```

```
   <url-mapping url="signoff.do" screen="signoff.screen" >
        <web-action-
class>com.sun.j2ee.blueprints.adventure.web.actions.SignOffHTMLAction
</web-action-class>
   </url-mapping>
</mappings>
```

Mapaketa gauzatu ondoren, Command patroiak egikaritzen du eskaerari esleitzen zaion ekintza. Ekintzak baliabideak eskuratu behar baditu, Façade patroia eraikiko da Command patroiari laguntzeko betebehar horretan. Façade patroiak, bere aldetik, zerbitzuak eta baliabideak eskuratzeko, Service Locator eta DAO (Data Access Object) patroi ezagunak erabiltzen ditu. Kontu egin ezen zerbitzu eta baliabide horien izaera polimorfikoa izan daitekeela, hala nola JNDI baliabide bat, EJB osagai bat edo JDBC konexio bat izan daiteke.

Kontrolatzaile nagusia ohartzen denean ekintza egikaritu dela eta emaitzak prest daudela bezeroari bidaltzeko, Eredu-Bista-Kontrol patroiaren bista-arloa antolatzen du HHTP erantzun egokia eraikitzeko. Horretarako, dispatching eragiketa bat abiarazten da eta CompositeView patroia eraikitzen du, emaitzaren informazioa, JSP orri batera egokitzeko. Egokitze-prozesua gauzatzeko, txantiloi bereziak erabiltzen ditu CompositeView patroia inplementatzen duen servletak, formatu egokia aplikatzeko emaitza bakoitzari. Hona hemen txantiloien egitura adierazteko eraikitzen den XML fitxategiaren zati partziala:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<screen-definitions>
  <default-template>/template.jsp</default-template>

  <screen name="main">
    <parameter key="title" value="Welcome to the Adventure Builder"
direct="true"/>
    <parameter key="banner" value="/banner.jsp" />
    <parameter key="sidebar" value="/sidebar.jsp" />
    <parameter key="body" value="/main.jsp" />
    <parameter key="footer" value="/footer.jsp" />
   </screen>

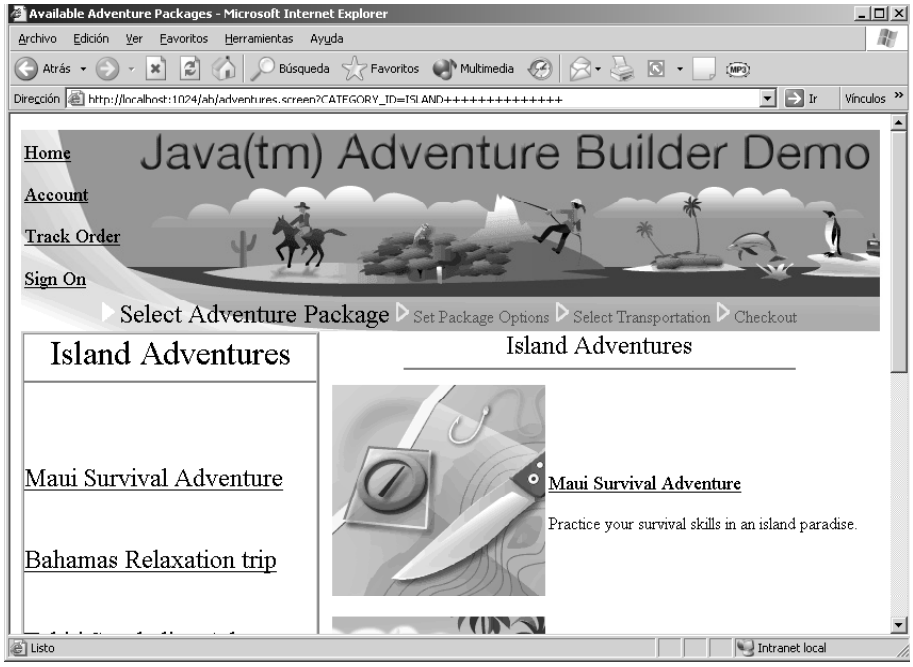
  <screen name="cart">
    <parameter key="title" value="Adventure Package" direct="true"/>
    <parameter key="banner" value="/banner.jsp" />
    <parameter key="sidebar" value="/sidebar.jsp" />
    <parameter key="body" value="/cart.jsp" />
    <parameter key="footer" value="/footer.jsp" />
   </screen>

  <screen name="adventures">
      <parameter key="title" value="Available Adventure Packages"
direct="true"/>
```

```
    <parameter key="banner" value="/banner.jsp" />
    <parameter key="sidebar" value="/sidebar.jsp" />
    <parameter key="body" value="/adventures.jsp" />
    <parameter key="footer" value="/footer.jsp" />
   </screen>

  <screen name="adventure">
      <parameter key="title" value="Adventure Package Details"
direct="true"/>
    <parameter key="banner" value="/banner.jsp" />
    <parameter key="sidebar" value="/sidebar.jsp" />
    <parameter key="body" value="/adventure.jsp" />
    <parameter key="footer" value="/footer.jsp" />
   </screen>
     ............
</screen-definitions>
```

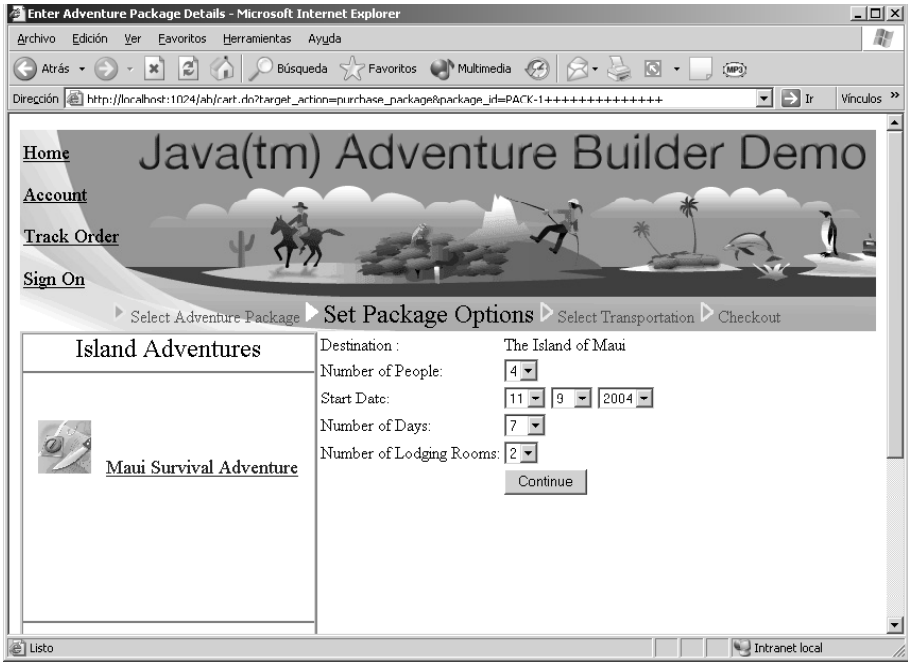
Composite View patroiak fitxategi hori kontsultatzen du, JSP orria aplikazioaren irteera-formatu estandarrera egokitzeko. Hala, erabiltzaileak, irteera grafikoaren formatu bera ikusiko du une oro.



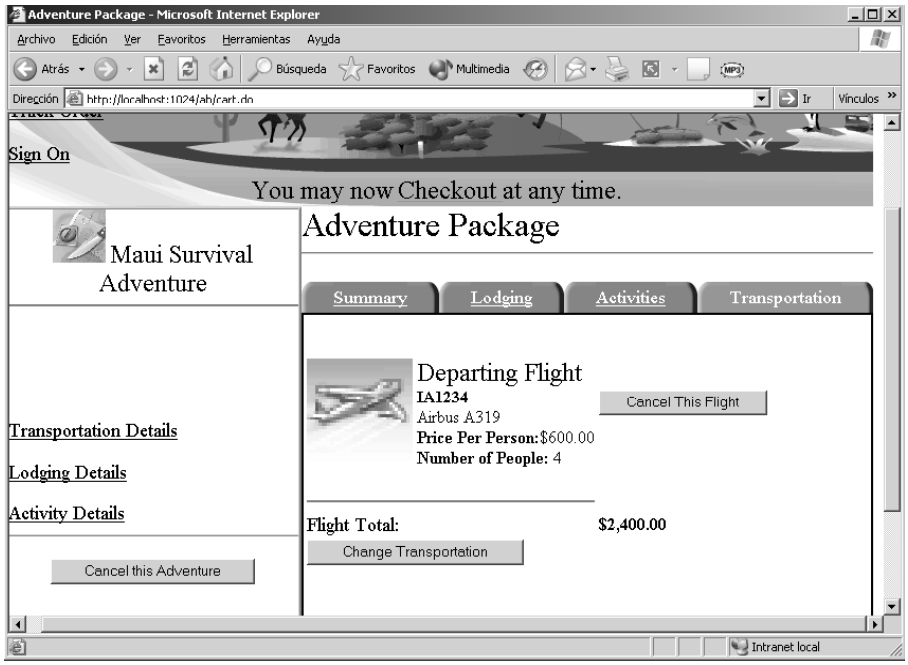
5.25. irudia. Bidaia-paketeen irteera grafikoa.



5.26. irudia. Bidaiaren jardueren informazio orokorra.



5.27. irudia. Bidaiaren erreserbaren ezaugarriak zehazteko interfaze grafikoa.



5.28. irudia. Bidaiaren hegaldiaren ezaugarri orokorrak.

5.7. LABURPENA

Hiru mailako arkitekturaren azterketak argi utzi du aplikazio informatikoen garapenaren konplexutasuna handitzen joan dela urtetik urtera. Testuinguru horretan eta Interneten eraginez, urteetan egonkorra izan den ekosistema, bat-batean, aldatu egin da. Albisteak eskuratzeko erak edota harreman berriak sortzeko inguruneak tokiz aldatu dira. Gaur egun, dena, birtualago eta konektatuago dago, albisteak ziztu bizian zabaltzen dira eta zu eta ni lau kliketara gaude.

Horri guztiari teknologien mosaiko heteroklitoa gehitzen badiogu, begibistakoa da arazo larri baten aurrean aurkitzen dela informatikaria. Nola eraiki aplikazio informatiko globalizatuak? Nola ipini harremanetan bidaia-agentziako aplikazioa hegazkin-konpainiako aplikazioarekin? Ikuspegi neutro eta agnostikoa aplikatuta, hau da, web zerbitzuak erabilita.

5.8. OINARRIZKO BIBLIOGRAFIA

McGovern, J.; Tyagi, S.; Stevens, M. eta Mathew, S. (2003): *Java Web Services Architecture*, Elsevier Science.

Monson-Haefel, R. (2003): *J2EE Web Services*, Addison Wesley Professional.

Myers, T. eta Nakhimovsky, A. (2003): *Google, Amazon, and Beyond: Creating and Consuming Web Services*, Apress.

Newcomer, E. eta Lomow, G. (2005): *Understanding SOA with Web Services*, Addison Wesley Professional.

Pulier, E. eta Taylor. H. (2005): *Understanding Enterprise SOA*, Manning.

5.9. WEB GUNE INTERESGARRIAK

Zer dago bertan	Web gunea
Sun erakundearen baliabideak Web zerbitzuak kudeatzeko	`http://java.sun.com/webservices/index.jsp`
Web zerbitzuen arkitektura	`http://www.sun.com/products/soa/index.jsp`
Web zerbitzuak eraikitzeko tutoretza	`http://java.sun.com/webservices/docs/2.0/tutorial/doc/`
Axis tresnaren ezaugarriak (I)	`http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=axis`
Axis tresnaren ezaugarriak (II)	`http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=webservicesAxis`

6. Ruby on Rails

6.1. SARRERA

Aurreko ataletan azaldu da web garapenaren hainbat arlo eta teknologia erabiltzen direla aplikazio informatiko xumeena ere diseinatu eta programatzeko. Informatikariarentzat egoera hori nahikoa etsigarria da behin eta berriz errepikatu behar baititu aurretik egindako urrats gehienak. Hala, Eredu-Bista-Kontrol erako arkitektura inplementatu nahi badu, tentu handiz diseinatu behar ditu aplikazioaren osagaiak geruza bakoitza bestearekin harremanetan jartzeko. Zer esanik ez informazioa biltegiratu eta atzitu nahi denean. Orduan, objektuak edota osagaiak erabilita diseinatu bada aplikazioa, objektu-erlazio mapaketa gauzatu beharko du informazioa biltegiratzeko.

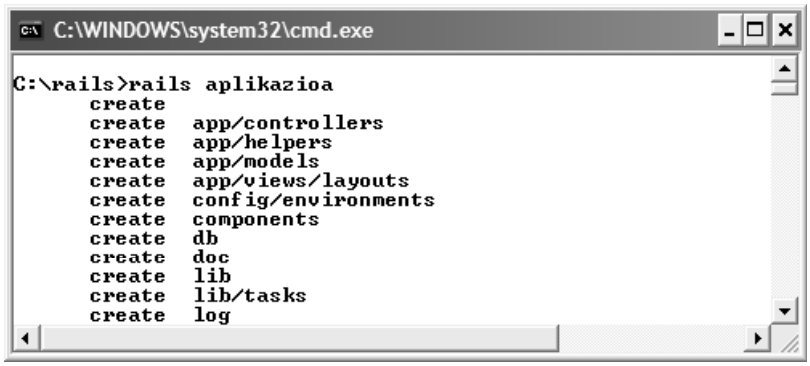
Jakina, aplikazioa egikaritzeko, zerbitzaria jarri beharko du martxan eta hainbat konfigurazio-fitxategi erabili beharko ditu aplikazioaren geruzak itsasteko.

Azpimarragarriena da aipatutako urrats eta ezaugarri horiek ez dutela aplikazioaren konplexutasuna aintzakotzat hartzen. Txakurraren txabola edota Gaudiren Familia Santuaren tenplua antzera kudeatzen eta eraikitzen dira. Sakrilegoa dirudiena arkitekturan normaltzat hartzen da informatikan.

Demagun, CRUD erako[74] web aplikazio informatikoa garatu nahi zaiola bezeroari eskaerak kudeatzeko. Jakin badakigu, aplikazio mota hori ez dela zaila inplementatzen, eragiketak lar ezagunak dira eta diseinuak ez du inolako zailtasunik.

Hori jakinda ere, aipatutako urratsak eta prozesuak gauzatu beharko ditu txintxo-txintxo aplikazioa martxan jartzeko. Eta horrela, behin eta berriz, nahiz eta lana errepikatu eta aplikazioaren tipologia berdintsua izan. David Heinemeier Hansson jauna antzeko arazoak ari zen aurkitzen 2004an aplikazioak inplementatu behar zituenean Ruby programazio-lengoaia erabilita. Zergatik ez automatizatu lan mekaniko eta errepikakor hori tresna informatiko aurreratu bat inplementatuta?

74. http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
http://wiki.rubyonrails.org/rails/pages/CRUD

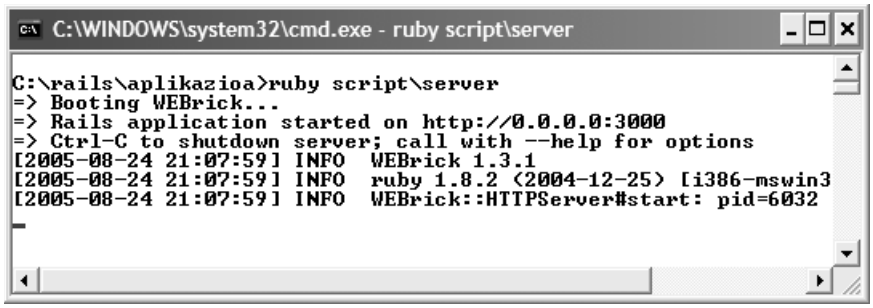


6.1. irudia. Aplikazioa eraikitzen `rails` tresna aurreratua erabilita.

Pentsa eta egin, 2004ko uztailean Ruby on Rails tresna informatiko aurreratua jaio zen ideia sinple eta eraginkorra inplementatuz: ordenagailuak egin dezala lan mekanikoa, informatikariaren eskuetan utziz lan sortzailea.

Hala, aplikazioak behar duen azpiegitura informatiko osoa eraikiko dio Ruby on Rails-ek informatikariari bizi-bizi eta txintik esan gabe. Eraikitze-prozesuak hainbat azpidirektorio (**6.1. irudia**) eta fitxategi sortuko ditu gero programatzen joango diren baliabideak ondo antolatzeko. **6.1. irudia**n ikusten den legez, Rails tresnak hainbat azpidirektorio eraikitzen ditu Eredu-Bista-Kontrol arkitektura erantsiz sortzen ari den aplikazioari. Kontu egin ezen aplikazioa erabat funtzionala dela, hau da, ez duela inongo konfigurazio-fitxategirik edota mapaketarik behar egikaritzeko.

Nahikoa da zerbitzaria martxan jartzea aplikazioa operatibo bihurtzeko. Zorionez, Rails tresnak, bere zerbitzari propioa dakar eta nahikoa da script bati deitzea zerbitzaria (`WEBrick`) martxan jartzeko, **6.2. irudia**n ikusten den legez.



6.2. irudia. `WEBrick` zerbitzariaren hasieraketak.

`aplikazioa` azpidirektorioa (**6.3. irudia**) aztertzen badu informatikariak, berehala konturatuko da aplikazioaren osagai bakoitza, toki zehatz eta ezagunean uzten duela tresnak, Ruby motako fitxategiez hornituz azpidirektorioak. Hala,

kontrolatzaile berri bat programatu behar duenean, badaki nora joan behar duen aurrekoak (`controller`) ikustera eta aztertzera. Gauza bera gertatzen zaio bista berri bat edota osagai bat gehitu nahi dionean ereduari. Lanak errazteko, script-ak eskaintzen dizkio tresnak programatzaileari, aipatutako osagaiak gehitzeko aplikazioari.

Demagun aplikazio eraiki berriari `Artikulu` osagaia gehitu nahi zaiola eskaeraren artikuluak kudeatzeko. Nahikoa da honako hau adieraztea Rails tresnari eraiki behar diren fitxategiak (**6.4 irudia**) sortzeko:

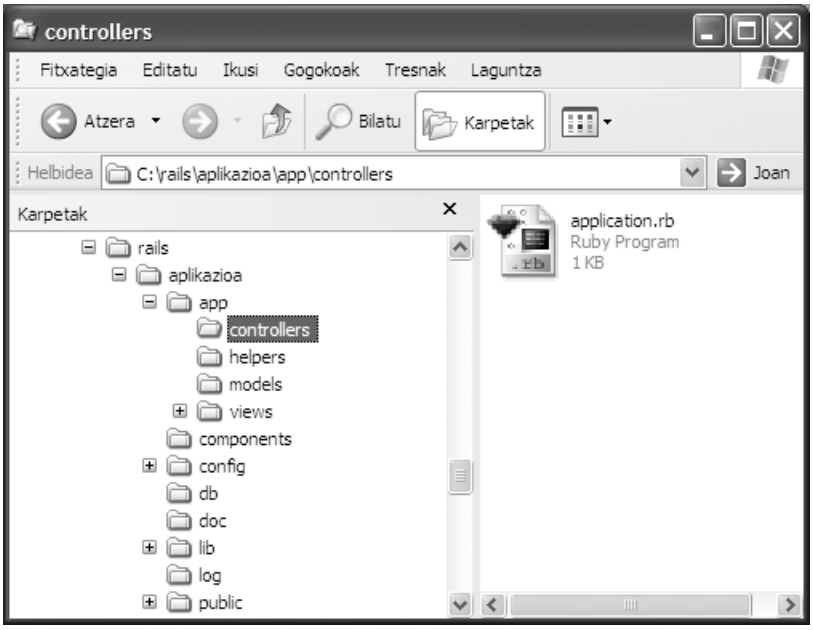
```
> ruby script\generate model Artikulu
```

Inplementatzen ari garen arkitekturan, kontrolatzaileek prozesatzen dituzte erabiltzailearen web galderak. Hori dela-eta, ereduan gehitu den osagaiari kontrolatzailea (**6.5. irudia**) gehituko zaio honako script hau erabilita:

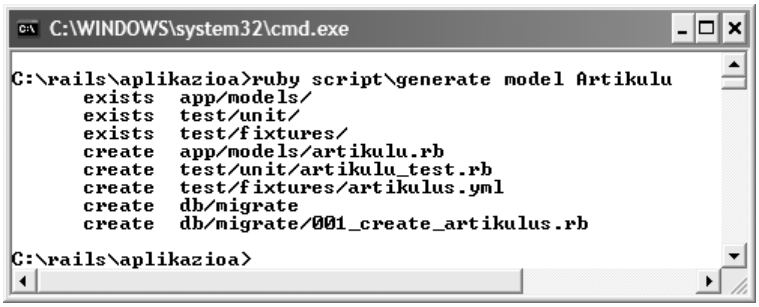
```
> ruby script\generate controller Artikulu
```

Lehen bezala, orain ere, Ruby motako hainbat fitxategi eraikitzen ditu Rails tresnak, `Artikulu` osagaiaren web galderak kontrolatzeko.

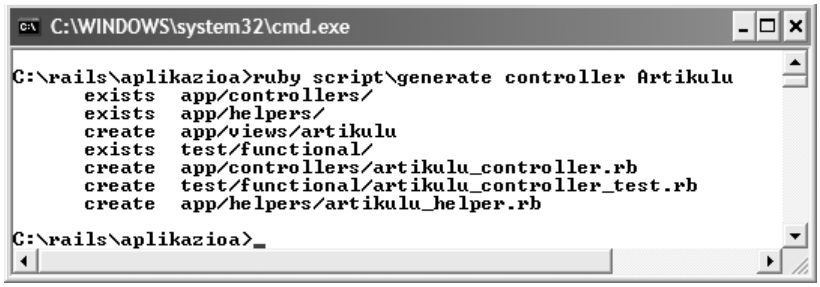
Bi script horiek argi adierazten dute ezen tresnaren lan egiteko modua Eredu-Bista-Kontrol arkitekturarekin estu-estu lotzen dela eta aplikazioaren programatzailea hiru arlo horietan mugituko dela, osagaiak gehitzeko eta eguneratzeko.



6.3. irudia. Azpidirektorioen hierarkia orokorra.



6.4. irudia. `Artikulu` **osagaiaren baliabideen sorkuntza.**



6.5. irudia. `Artikulu` **osagaiaren kontrolatzailearen eraikuntza.**

6.2. OSAGAIEN IRAUNKORTASUNA RUBY ON RAILS ERABILITA

Rails ingurune informatiko aurreratuak egungo teknologia gehienek legez, datubase erlazionaletan biltegiratzen ditu aplikazioaren osagaiak. Hala eta guztiz, teknologia horien eta Rails ingurunearen artean, egon badago aldea, iraunkortasuna tratatzeko orduan. Teknologia gehienetan objektuen eta entitateen arteko mapaketa konfigurazio-fitxategiak erabilita gauzatzen da; Rails ingurunean, aldiz, mapaketa, automatikoa eta gardena da programatzailearentzat. Active Record patroia[75] erabilita, aplikazioak prozesatzen dituen domeinuko klaseak automatikoki erlazionatzen dira datu-baseko entitateekin. Hala, domeinuko klase bat datu-baseko taula batekin erlazionatuko da eta taularen zutabe bakoitza domeinuko klasearen eremu bihurtuko da automatikoki. Hori guztia gauzatzeko, nahikoa da adieraztea kontrolatzailean, ereduaren zein osagairekin lan egin nahi den, kateaketa egiteko.

Lotura, `scaffold` hitz erreserbatua erabilita gorpuzten da. Gure adibidean, artikuluaren kontrolatzaileari ohartarazi nahi bazaio ereduko `artikulu` osagaiarekin lan egin nahi dela, honako hau idatzi (**6.6. irudia**) beharko du eraiki berri den `artikulu_controller.rb` fitxategian:

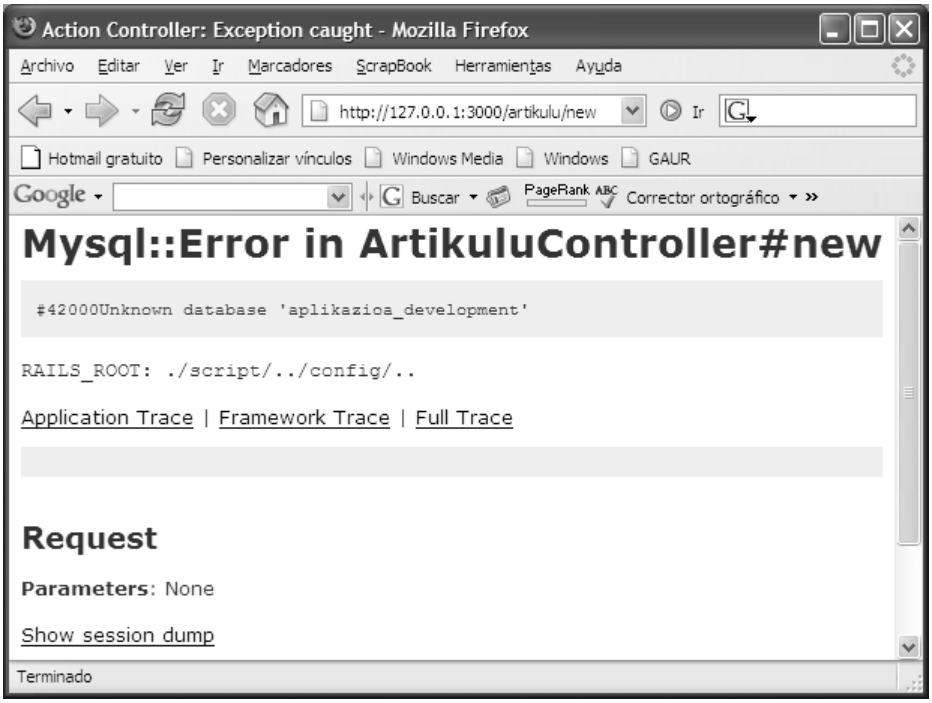
```
scaffold: artikulu
```

75. `http://en.wikipedia.org/wiki/Active_record_pattern`
`http://www.martinfowler.com/eaaCatalog/activeRecord.html`

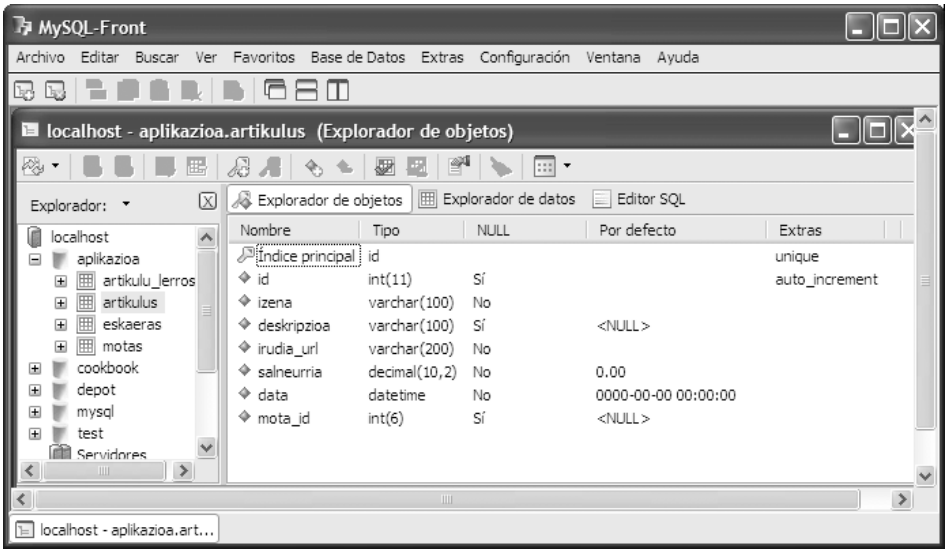


6.6. irudia. `scaffold` baliabidea txertatzen `Artikulu` osagaiaren kontrolatzailean.

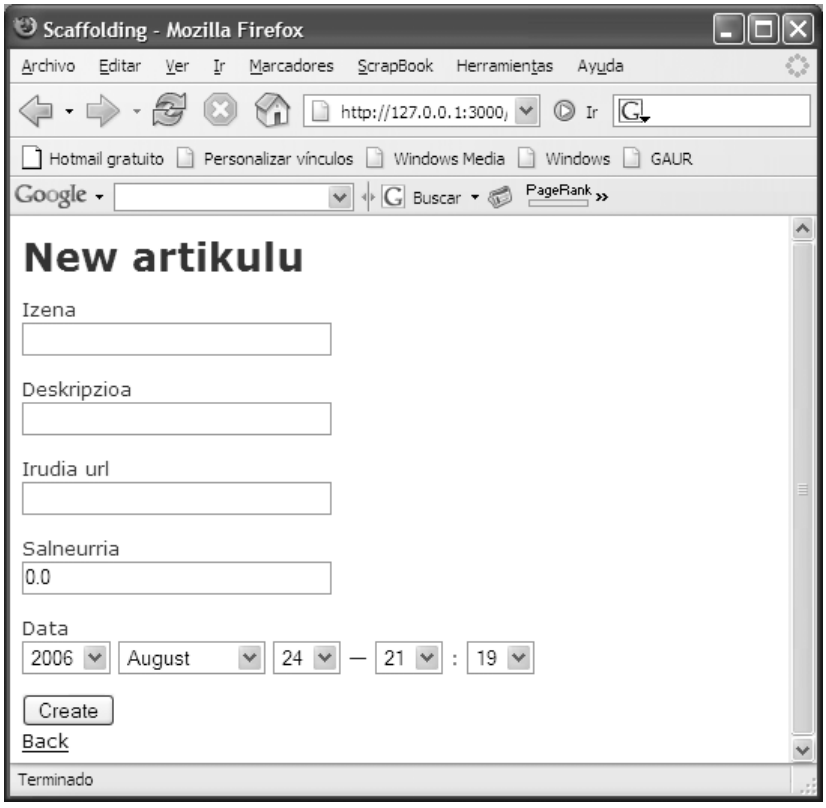
Fitxategia gorde eta aplikazioa egikaritzen bada, **6.7. irudia**n agertzen dena azalduko da nabigatzailean, hau da, errorea kontrolatzailean. Inguruneak ez du datu-basea aurkitzen mapaketa gauzatzeko, datu-basearen aldean ez baita inongo eragiketarik egin. Aztertzen ari garen CRUDerako web aplikazioaren iraunkortasuna kudeatzeko MySQL motako datu-basea eraikiko da. Eragiketa hori `MySQL-Front` baliabidea erabilita gauzatu da, **6.8. irudia**n ikusten den legez.



6.7. irudia. Errore-mezuaren interfaze grafikoa.



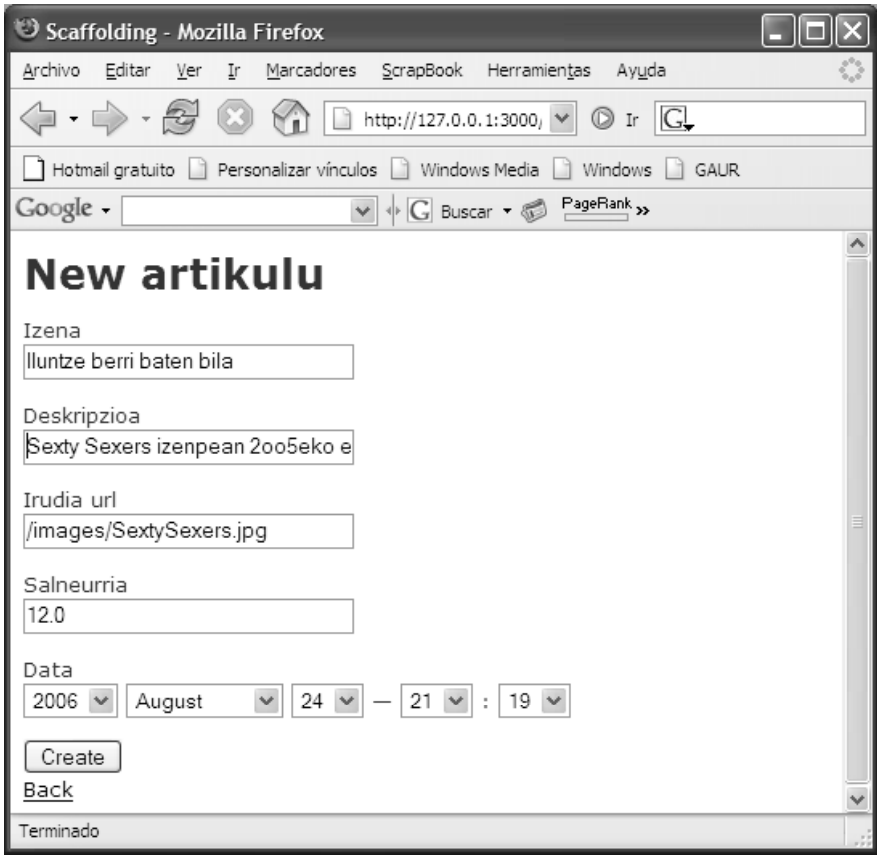
6.8. irudia. MySQL-Front baliabidearen interfaze grafikoa.



6.9. irudia. Aplikazioaren lehen irteera grafikoa.

Eragiketak eraginik izan ote duen ikusteko berriz egikaritzen da aplikazioa eta oraingo honetan, aleluia! hor dago martxan gure CRUD erako web aplikazioa (**6.9. irudia**), lerro bakarra (`scaffold:artikulu`) programatuta!

6.9. irudian ikusten den legez, aplikazioa zain gelditzen da artikuluak onartzeko nabigatzailetik. Pare bat sartzen dugu (**6.10. irudia**) eta aplikazioaren portaera aztertzen dugu. **6.11.–6.14. irudie**k islatzen dute nola gauzatzen diren eguneraketak nabigatzailean eta datu-basean.

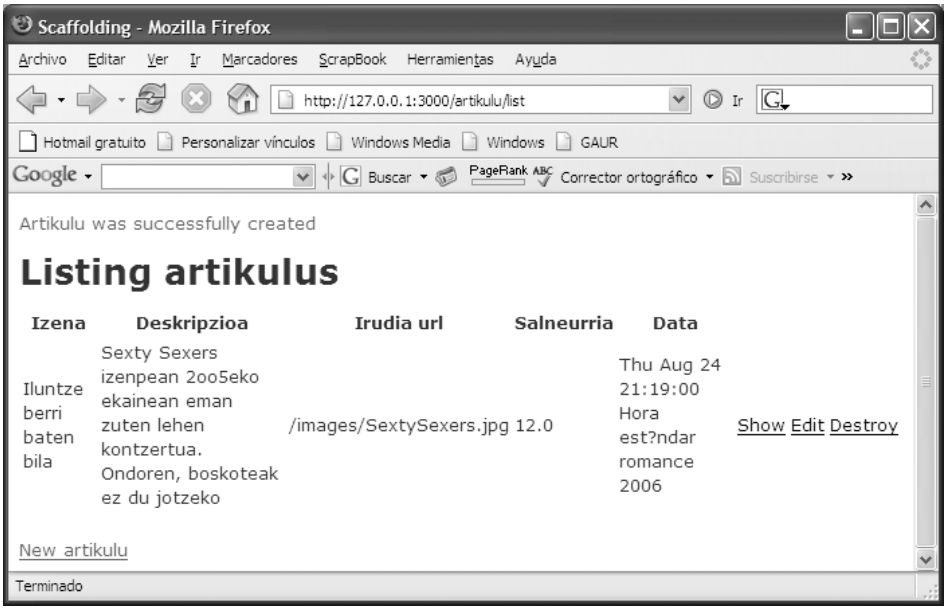


6.10. irudia. Artikuluak gehitzen datu-basera.

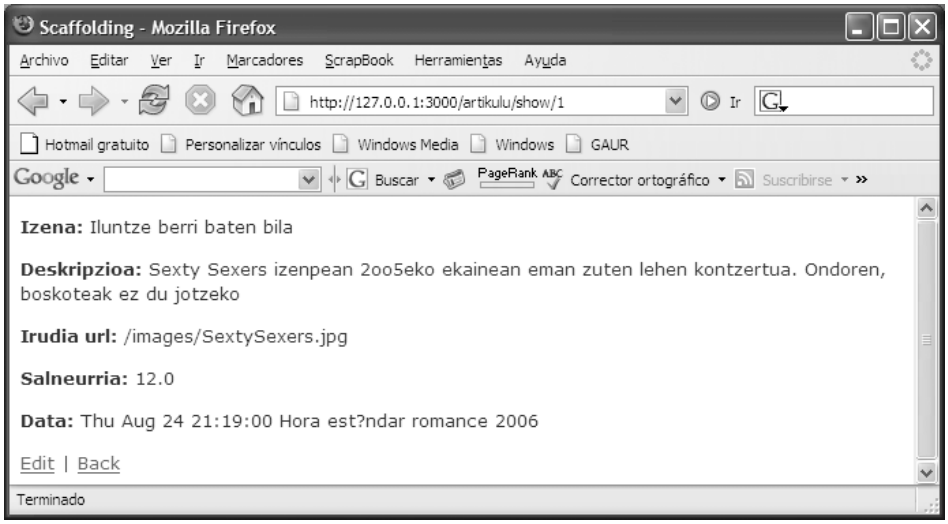
Jolastu dezagun zertxobait inplementazioarekin. Ez dugunez Ruby[76] programazio-lengoaia ezagutzen, ez gara ur handitan sartuko. Oraingoz, arkitekturaren eredu-arloa bere horretan utziko da. Lehenengo adibidean, artikuluaren kontrolatzailea eta aplikazioaren irteera grafikoa eguneratuko dira, funtzionalitate berri bat programatuta.

76. `http://en.wikipedia.org/wiki/Ruby_programming_language`
`http://www.ruby-lang.org/`

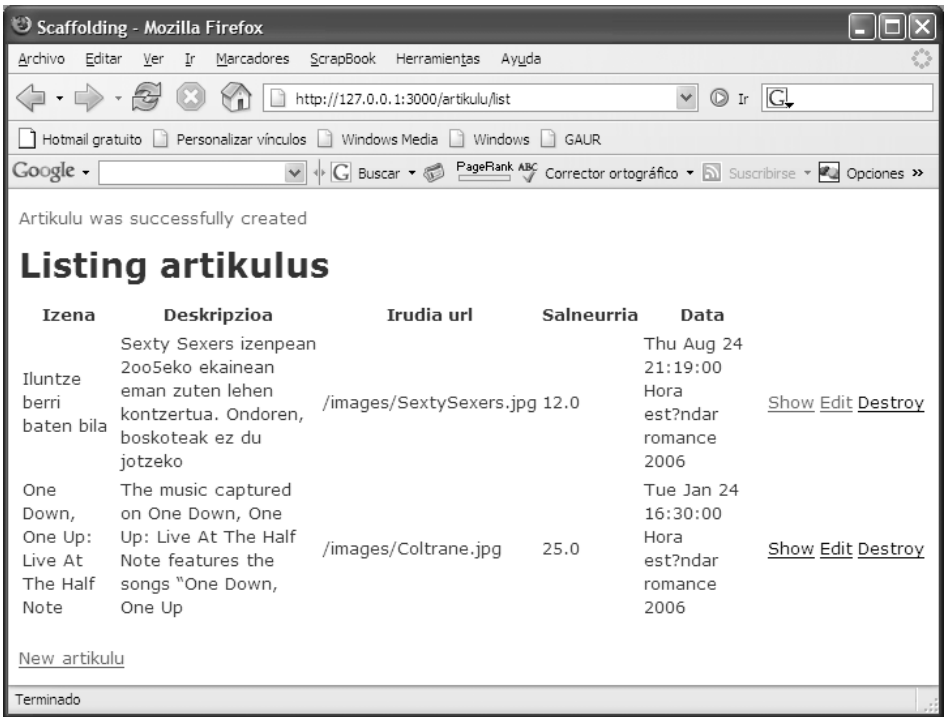
Aurreko irudietan ikusi da aplikazioak baduela `list` erako funtzionalitatea (**6.11. irudia** eta **6.13. irudia**) lehenetsia eta arazorik gabe egikaritzen dela baina, askotan, funtzionalitate hori egokitu nahi izaten da aplikazioaren beharretara beste itxura bat emanda irteera grafikoari.



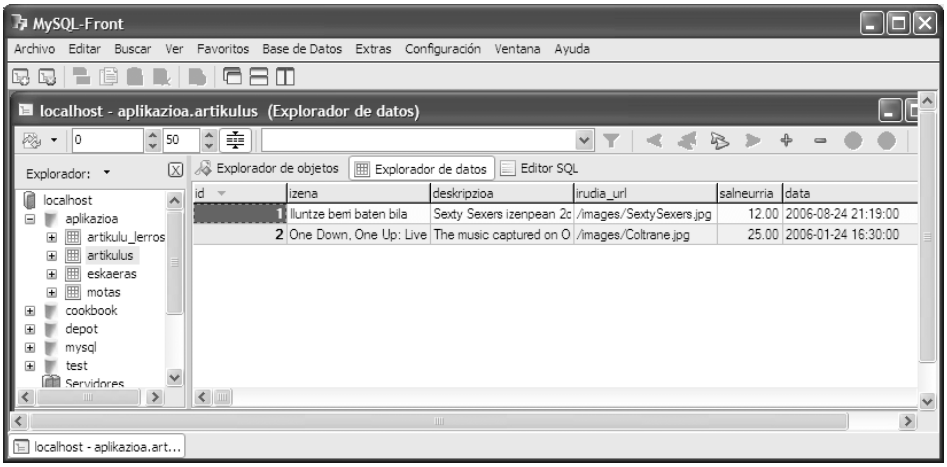
6.11. irudia. Artikuluen hasierako zerrenda (`list`).



6.12. irudia. Artikulu baten informazio zehatza (`show`).



6.13. irudia. Artikuluen zerrenda (`list`) eguneratua.



6.14. irudia. Datu-basean biltegiratutako artikuluak.

Hori egiteko artikuluaren (**6.15. irudia**) kontrolatzailean (`artikulu_controller.rb`) `list` metodo berria txertatzen da honako hau jarrita:

```
def list
   @artikulus = Artikulu.find_all
end
```

Metodoaren lerro bakarrak `Artikulu` klasearen bilduma eskuratzen du datu-basetik eta `@artikulus` aldagaiari esleitzen dio aipatutako bilduma.

Eragiketa horrekin, datu-basetik lortu behar zuen informazioa eskuratzen du kontrolatzaileak eta prest uzten du irteera grafikotik ateratzeko.



6.15. irudia. `list` metodoa txertatzen `Artikulu` osagaiaren kontrolatzailean.

Bigarren eragiketa hori gauzatzeko, artikuluaren kontrolatzailea eraiki den unean, `views` azpidirektorioan sortu den `artikulu` azpidirektorioan `rhtml` motako fitxategia eraikitzen da `list` izenarekin eta honako eduki hau idatzita:

```
<html>
<head>
<title>Artikulu guztiak</title>
</head>
<body>

<h1>OnLine APLIKAZIOA- Artikulu guztiak</h1>
<table border="1">
 <tr>
  <td width="25%"><p align="center"><i><b>Irudia</b></i></td>
  <td width="40%"><p align="center"><i><b>Artikulua</b></i></td>
  <td width="20%"><p align="center"><i><b>Data</b></i></td>
 </tr>

 <% @artikulus.each do |artikulu| %>
  <tr>
   <td>
  <img width="140" height="140" src ="<%= artikulu.irudia_url %>" />
   </td>
   <td><%= link_to artikulu.izena, :action => "show", :id =>
artikulu.id %></td>
   <td><%= artikulu.data %></td>
  </tr>
 <% end %>
</table>
<p><%= link_to "Eraiki artikulu berria", :action => "new" %></p>
</body>
</html>
```

JSP eta ASP teknologiak ezagutzen dituenak ez du inongo arazorik izango fitxategi horren edukia ulertzeko. Aipatutako bi teknologietan bezala, hemen ere, HTML txantiloia erabiltzen da irteera grafikoa egituratzeko. Irteera grafikoaren egokitzapenak Ruby motako iturburu-kodea txertatuta gauzatzen dira eta HTML estandarra erabiltzen du fitxategiak, irteera grafikoaren itxura orokorra egituratzeko. Zalantzarik gabe iturburu-kodearen zatirik garrantzitsuena honako hau da:

```
  <% @artikulus.each do |artikulu| %>
  <tr>
   <td>
    <img width="140" height="140" src ="<%= artikulu.irudia_url %>" />
   </td>
   <td><%= link_to artikulu.izena, :action => "show", :id =>
artikulu.id %></td>
   <td><%= artikulu.data %></td>
  </tr>
 <% end %>
</table>
<p><%= link_to "Eraiki artikulu berria", :action => "new" %></p>
```

Ruby motako iturburu-kodearekin kontrolatzaileak eskuratutako artikuluak prozesatu egiten dira eta nabigatzailearen taularen gelaxketan (**6.16. irudia**) txertatzen dira beste itxura bat emanez irteera grafikoari.



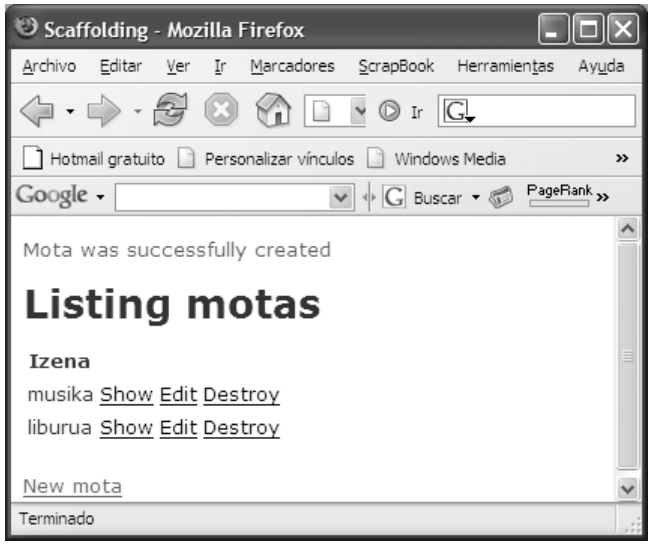
6.16. irudia. Aplikazioaren irteera grafiko berria.

Orain arte aztertutakoarekin, lehenengo adibidean azaldu da nola jartzen diren harremanetan kontrolatzaileak eta bistak, bigarren eta azken adibidean, aplikazioaren ereduen ezaugarriak aldatuko dira, datu-basearen erlazioak adierazteko aplikazioaren iturburu-kodean. Horretarako, motetan sailkatuko dira datu-basearen artikuluak.

Lehendabizi, artikuluak motetan sailkatu baino lehen, `Mota` osagaiaren kontrolatzailea eta eredua eraikitzen dira:

```
> ruby script\generate controller Mota
> ruby script\generate model Mota
```

Ondoren, aplikazioa egikaritzen da eta bi mota berri sortzen dira. Prozesua ondo joan bada, **6.17. irudia**n agertzen dena azalduko da nabigatzailean, `new` ekintzak egikarituta.

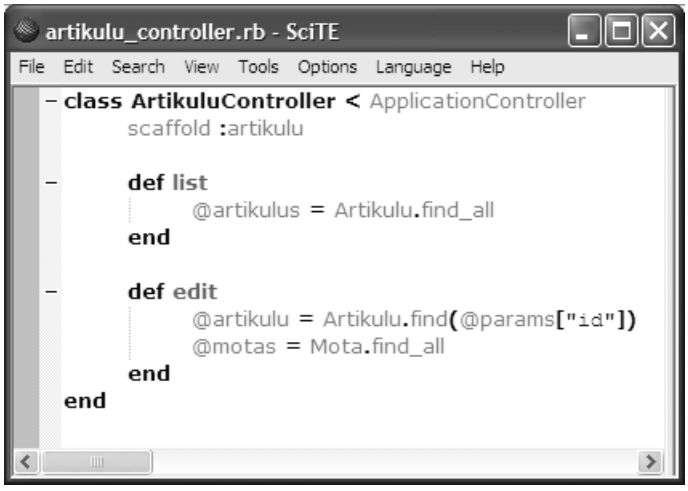


6.17. irudia. Artikuluen motak txertatzen aplikazioan.

Azpiegitura finkatuta uzteko, artikuluen moten eta artikuluen arteko erlazioak adierazi behar ditugu inplementazioaren `models` azpidirektoriora joanda. Azpidirektorioan dagoen `artikulu.rb` fitxategia editatzen da eta honako hau idazten da: `belongs_to:mota`. Amaitzeko, gorde egiten da aldaketa.

Antzeko eragiketa gauzatzen da `mota.rb` fitxategiarekin baina oraingoan, honako hau idazten da: `has_many:artikulu`. Kontu egin ezen eragiketa horiekin motaren eta artikuluaren artean dagoen erlazioa adierazten dela aplikazioaren eredu-arloan.

Azpiegitura prestatuta, txerta dezagun domeinuaren logika berria artikuluaren kontrolatzailean (**6.18. irudia**). Horretarako, lehen bezala, artikuluaren kontrolatzaileari (`artikulu_controller.rb`) `edit` metodo berria erantsiko zaio, domeinuaren kudeaketa-arau berria programatzeko. Oraingo honetan, `@artikulu` eta `@motas` aldagaiak erabiliko dira, HTML txantiloia elikatzeko.



6.18. irudia. Domeinuaren logika berria txertatzen `Artikulu` osagaiaren kontrolatzailean.

Gelditzen den azken urratsa `edit.rhtml` fitxategia sortzea da:

```
<html>
 <head>
  <title>Editatu Artikulua</title>
 </head>
 <body>
 <h1>Editatu Artikulua</h1>

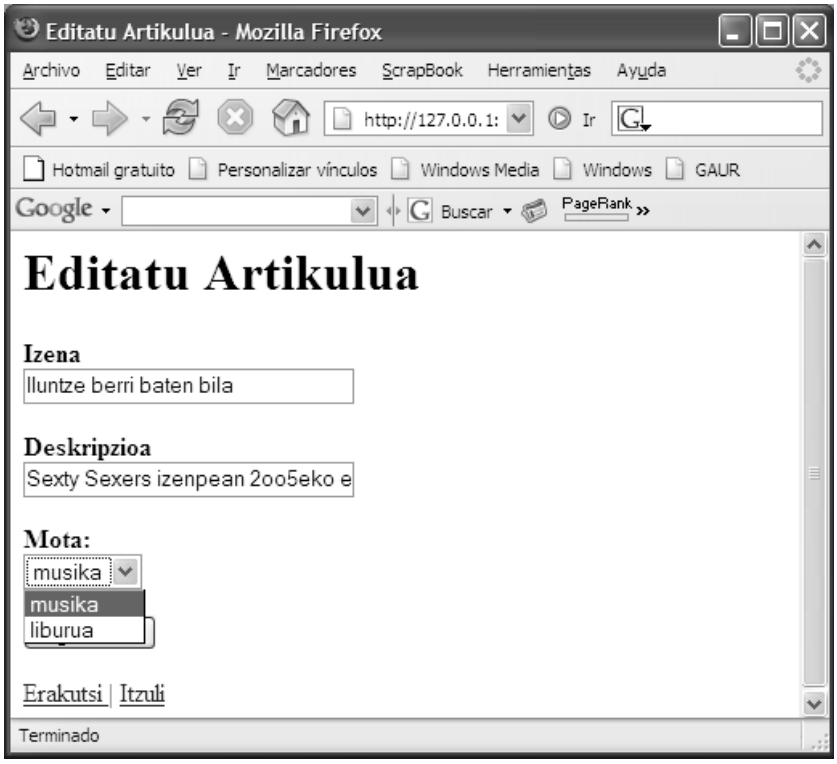
 <form action="../update/<%= @artikulu.id %>" method="POST"">
  <input id="artikulu_id" name="artikulu[ id]" size="30"
         type="hidden" value="<%= @artikulu.id %>" />
  <p><b>Izena</b><br>
  <input id="artikulu_izena" name="artikulu[ izena]" size="30"
         type="text" value="<%= @artikulu.izena %>" />
  </p>
  <p><b>Deskripzioa</b><br>
  <input id="artikulu_deskripzioa" name="artikulu[ deskripzioa]"
         size="30" type="text"
         value="<%= @artikulu.deskripzioa %>" />
  </p>
  <p><b>Mota:</b><br>
```

```
  <select name="artikulu[ mota_id]">
   <% @motas.each do |mota| %>
       <option value="<%= mota.id %>"
         <%= ' selected' if mota.id == @artikulu.mota_id %>>
         <%= mota.izena %>
       </option>
   <% end %>
</select></p>

 <input type="submit" value="Eguneratu" />
 </form>

 <a href="/artikulu/show/<%= @artikulu.id %>">
   Erakutsi
 </a> |
  <a href="/artikulu/list">
  Itzuli
 </a>

 </body>
</html>
```



6.19. irudia. `Artikulu` osagaiaren interfaze grafiko berria.

Txantiloiak goian aipatutako aldagaien informazioa eskuratzen du eta, nabigatzailea erabilita, erabiltzailearen esku (**6.19. irudia**) jartzen du nahi dituen aldaketak gauzatzeko. Kontu egin ezen aukeran dituen mota guztiak goitik beherako menutxoan jartzen dizkiola aplikazioak erabiltzaileari.

6.3. LABURPENA

Bertrand Meyer-en *Object-Oriented Software Construction* liburuan[77] diseinupatroi polita aurkezten da, aplikazioaren menuen arteko trantsizioak egituratzeko balioko duena. Planteatzen den ebazpena generikoa da eta edozein testuingurutan aplika daiteke. Hala ere, informatikari gehienok, gure *gurpiltxoa* eraikitzen dihardugu egunero gure lanpostuan.

Ruby on Rails tresna aurreratuak antzeko ikuspegiarekin prozesatzen ditu Eredu-Bista-Kontrol arkitekturaren erregulartasunak. Gainera, aplikazioaren osagaien iraunkortasuna, Active Record patroia inplementatuta kudeatzen du. Horretarako, aplikazioaren domeinuko klaseak automatikoki erlazionatzen ditu datu-baseko entitateekin.

6.4. OINARRIZKO BIBLIOGRAFIA

Black, D. A. (2006): *Ruby for Rails. Ruby techniques for Rails developers*, Manning.

Fowler, C. (2006): *Rails Recipes*, Pragmatic Bookshelf.

Holzner, D. (2006): *Beginning Ruby on Rails*, Wrox.

Koenig, D.; Glover, A.; King, P.; Laforge, G. eta Skeet, J. (2007): *Groovy in Action*, Manning.

Rocher, G. (2006): *The Definitive Guide to Grails*, Apress.

Tate, B. eta Hibbs, C. (2006): *Ruby on Rails: Up and Running*, O´Reilly.

Thomas, D. eta Heinemeier, D. (2006): *Agile Web Development with Rails*, Pragmatic Bookshelf.

77. http://en.wikipedia.org/wiki/Object-Oriented_Software_Construction

6.5. WEB GUNE INTERESGARRIAK

Zer dago bertan	Web gunea
Ruby programazio-lengoaia	http://www.ruby-lang.org/
Ruby on Rails teknologia (I)	http://www.rubyonrails.org/
Ruby on Rails teknologia (II)	http://www.rubyonrails.org.es/
Tutoretza txikia (I)	http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html
Tutoretza txikia (II)	http://www.onlamp.com/pub/a/onlamp/2005/03/03/rails.html
JRuby-ren ezaugarriak	http://jruby.codehaus.org/
Groovy programazio-lengoaia	http://groovy.codehaus.org/
Grails tresnaren informazioa	http://grails.codehaus.org/
Sarrera txikia Grails tresnara	http://grails.org/Quick+Start

7. Eranskinak: ariketak eta ordenagailuarekin egiteko laborategiak

7.1. JAVA: ARIKETA PROZEDURALAK

Programa bat sortu nahi dugu Java lengoaia erabilita honako hau gauzatzeko. DiruGose bankuak programa bat garatu nahi du bere langileek ikus dezaten nola hazten den kapital bat urteetan zehar. Kapitala eta aplikatu beharreko interesa eskatuko ditu programak eta, emaitza gisa, histograma bat aterako du zeinak kapitala urtetik urtera nola aldatzen den islatuko duen.

```
Inbertsio Plana
===================
KAPITALA:  1000.0 RATIOA: 12.5

Urteak
5     |***************** 1750
6     |******************* 1900
7     |********************* 2050
8     |*********************** 2200
9     |************************* 2350
10    |*************************** 2500
11    |***************************** 2650
12    |******************************* 2800
13    |********************************* 2950
14    |*********************************** 3100
15    |************************************* 3250
      ================================================
      +         +         +         +         +         +
      0      1000      2000      3000      4000      5000
                             Balioak
```

oOo

Ariketa honetan ausazko zenbaki (0-10000) bat sortuko du ordenagailuak eta erabiltzaileak igarri egin behar du. Gauzak errazteko, erabiltzaileak zenbakia sartu eta igartzen ez badu, laguntza txiki bat eskainiko du ordenagailuak, honako mezu hau idatzita:

- **Asmatu behar den zenbakia tarte honetan dago: ?????? - ??????**

Gogoratu ezen zenbakia irakurtzeko nahikoa dela honako agindu hau idaztea beheko iturburu-kodean (ordenagailuak **`String`** motako balioa itzultzen du):

```
adibidea.sarrera.readLine();
```

Hona hemen idatzi behar duzun iturburu-kodearen hasiera:

```
import java.lang.Math;
import java.io.*;

class Zenbaki {
         int maxZen,    // Tartearen maximoa.
               minZen,    // Tartearen minimoa.
               saialdiak, // Saialdi kopurua.
               zenbaki,   // Igarri behar duen zenbakia.
               n;         // Erabiltzaileak sartutako zenbakia.
       boolean amaiera;  // True zenbakia igartzen denean.
       BufferedReader sarrera = new BufferedReader(new
InputStreamReader(System.in));
       String buffer;

    public static void main(String args[]) throws IOException {
                Zenbaki adibidea= new Zenbaki();
          adibidea.maxZen = 10000;
          adibidea.minZen = 0;
          adibidea.saialdiak = 1;
          adibidea.zenbaki = (int)Math.round(Math.random() *
adibidea.maxZen);
          adibidea.amaiera = false;
          while(!adibidea.amaiera) {
               System.out.println("\nIgarri behar duzun zenbakia
tarte honetan dago: " + adibidea.minZen + " - " + adibidea.maxZen);
               // Zenbakia irakurri
               boolean zenbakiaDa=false;
               while (!zenbakiaDa) {
                    adibidea.buffer = adibidea.sarrera.readLine();
                    try {
               adibidea.n =
Integer.valueOf(adibidea.buffer).intValue();
                              zenbakiaDa=true;
                    }
                    catch (NumberFormatException e) {
      System.out.println("\n    Zenbaki bat !!! zenbakia izan behar
du !!!!");
               }
           }
```

oOo

Azkar-azkar enpresak kamioi berriak erosi ditu, eta, ondorioz, egin behar dituen bidaien iraupena laburtu egin da. Lortu den hobekuntza % 15ekoa izan da batez beste.

1. Enpresak programa bat sortu nahi du lehengo eta oraingo orduak eta bidaiaren iraupena ikusteko.

   Jarraituko dugun sekuentzia honako hau izango da:

   - Irakurri irteera- eta iriste-orduak.
   - Bidaiaren iraupena kalkulatu.
   - Bidaiaren iraupena % 15 murriztu.
   - Kalkulatu iriste-ordu berria.
   - Atera emaitza guztiak:

```
Irteera-ordua: 900
Lehengo iriste-ordua: 1015
Bidaiaren lehengo iraupena: 75 minutu
Bidaiaren oraingo iraupena: 64 minutu
Oraingo iriste-ordua: 1004
```

2. Hedatu programa funtzionalitate berri hau itsatsita: bidaia herrialde urrunetara denean, bertako ordua ere azaldu beharko du dei bat egin edo fax bat bidali nahi badugu eta ordu egokia den ala ez jakiteko. Horretarako ordutegitaula hau aplikatuko dugu:

Tokia	Ordua	Minutua
Belgrado, Sofia	1	20
Mosku	3	20
Bangkok, Hanoi	7	30
Singapur	8	10

oOo

Jarraian agertzen den listatua aztertuta, bete honako taula hau:

Iturburu-kodea	Arazorik ez	Arazoak	Zergatik? (Arazorik ez / Arazoak)
`idazkari1 = pertsona` `idazkari1.kalkulatuSoldata()`			
`idazkari2 = (Idazkaria)pertsona` `idazkari2.kalkulatuSoldata(88)`			
`pertsona = langile1` `pertsona.kalkulatuSoldata(35)`			
`pertsona = langile2` `pertsona.kalkulatuSoldata()`			
`pertsona = langile2` `pertsona.ikuskatu()`			
`pInterfazea = idazkari1` `pInterfazea.kalkulatuPrima(88)`			
`pInterfazea1 = idazNagusia` `pInterfazea1.kalkulatuPrima(88)`			

```
interface PrimaInterfazea {
  void kalkulatuPrima(int orduak);
  void idatziPrima();
  }

interface PrimaInterfazeaId extends PrimaInterfazea  {
  void kalkulatuPrima(int orduak);
  void idatziPrima();
  }

class Pertsonala {

       private String nan;
       private String Izena;
       private String Helbidea;
       private String Telefonoa;
       public int antzinatasuna;
       public int gehigarria;
       public int oinarrizkoa;
       private int oinarrizkoSoldata;

.............
public void kalkulatuSoldata( ) {

       oinarrizkoSoldata= antzinatasuna*gehigarria + oinarrizkoa;
                                    }
......}

class Langilea extends Pertsonala {
  int soldata;
  public final  int gehigarriaL = 65;
```

```
.......
public void kalkulatuSoldata( int orduak ) {

    soldata = orduak * oinarrizkoa + igoeraL*antzinatasuna+ gehigarriaL;
.................}
public void ikuskatu(){ ......}

....}

class Idazkaria extends Pertsonala implements PrimaInterfazea  {
  int soldata;
  public final  int gehigarriaI = 85;
.......
public void kalkulatuPrima( int orduak ){
       System.out.println("kalkulatuPrima -> Idazkaria");
}

public void idatziPrima(){
       System.out.println("idatziPrima -> Idazkaria");
}


public void kalkulatuSoldata( int orduak ) {

    soldata = orduak * oinarrizkoa + igoeraI*antzinatasuna+ gehigarriaI;
.................}
....}

class INagusia extends Idazkaria {

int osagarriak;
.....
public void kalkulatuPrima( int orduak ){
       System.out.println("kalkulatuPrima -> INagusia");
}

public void idatziPrima(){
       System.out.println("idatziPrima -> INagusia");
}

public void kalkulatuSoldata( int orduak ) {

    soldata = orduak * oinarrizkoa + igoeraI*antzinatasuna+ gehigarriaI+
osagarriak*antzinatasuna;
................}
....}

public class aplikazioa {
public static void main( String args[] ) {
```

```
        Pertsonala pertsona = new Pertsonala(...);
        Idazkaria idazkari1 = new Idazkaria(...);
        Idazkaria idazkari2 = new Idazkaria();
        Langilea langile1 = new Langilea();
        Langilea langile2 = new Langilea();
        INagusia idazNagusia = new INagusia(...);
        PrimaInterfazea pInterfazea, pInterfazea1;
        ......


......}
```

oOo

`Irudi` eta `Marrazki` klaseak emanda:

```
public class Irudi { …
      void static marraztu (Marrazgarri b) { …}  … }

public class Marrazki extends java.awt.Frame { …}
```

Eta: `Marrazki m;` definituta badugu:

Aukera ezazu erantzun egokia:

a. `Irudi.marraztu(m)` egikari daiteke `marraztu` static metodoa baita.

b. `Irudi.marraztu(m)` egikari daiteke `Marrazgarri` klasea honela definitzen badugu `Marrazgarri extends java.awt.Frame { …}`

c. `Irudi.marraztu(m)` egikari daiteke `Marrazgarri` klasea honela definitzen badugu `public class Marrazgarri extends Marrazki { …}` eta aurretik, `m = new Marrazki();` egikaritu bada.

d. `Irudi.marraztu(m)` egikari daiteke `Marrazgarri` klasea honela definitzen badugu `public class Marrazgarri extends Marrazki { …}` eta aurretik, `m = new Marrazgarri();` egikaritu bada.

e. Aurreko erantzun guztiak faltsuak dira.

oOo

Enpresa batek bezeroaren eskaerak tratatu nahi ditu eta horretarako klase bat (`BezeroEskaerak`) diseinatu nahi du inplementazio egokiarekin. Klaseak honako diseinu hau izango du:

```
class Artikulu {
       String izena;
       float salneurria;
             }
class Item {
       Artikulu artikulua;
       int kopurua;
```

```
            }
class Eskaera {
       Bezero bezeroa;
       Item[ ] elementuak;
             }
class Bezero {
       String izena, helbide, telefono;
       float kreditua; }

class BezeroEskaerak {
       private Eskaera[ ] eskaerak = new Eskaera[50];
       static final float Kreditu = 1000;

       (a)public Eskaera[ ] eskaerakEman(String bezero);
       (a)public Eskaera[ ] eskaerakEman(Bezero bezero);

       (b)public boolean sartuEskaera(Eskaera eskaera);
       (c)public Eskaera[ ] sailkatuEskaerak(Eskaera [ ] eskaerak);
}
```

Metodo bakoitzak honako funtzionalitate hau izango du:

a. **`EskaerakEman()`** metodoak bezeroaren eskaerak eskuratuko dizkigu.

b. Eskaera `sartuEskaera()` metodoarekin gordetzen dugu, bezeroaren kreditua `Kreditu` balioa baino txikiagoa denean.

c. Eskaerak sailkatu (`sailkatuEskaerak()`) nahi ditugu bezeroaren kredituaren arabera. Horretarako JGL liburutegiaren paketeak erabiliko ditugu, behar ditugun algoritmoak eta predikatuak antolatzeko.

oOo

Zein izango litzateke iturburu-kode honen portaera `exekutatuProzesua()` metodoa konpilatzean eta egikaritzean? Adierazi `exekutatuProzesua()` metodoaren iturburu-kodearen lerro bakoitzaren portaera.

```
interface A {

public void metodo1();
public void metodo3();
}

interface B extends A {

public void metodo2();
}

class A1 implements A {

public void metodo1() {
System.out.println("A1 motako objektuaren –metodo1()– metodoa");
```

```
}
public void metodo3() {
System.out.println("A1 motako objektuaren –metodo3()– metodoa");
}
public void metodo4() {
System.out.println("A1 motako objektuaren –metodo4()– metodoa");
}}

class B1 implements B {

public void metodo1() {
System.out.println("B1 motako objektuaren –metodo1()– metodoa");
}
public void metodo2() {
System.out.println("B1 motako objektuaren –metodo2()– metodoa");
}

public void metodo3() {
System.out.println("B1 motako objektuaren –metodo3()– metodoa");
}  }

public class programaNagusia {

       A1 a11 = new A1();
       B1 b11 = new B1();
       A a1,a2;
       B b1,b2;

public programaNagusia() {};

public static void main(String[] args)
{
programaNagusia instantziNagusia = new programaNagusia();
instantziNagusia.exekutatuProzesua();
}

private void exekutatuProzesua() {

          a1 = a11;
          a2 = b11;
          b1 = a11;
          b2 = b11;
          b1 = a2;
          a1 = b2;

           a2.metodo1();
           a2.metodo2();
           a1.metodo1();
           a1.metodo4();

           b2.metodo3();
           b2.metodo4();} }
```

oOo

Ondoko iturburu-kodean `Langilea` motako bi objektu eraikitzen ditugu. Kontu egin ezen biek `nan` kode bera dutela. Gauzak horrela:

1. Zein irteera izango luke `programaNagusiak` egikaritu ondoren?
2. Zer egin beharko genuke, ezer egin behar badugu, bi instantziak konparatzean `nan`-a bera badute, langile berari dagozkiola azpimarratzeko?

```
import java.text.*;

class Langilea
{
    String nan;
    String izena;

    Langilea(String nan, String izena)
    {
        this.nan = nan;
        this.izena = izena;
    }}

public class programaNagusia {

    public programaNagusia() {  }

    public static void main(String[] args) {
        programaNagusia instantziaNagusia = new programaNagusia();
        instantziaNagusia.exekutatuProzesua();   }

    private void exekutatuProzesua()
    {
        Langilea Ekaitz    = new Langilea("1234", "");
        Langilea EkaitzEre = new Langilea("1234", "");

 idatzi(" Ekaitz objektuaren barne kodea: " + Ekaitz.hashCode());
 idatzi(" EkaitzEre objektuaren barne kodea: " + EkaitzEre.hashCode());
        System.out.println();

        if (Ekaitz.equals(EkaitzEre) == true)
        {
            idatzi("Bi Objektuak berdinak dira");
        }
        else
        {
            idatzi("Bi Objektuak desberdinak dira");
        }  }

    void idatzi(String idatziStr)
    {
        System.out.println(idatziStr);
    }  }
```

oOo

Bektore klasea inplementatu nahi dugu zeinak Javaren **Vector** klasearen funtzionalitatea duen. Horretarako, bere eraikitzaileak eta metodo nagusiak inplementatuko ditugu. Eraikitzaileari ez badiogu parametrorik pasatzen, 10 elementuko array bat sortuko du, bestela, bi eraikitzaile hauek izango ditu: `Bektore(int kopuru)` eta `Bektore(Collection c)`. Bigarren eraikitzailearekin `Collection` motako `c` parametroarekin eraikiko dugu bektorea eta `iterator()` metodoa erabilita eragiketa asko erraztuko da.

`Bektore` klaseak metodo hauek ere izan behar ditu inplementatuta: `elements()`, `elementAt(int indizea)` eta `insertAt(int indizea, Collection c)`.

Ez dago esan beharrik `Bektore` klaseak ezin duela `Vector` klasearen baliabiderik erabili. Gogoratu, halaber, `Array` klaseak ezin duela `Array` motako instantziarik sortu.

oOo

`Tren`, `Hegazkin` eta `Auto` klaseek `Aparatu` interfazea inplementatzen dute honako era honetan:

```
interface  Aparatu {
       static final double zerga = 12.45;
       boolean martxan ();
       boolean aldatu_abiadura(double km-orduko);
       boolean test (Zati z);
                    }

class Tren implements Aparatu {
              private int pisua;
              public int lortu_pisu () { return pisua ;}
              .............}
class Hegazkin implements Aparatu {  .......}
class Auto    implements Aparatu {  .......}
```

`Hegazkin` eta `Auto` berdin egongo dira eratuak.

Klase berri bat (`Kontrolatzaile`) eraiki nahi dugu aparatuak kontrolatzeko:

```
class Kontrolatzaile   {
       ..................................
       public void testeatzaile  (  )   {  ..........};
                   }
```

Klase horrek metodo bat izango du (`testeatzaile()`) honako eragiketa hau gauzatu beharko duena aparatu bakoitzeko:

1. Egiaztatu aparatua martxan dagoela.
2. Aparatua martxan badago, testeatu.

Testa gainditu duten aparatuak gordeko ditu metodoak eta aparatu multzo bat aztertzeko balio beharko du.

Adierazi nolakoa izango den `Kontrolatzaile` klasea goian aipatu dugun funtzionalitatea gauzatzeko.

oOo

Klase bat (**`Matrize`**) sortu nahi dugu, bat, bi edo hiru dimentsio dituzten matrizeak tratatzeko. Berez klaseak erabiliko duen matrizea (`matrize`) dimentsio bakarrekoa izango da. `nDims` atributuak adieraziko du zenbat dimentsio izango duen matrizeak, eta `dimsMatrize` atributuak, dimentsio bakoitzaren balioa.

Adibidea:
```
A[15][10] matrizea
nDims = 2
dimsMatrize = {15, 10}
```

Klaseak honako atributu eta metodo hauek izango ditu:

```
public class Matrize
           {
  private double[] matrize;   // dimentsio bakarreko matrizea
  private int nDims;          // dimentsio kopurua
  private int[] dimsMatriz;   // dimentsio bakoitzaren balioa

  private void eraiki( int[] dim ) // eraikitzailearen lagungarria
  public Matrize() // eraikitzailea
  public Matrize( int d1 ) // eraikitzailea
  public Matrize( int d1, int d2 ) // eraikitzailea
  public Matrize( int d1, int d2, int d3 ) // eraikitzailea
  public int elementuKopuru()
  public int desplazamendu( int[] subind)
  public void balioaEsleitu( int dato, int i1 )
  public void balioaEsleitu( int dato, int i1, int i2 )
  public void balioaEsleitu( int dato, int i1, int i2, int i3 )
  public double balioaEskuratu( int i1 )
  public double balioaEskuratu( int i1, int i2 )
  public double balioaEskuratu( int i1, int i2, int i3 )

}
```

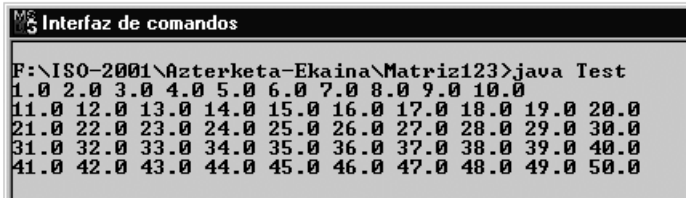
Hona hemen `Matrize` klasearen erabilera bat eta bera egikarituta sortuko lukeen irteera:

```
public class Test
{
  public static void main(String[] args)
  {
    final int A = 5;
    final int B = 10;
    int i, j, konta = 1;
```

```
Matrize m = new Matrize( A, B );

// m matrizeari balioak esleitu
for ( i = 0; i < A; i++ )
  for ( j = 0; j < B; j++ )
    m.balioaEsleitu(konta++, i, j );

// m matrizea ikuskatu
for ( i = 0; i < A; i++ )
{
  for ( j = 0; j < B; j++ )
    System.out.print(m.balioaEskuratu( i, j ) + " ");
  System.out.println();
} }  }
```



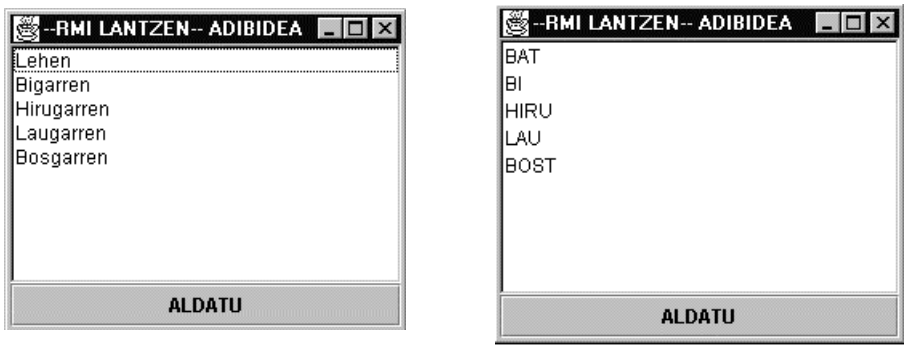
7.1. irudia. Ariketaren exekuzioa irteera estandarrean.

oOo

7.2. JAVA: RMI ARIKETAK

Programa bat eraiki nahi dugu, RMIren ezaugarri nagusiak erabiliko dituena. Programa horrek leiho bat aterako du zerrenda lokalean dagoen informazioarekin eta botoi bat eskainiko du erabiltzaileak zapal dezan. Botoia zapaltzen badu erabiltzaileak, zerrenda aldatu egingo da. Zerrendaren balio berriak eskuratzeko, urruneko objektu bat erabiliko da.

Aplikazioak nola funtzionatzen duen adierazten dute leiho hauek:



7.2. irudia. Ariketaren leiho nagusiak.

Hona hemen osatu behar duzun iturburu-kodea:

```
/**
* Programa honek leiho bat aterako du zerrenda lokalean dagoen
* informazioarekin eta botoi bat eskainiko du erabiltzaileak zapal
* dezan. Erabiltzaileak botoia zapaltzen badu zerrenda aldatu
egingo
* da. Zerrendaren balio berriak eskuratzeko urruneko objektu bat
* erabiliko da.
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.rmi.*;

class java2004B extends JFrame implements ActionListener {


    };

  // Leihoaren osagarri grafikoak
  JButton botoi;
  JPanel panel;
  // Zerrenda lokala eta urruneko zerrenda



  java2004B() {
    // Zerrenda lokala sortu eta hasieratu egiten dugu



    // Zerrenda eguneratzeko botoia



    // Osagai guztien edukiontzia
    panel = new JPanel();
    panel.setLayout( new BorderLayout() );
    panel.setBackground( Color.white );
    getContentPane().add( panel );
    // Osagiak panelean sartzen ditugu
    panel.add( "Center",zerrenda );
    panel.add( "South",botoi );
    }


  // Ekintzak gauzatzeko tokia botoia zapaldu eta gero

  public void actionPerformed( ActionEvent gert ) {
    // Gertaeraren sorrera aztertzen dugu
    Object sorrera = gert.getSource();
```

```
    if( sorrera == botoi ) {
      // Zerrendaren datuak eskatzen dizkiogu urruneko objektuari
      // eta zerrenda lokala eguneratzen dugu



      } catch( Exception e ) {
        e.printStackTrace();
        }
      }
    }

  // Bezeroaren metodo nagusia
  public static void main( String[] args ) {




    WindowListener wl = new WindowAdapter() {
      public void windowClosing( WindowEvent evt ) {
        System.exit( 0 );
        }
      };

    frame.addWindowListener( wl );
    frame.pack();
    frame.setVisible( true );
    }
  }

//---------------------- java2004B.java
//---------------------- fitxategiaren bukaera

                                oOo


//  java2004I.java


/**
 * Urruneko Zerrenda tratatzeko RMI interfazea
 */
import java.rmi.*;

public interface java2004I extends Remote {



}

//---------------------- java2004I.java
//---------------------- fitxategiaren bukaera
```

```
                                    oOo
//  java2004L.java


/**
 * Zerrenda objektua. DATUAK eguneratzeko metodoa inplementatzen du.
 */
import java.awt.*;
import java.awt.event.*;

public class java2004L extends List {

  public java2004L() {}

  public java2004L( int osagaiak ) {
    super( osagaiak );
    }

  public java2004L( int osagaiak,boolean eraAnizkuna ) {
    super( osagaiak,eraAnizkuna );
    }

  // DATUAK eguneratzeko metodoak
  public synchronized void setDATUAK( String[] DATUAK ) {
    // Lehenengo egungo balioak ezabatzen ditugu
    removeAll();
    // Parametro gisa pasatutako osagaiak kopiatzen ditugu zerrendan



    }
  }

//---------------------- java2004L.java
//---------------------- fitxategiaren bukaera

                                    oOo

//  java2004Z.java


/**
* Urruneko zerbitzaria.
* Urruneko zerrendaren interfazea inplementatzen du
 */
import java.net.*;
import java.rmi.*;
import java.rmi.server.*;

public class java2004Z
```

```
  // Eraikitzailea
  public java2004Z() throws RemoteException {}
  // Zerrenda itzultzeko metodoa



    }

  public static void main(String args[]) {

    try {
      // Segurtasun kontrolatzailea instalatzen dugu




       }

      // Urruneko objektua sortzen dugu




      // Erregistratu egiten dugu sortutako objektua



    } catch( Exception e ) {
      e.printStackTrace();
                                  return;
    }
  }
}

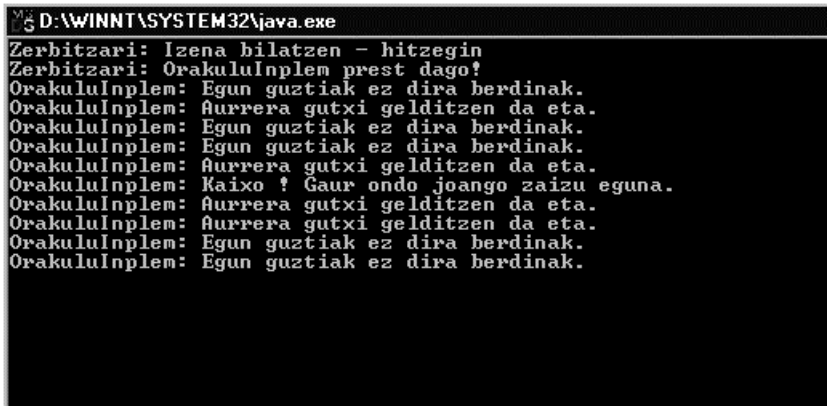
//---------------------- java2004Z.java
//---------------------- fitxategiaren bukaera
```

oOo

Programa bat eraiki nahi dugu honako betekizun hauekin: bezeroaren aldetik esaldi bat itzultzeko eskatuko dio programak zerbitzariari. Agindua jaso ondoren, aukeraketa egingo du zerbitzariak ausazko eran `String` array batetik eta esaldi bat itzuliko dio bezeroari.

Programak diseinu banatua izango du eta hori gauzatzeko Javaren RMI paketea erabiliko du.

Bezeroaren programa nagusiak hamar aldiz deituko dio zerbitzariari esaldiak eskatzeko. Hona hemen nola erantzungo lukeen zerbitzariak:



7.3. irudia. Ariketaren emaitza irteera estandarrean.

Hona hemen osatu behar duzun iturburu-kodea:

```
import java.rmi.*;

/**
 *      Fitxategia:    Bezeroa.java
 *
 *
 */

public class Bezeroa {

       public static void main(String[] args) {

              try {


       System.out.println("Begiratzen nago !!!!..." + OrakuluIzena);


       System.out.println("bilatuta!");

       for (int i=0; i < 10; i++) {


                              }


              }
              catch (Exception exc) {
                     exc.printStackTrace();
              }
System.exit(0); }}
```

```
import java.rmi.*;
import java.rmi.server.*;

/**
 *     Fitxategia:    Zerbitzari.java
 *
 */
public class Zerbitzari {

       public static void main(String[] args) {

              try {
       // Segurtasun kontrolatzailea instalatzen dugu



        }




     System.out.println("Zerbitzari: Izena bilatzen - " +
ObjektuIzena);


    System.out.println("Zerbitzari: OrakuluInplem prest dago!");
                                                }
              catch (Exception exc) {
                     exc.printStackTrace();
              }
       }
}
```

oOo

```
 import java.rmi.*;

/**
 *     Fitxategia:      OrakuluI.java
 *
 *     Urruneko Interfazea.
  */
public interface OrakuluI extends Remote {

       public String getEmaitza() throws RemoteException;

}
```

oOo

```
import java.rmi.*;
import java.rmi.server.*;
```

```
/**
 *     Fitxategia:   OrakuluInplem.java
 *
 *    Urruneko Interfazearen Inplementazioa.
 *     Ausazko esaldia itzultzen du eta itzulitakoa ikuskatu egiten da
 */


public class OrakuluInplem
                      extends UnicastRemoteObject
                      implements OrakuluI {

       private String[] esaldiak =
                          {"Kaixo ! Gaur ondo joango zaizu eguna.",
                           "Egun guztiak ez dira berdinak.",
                           "Aurrera gutxi gelditzen da eta."};

        //   Eraikitzailea
        public OrakuluInplem() throws RemoteException {
        }

        // Urruneko Interfazearen metodoaren inplementazioa
        public String getEmaitza() throws RemoteException {









              return emaitza;
       }

       // Barne metodoa Emaitza ikuskatzeko
       private void ikusi(.....) {



       }
}
```

oOo

Javaren RMI eta JDBC teknologiak erabilita, ostalari-zerbitzarian dagoen datu-base batera atzitu nahi dugu informazioa txertatzeko edota eskuratzeko.

Hona hemen osatu behar duzun iturburu-kodea:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.sql.*;

public interface DatuBaseaRMI extends Remote
{
```

```
    public DatuakEskaera bilatuEskaera(int n) throws
RemoteException,SQLException;
    public boolean txertatuEskaera(int n,int c,String f) throws
RemoteException,SQLException;
    public String lorBertsio() throws RemoteException;
    public String lorzerbitzari() throws RemoteException;
}
```

```
=====================================================================
```

```
import java.io.*;

class DatuakEskaera implements Serializable
{
 private int eskaerazenb;
 private int kostu;
 private String data;
 DatuakEskaera(int num,int kos,String data)
 {
  this.eskaerazenb=num;
  this.kostu=kos;
  this.data=data;
 }
 public int lorEskaera(){ return this.eskaerazenb;}
public int lorKostu(){ return this.kostu;}
 public String lorData(){ return this.data;}
}
```

```
=====================================================================
```

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.sql.*;
import java.sql.Connection;
import java.io.PrintStream;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class DatuBasea extends UnicastRemoteObject
    implements DatuBaseaRMI {
```



```
    public static final String bertsio="DATU-BASEAREN BERTSIOA : 1.0";
  public static final String zerbitzari="ZERBITZARIA: 158.227.112.99";


    public DatuBasea() throws RemoteException
          {
     super();
     try
     {
```

```
       System.out.println("Datu-basearekin konektatzen");
   // Izena: DBAdibidea
```



```
     System.out.println("Datu-basearekin konektatuta");
     }
     catch (Exception e)
     {
      System.out.println("Ez da estekadura lortu: "+e.toString());
     }
    }

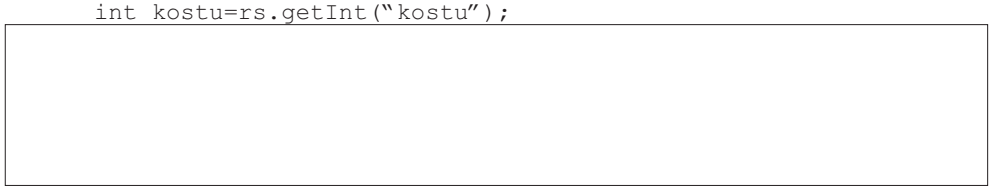

    public DatuakEskaera bilatuEskaera(int n) throws
RemoteException,SQLException
    {
```



```
try
     {
      System.out.println("Kontsulta gauzatzen....");
```



```
      if (rs.next())
      {
      int eska=rs.getInt("eskaerazenb");
      int kostu=rs.getInt("kostu");
```



```
}
      System.out.println("Kontsulta gauzatuta.");
     }
     catch (SQLException sqle){throw new SQLException("Errorea
kontsultaren exekuzioan, eskaera:  "+n+". "+sqle.toString());}
     return [     ];
    }
    public boolean txertatuEskaera(int n,int c,String data) throws
RemoteException,SQLException
    {
     Statement kontsulta;
     int lerroak;
     try
     {
```

```
      System.out.println("Txertaketa gauzatzen....");
```



```
System.out.println("Txertaketa gauzatuta.");
     }
     catch (SQLException sqle){throw new SQLException("Errorea
kontsultaren eguneratzearen exekuzioan, eskaera: "+n+".
"+sqle.toString());}
     return (lerroak>0);
    }
    public String lorBertsio() throws RemoteException
    {
     return this.bertsio;
    }
    public String lorzerbitzari() throws RemoteException
    {
      return this.zerbitzari;
    }

    public static void main(String args[])
    {
     System.setSecurityManager(new RMISecurityManager());
     try {
```



```
         System.out.println("Datuen objektu zerbitzaria
erregistratu egin da");
         }
        catch(Exception exception)
        {
            exception.printStackTrace();
        }
    }
}
```

```
=====================================================================

import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.sql.*;

public class BezeroDB {

    public BezeroDB()
    {
    }

    public static void main(String args[]) {
        System.setSecurityManager(new RMISecurityManager());
```

```
        DatuakEskaera de;
        boolean emaitza;
        try
        {
```



```
  System.out.println("Zerbitzariaren Bertsioa: "+bd.lorBertsio());
  System.out.println("Zerbitzari           : "+bd.lorzerbitzari());
            System.out.println("<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>");
    System.out.println("*********Datuak txertatzen***********");
            emaitza=bd.txertatuEskaera(1,200,"26/12/2000");
            emaitza=bd.txertatuEskaera(2,400,"24/12/2000");
            emaitza=bd.txertatuEskaera(3,1200,"30/12/2000");
 System.out.println("*********Datu txertaketa bukatuta***********");
 System.out.println("*********Datuak Kontsultatzen***********");
       // Bilatu Bigarren eskaera
```

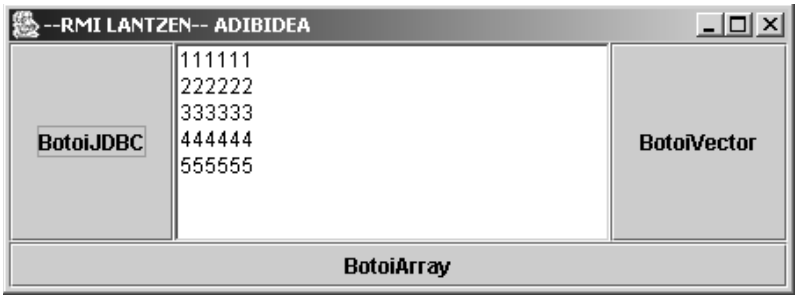


```
    System.out.println("Eskaeraren Zenbakia: "+de.lorEskaera());
            System.out.println("Eskaeraren Kostua  : "+de.lorKostu());
            System.out.println("Eskaeraren Data: "+de.lorData());
        }
        catch(Exception exception)
        {
              System.out.println("Datuen zerbitzariak ez du erantzun
");
            exception.printStackTrace();
        } }  }
```

oOo

Ariketa hau aurretik egindako baten luzapena da. Oraingoan, RMIren malgutasuna probatuko dugu urruneko objektuei deiak egiteko orduan. Bezeroak, objektuaren kokapena eskuratu ondoren, objektuaren zerbitzuak eskatuko ditu mezuak bidalita. Gure kasuan, urruneko objektuaren `getDATUAK()` metodoa da zerbitzu hori. Gogoan izan ezen bezeroari berdin zaiola metodo hori nola dagoen inplementatuta; berak nahi du urruneko objektuari dei egiten dionean objektu horrek `String`-ez osatutako array bat itzul dezan. Programa honetan aplikazioaren interfaze grafikoa zertxobait aldatuko dugu eta aukera emango diogu erabiltzaileari hiru botoi zapaltzeko. Botoi bakoitzak bere urruneko objektua atzituko du eta eskuratutako informazioa erdiko leihoan bistaratuko da. Gauzak horrela, ***BotoiJDBC*** botoia zapaltzen badu erabiltzaileak, `192.168.37.10` IP helbidean dagoen objektura atzitzen du bezeroak. Han dagoen zerbitzariak bezeroak behar duen informazioa datu-base batetik eskuratzen du eta bezeroari itzuliko dio (**7.5. irudia**). Aldiz, ***BotoiVector*** botoia zapaltzen badu erabiltzaileak, `158.165.35.10` IP helbidean dagoen objektura atzituko du bezeroak. Orduan, han dagoen zerbitzariak, bezeroak behar duen informazioa `Vector` motako objektu

batetik eskuratuko du eta bezeroari itzuliko dio (**7.6. irudia**). ***BotoiArray*** botoia zapaltzen badu erabiltzaileak, `157.135.38.10` IP helbidean dagoen objektura atzituko du bezeroak eta han dagoen zerbitzariak bezeroak behar duen informazioa eskuratuko du `String`-ez osatutako array batetik.



7.4. irudia. Bezeroaren aldean hasieran agertzen den leihoa.



7.5. irudia. Datu-basera atzituta eskuratutako informazioa.



7.6. irudia. `Vector` motako objektura atzituta eskuratutako informazioa.

oOo

Aplikazio informatiko bat diseinatu eta inplementatu nahi dugu gure enpresako eskaerak kudeatzeko. Aplikazioaren datuak datu-base erlazional batean gordeko ditugu eta eginkizun hau gauzatzeko zerbitzari bat erosi du enpresak

oraintsu eta gure esku utzi du hura antolatzea. Hori dela-eta, inplementatu nahi dugun aplikazioa egungo teknologietan oinarritu nahi dugunez, Javaren RMI teknologia erabiltzea erabaki dugu, eta, datu-basearen informaziora atzitzeko zerbitzari horretan kokatuko dugu datu-basea bere software eta guzti. Horrela, urruneko edozein bezerok zerbitzua eskatzen duenean RMI teknologia erabilita, datu-basera atzituko du zerbitzariak eta eskatutakoa gauzatu ondoren, urruneko bezeroari erantzungo dio.

Datu-basera atzitzeko software guztia zerbitzarian kokatuko dugu eta bezeroarentzat erabat gardena izango da datuetara atzitzea.

Aplikazioa diseinatzeko eta inplementatzeko enpresako arduradunekin hitz egin dugu eta pantaila bat (**7.7. irudia**) diseinatu dugu gure beharretara egokitzen dena. Pantaila horrek funtzionalitate hau edukitzea nahi genuke:

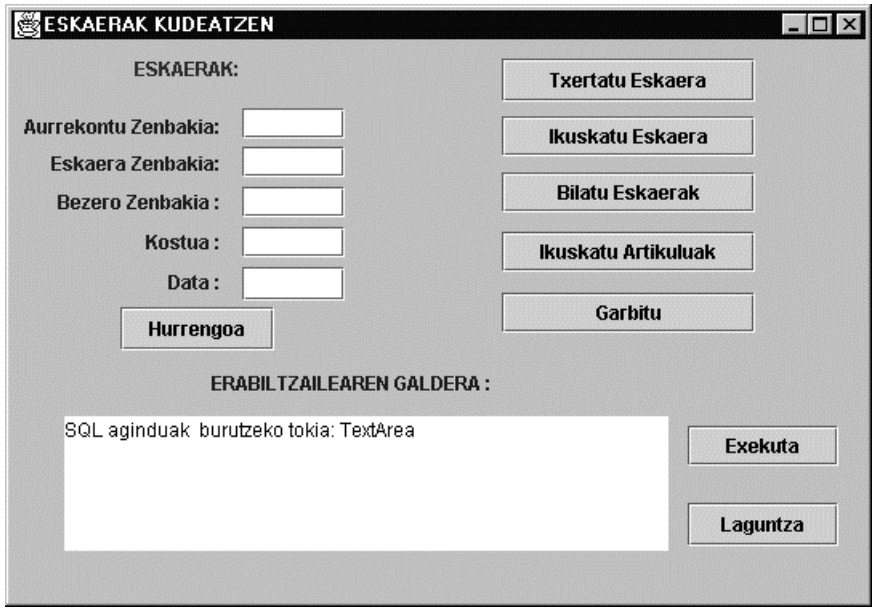
1. ***Txertatu Eskaera*** botoia zapalduta pantailaren ezkerraldean sartu dugun informazioa erabilita eskaera bat sortuko du sistemak eta, ondoren, bigarren pantaila (**7.8. irudia**) batera pasatuko du kontrola eskaeraren artikuluaren informazioa sartzeko. Dena ondo badago, eskaera oso bat sortuko du sistemak.

2. **Ikusketa Eskaera** botoia zapalduta eskaera eskuratzen digu sistemak datu-basetik eta pantailan bistaratu. Kasu honetan aurrekontu-zenbakia sartzen badugu, eskaerarik esleitua duen ala ez esaten digu sistemak. Ondoren ***Ikuskatu Artikuluak*** botoia zapal dezakegu eskaeraren artikuluak bistaratzeko.

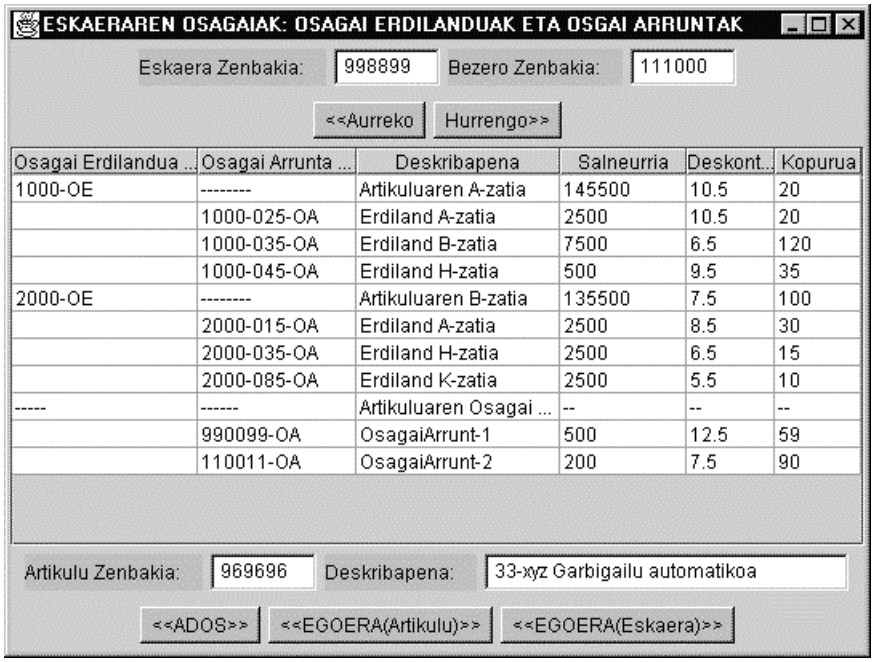
3. Bezeroaren zenbakia sartuta eta ***Bilatu Eskaerak*** botoia zapalduta bezero horrek dituen eskaerak eskuratuko ditugu datu-basetik eta bere artikuluak ikusteko, lehen bezala, ***Ikuskatu Artikuluak*** botoia zapal dezakegu.

4. ***Hurrengoa*** botoiak bezeroaren hurrengo eskaera eskuratzen digu, eta 3. atalean esan duguna egikaritzen du ondoren.

5. Orain arte esandako guztia erabiltzaile arruntarentzat —idazkaria, gerentea, saileko zuzendaria…— pentsatua dago, hau da, ezaguera informatiko gutxi duen pertsonalarentzat. Erabiltzaile adituarentzat beste funtzionalitate hau prestatu dugu: testu-area bat diseinatu dugu erabiltzaileak berak eratutako `SELECT` agindua egikaritzeko. Horrela, datu-basearen tauletako informazioa atzi dezake erabiltzaileak eta bertako informazioa eskuratu zuzenean eta testu-area berean ikusi. Are gehiago ere egin dezake, erabiltzailea gauza bada —datu-basearen taulak eta atributuak nola dauden diseinatuta ezagutzen badu— testu-area erabil dezake informazioa eguneratzeko (`UPDATE`), txertatzeko (`INSERT`) eta ezabatzeko(`DELETE`).



7.7. irudia. Ariketaren pantaila nagusia.



Osagai Erdilandua ...	Osagai Arrunta ...	Deskribapena	Salneurria	Deskont...	Kopurua
1000-OE	--------	Artikuluaren A-zatia	145500	10.5	20
	1000-025-OA	Erdiland A-zatia	2500	10.5	20
	1000-035-OA	Erdiland B-zatia	7500	6.5	120
	1000-045-OA	Erdiland H-zatia	500	9.5	35
2000-OE	--------	Artikuluaren B-zatia	135500	7.5	100
	2000-015-OA	Erdiland A-zatia	2500	8.5	30
	2000-035-OA	Erdiland H-zatia	2500	6.5	15
	2000-085-OA	Erdiland K-zatia	2500	5.5	10
-----	------	Artikuluaren Osagai ...	--	--	--
	990099-OA	OsagaiArrunt-1	500	12.5	59
	110011-OA	OsagaiArrunt-2	200	7.5	90

7.8. irudia. Eskaeraren artikuluen informazioa ikusteko interfaze grafikoa.

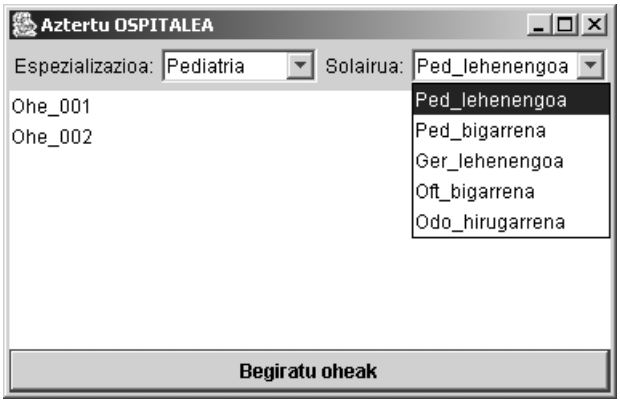
Ariketaren bigarren zati honetan honako hau eskatzen da:

1. Emandako informazioa kontuan izanda, adierazi zenbat erabilera-kasu agertzen diren informazioan eta azaldu laburki bakoitzaren ezaugarriak.
2. Azaldu zein diseinu-patroi erabiliko duzun eta zertarako.
3. Programatu aplikazioaren iturburu-kodea adieraziz nolako Sekuentzia Diagrama erabili duzun. Bai diseinuan bai inplementazioan, saia zaitez diseinu eta inplementazio berrerabilgarriak sortzen.
4. Zer aldatu beharko zenuke bezeroaren aurrekontuaren eta eskaeraren ordez, hornitzailearen aurrekontu eta eskaerak tratatu nahi bagenitu?

oOo

7.3. ARIKETAK: HIRU MAILAKO ARKITEKTURA

`KontsultatuOheak` klaseak `aztertu oheak` erabilera-kasua inplementatzen du. Erabilera-kasuak, espezializazio bat eta solairu bat aukeratu ondoren, solairu horretan dauden oheak eskuratzen ditu.



7.9. irudia. Ariketaren interfaze grafiko nagusia.

Espezializazioak, solairuak eta oheak honako taula hauetatik ateratzen dira:



7.10. irudia. Datu-basearen taula nagusiak.

Honako hau eskatzen da:

Egokitu ebazpena hiru mailako arkitektura bat proposatuta. Horretarako, beharrezkoak izango diren klaseak diseinatu behar dituzu UML notazioa erabilita. Klaseen barnean adierazi erabili beharreko metodoak eta haien signatura. Ebazpena ez duzu osorik inplementatu behar baina bai adierazi Javaren RMI, JDBC eta interfazeen erabilera.

Hasierako iturburu-kodea honako hau da:

```
package ospitalea;
import ....

public class KontsultatuOheak extends JFrame
{
  private JPanel jPanel1 = new JPanel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JLabel jLabel1 = new JLabel();
  private DefaultComboBoxModel model1 = new DefaultComboBoxModel();
  private JComboBox jComboBox1 = new JComboBox(model1);
  private JLabel jLabel2 = new JLabel();
  private DefaultComboBoxModel model2 = new DefaultComboBoxModel();
  private JComboBox jComboBox2 = new JComboBox(model2);
  private JButton jButton1 = new JButton();
  private DefaultListModel model3 = new DefaultListModel();
  private JList jList1 = new JList(model3);
  private Statement st;
  private ResultSet rs;
  private static Connection konexioa;

  public KontsultatuOheak()
  { try { jbInit(); } catch(Exception e){ e.printStackTrace();} }
  private void jbInit() throws Exception
  {   try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      System.out.println("driverra kargatuta");
      konexioa = DriverManager.getConnection("jdbc:odbc:dbOspitalea");
      System.out.println("konexioa sortua");
      st = konexioa.createStatement();  }
    catch (Exception e){
       System.out.println("Errorea: " +e.toString());
    }
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Aztertu OSPITALEA");
    jLabel1.setText("Espezializazioa:");
    jLabel2.setText("Solairua:");
    jButton1.setText("Begiratu oheak");
    jButton1.setFont(new Font("Dialog", 1, 12));
    jButton1.addActionListener(new ActionListener()
```

```
      {
        public void actionPerformed(ActionEvent e)
        { jButton1_actionPerformed(e);  }  });
    jPanel1.add(jLabel1, null);
    jPanel1.add(jComboBox1, null);
    jPanel1.add(jLabel2, null);
    jPanel1.add(jComboBox2, null);
    osatuModeloa(model1,lortuElementuak("SELECT ESPEZIALIZAZIOA FROM
OSPITALEA"));
    osatuModeloa(model2,lortuElementuak("SELECT SOLAIRUA FROM
OSPITALEA"));

    this.getContentPane().add(jPanel1, BorderLayout.NORTH);
    this.getContentPane().add(jButton1, BorderLayout.SOUTH);
    this.getContentPane().add(jList1, BorderLayout.CENTER);
  }
  public void osatuModeloa(DefaultComboBoxModel d,Vector elems)
  {
    d.removeAllElements();
    for (int i=0;i<elems.size();i++)
      { d.addElement(elems.elementAt(i)); }  }
  public void osatuModeloa(DefaultListModel d,Vector elems)
  {
    d.removeAllElements();
    for (int i=0;i<elems.size();i++)
      { d.addElement(elems.elementAt(i)); }  }
 public Vector lortuElementuak(String sql)
  {   Vector v = new Vector();
    try{
      rs = st.executeQuery(sql);
      while (rs.next())
        {v.addElement(rs.getString(1)); }
      System.out.println("Combobox osagaian kokatuak");  }
    catch(Exception e) {System.out.println("Errorea: "+e.toString());}
    return v;  }
  private void jButton1_actionPerformed(ActionEvent e)
  {  osatuModeloa(model3,
      lortuElementuak("SELECT ohea "+
  "FROM Oheak INNER JOIN Ospitalea ON Oheak.kodea=Ospitalea.kodea "+
   "WHERE ESPEZIALIZAZIOA='"+jComboBox1.getSelectedItem()+"' AND
SOLAIRUA='"+jComboBox2.getSelectedItem()+"'"));      }}
```

oOo

Web gune bat eraiki nahi da erosketa elektronikoak kudeatzeko. Horretarako, hiru mailako arkitektura erabiliko da aplikazio informatikoa eraikitzeko. Ariketa honetan arkitekturaren hirugarren maila landuko dugu, hau da, Enpresaren Informazio Sistema. Eraikiko den aplikazio informatikoa ez da hutsetik sortuko eta aurretik genuen informazio-sistema berrerabili egingo da. Hori dela-eta, hirugarren maila hori ahal den eta malguen eta egokien diseinatu behar da.

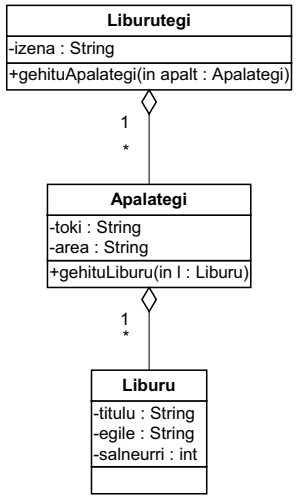
Bere bilakaera historikoan era desberdinak erabili ditu enpresak domeinuaren entitateak (bezeroak, artikuluak, hornitzaileak, eskaerak...) biltegiratzeko. Hala, datu-base erlazional batean biltegiratu ditu bezeroak artikuluak eta eskaerak. Aldiz, kamioiak eta garraiolariak objektuetara zuzendutako datu-base batean biltegiratzen ditu, eta produktuen hornitzaileak serializatuta gordetzen ditu.

Honako hau eskatzen da:

- Diseinatu aipatutako arkitekturaren hirugarren mailaren Klase Diagrama. Diseinuan patroiren bat erabiltzen baduzu, adierazi zergatik eta non erabiltzen den.
- Programatu eta adierazi Sekuentzia Diagrama txiki baten bidez beharbeharrezkoak diren metodoak, metodo horien signatura (parametroak[mota eta kopurua]) eta metodoak gauzatuko lukeen eragiketa.

7.4. ARIKETAK: JGL (JAVA GENERIC LIBRARY)

Liburutegi batean liburuak apalategietan ordenatu ditugu ondoko UML diagramaren egiturari jarraituta:



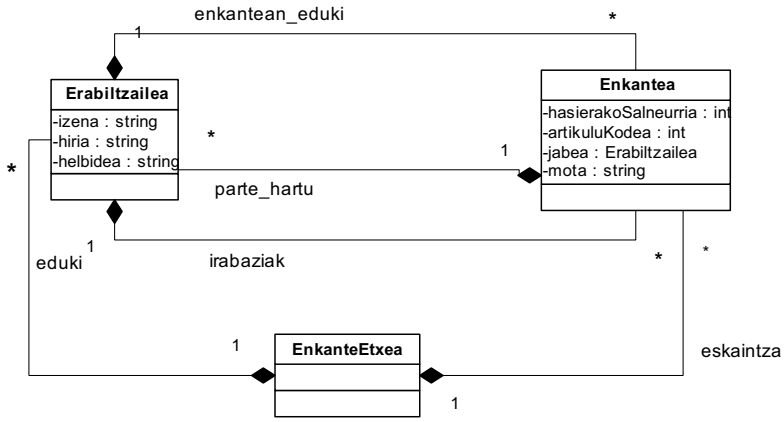
7.11. irudia. Ariketaren Klase Diagrama.

JGLren liburutegiak erabilita, `Liburutegi` eta `Apalategi` klaseen ondorengo metodoak inplementatu nahi ditugu, beharrezkoak diren atributuak ere adierazita bide batez:

1. `bistaratuApalategikoLiburuak( )` metodoa: liburutegiaren apalategi bakoitza bistaratuko du alfabetikoki (**`toki`**aren arabera) ordenatuta eta bertako liburuak, horiek ere alfabetikoki (**`titulu`**aren arabera) ordenatuak.

2. `Apalategi` klasearen `handienakIkuskatu(int s)` metodoa: metodo honek `s` salneurria baino salneurri handiagoa daukaten liburuak erakusten ditu.

oOo



7.12. irudia. Ariketaren Klase Diagrama.

Goiko irudian agertzen den UML Klase Diagrama enkante-etxe batek erabiltzen du bere eguneroko jarduna kudeatzeko. Irudiko (**7.12. irudia**) erronbo beltz guztiek UMLren agregazio-erlazioa adierazten dute. Enkante bakoitzerako honako hau adierazten dugu: hasierako salneurria zein izango den, enkantean saltzen ari garen artikuluaren kodea, zein den artikuluaren jabea eta zein den artikuluaren mota.

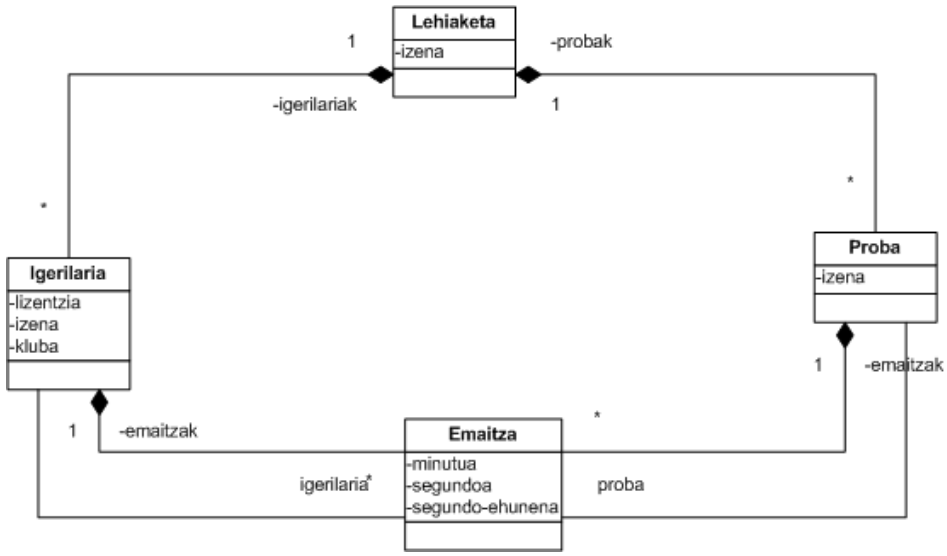
JGLren edukiontziak, predikatuak, funtzioak eta algoritmoak erabilita, programatu honako metodo hauek:

1. `Erabiltzailea` klasean, metodo bat programatu erabiltzailearen hiri berean dauden enkante guztiak itzultzen dituena.

2. `EnkanteEtxea` klasean, metodo bat programatu erabiltzaileak itzultzen dituena honako baldintza hau betetzen denean: erabiltzaileak gutxienez enkante bat dauka eskaintzan enkante-etxean.

3. `EnkanteEtxea` klasean, metodo bat programatu erabiltzaileak itzultzen dituena honako baldintza hau betetzen denean: erabiltzailearen enkante batean, gutxienez norbaitek parte hartu du.

oOo



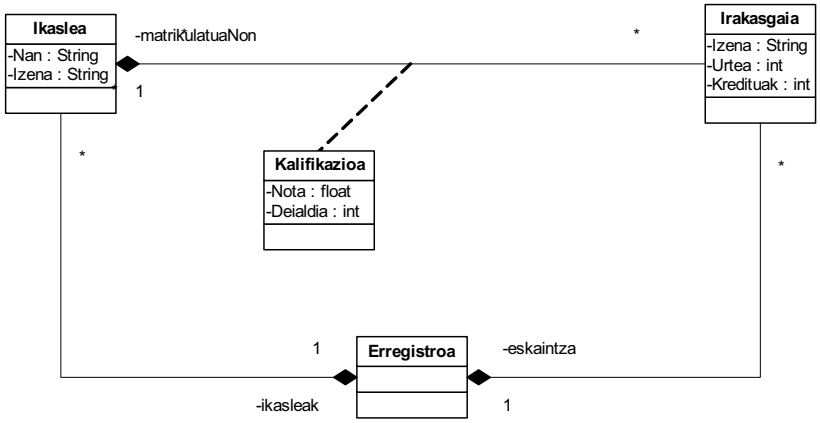
7.13. irudia. Ariketaren Klase Diagrama.

Aplikazio informatiko bat eraiki nahi da igeri-lehiaketa bat kudeatzeko. Igeri-lehiaketa bat `Igerilariz` eta `Probaz` osatzen da, igerilari batek parte hartzen badu proba batean, emaitza bat eskuratzen du. Denbora (minutu, segundo, segundo-ehunena) eta beste bi atributu gehiago izango ditu emaitzak adieraziz: zein izan den igerilaria eta zein probatan parte hartu duen. Goiko UML diagraman aplikazioak izango dituen klaseak eta erlazioak adierazten dira.

Honako metodo hauek programatzea eskatzen da `Lehiaketa` klasean JGL liburutegiaren algoritmoak eta predikatuak erabilita:

1. Bistaratu `proba` proban parte hartu duten igerilariak denboraren arabera sailkatuta:
   `void igerilariakProba(String proba)`
2. Bistaratu `n` probatan baino gehiagotan parte hartu duten igerilariak:
   `void igerilariaknProbaGehiago(int n)`
3. Bistaratu `proba` proban `m` minutu, `s` segundo eta `se` segundo-ehuneneko baino denbora gutxiago egin duten igerilariak:
   `void igerilariakDenboraGutxiago(String proba,int m,int s,int se)`

oOo



7.14. irudia. Ariketaren Klase Diagrama.

Ikastetxe baten erregistroan ikastetxe horretako ikasleak eta eskaintzen diren irakasgaiak biltegiratzen dira. Ikasle bakoitzak Hash taula bat (`matrikulatuaNon`) dauka matrikulatuta dagoen irakasgaien (GAKO) zerrenda biltegiratzen duena eta lortutako kalifikazioa (BALIO) kalifikatuta badago.

JGLren predikatuak eta algoritmoak erabilita, honako metodo hauek programatu behar dituzu:

1. `Erregistroa` klasean, bistaratu U urtean eskaintzen diren irakasgaiak, izenaren arabera sailkatuak.
2. `Ikaslea` klasean, programatu metodo bat ikasleak gainditu dituen irakasgaiak bistaratzen dituena.

Oharra: Gogoratu Hash taula bateko GAKO-BALIO bikote bakoitza `Pair` motako objektu batekin eraikitzen dela. `Pair` objektuaren GAKO balioa `first` aldagaiarekin eskuratzen da eta BALIO objektuaren balioa `second` aldagaiarekin.

oOo

JGL tresnaren liburutegia hedatu nahi da egitura berri bat eraikita. Egitura horrek `arau` bat egikaritu behar du elementu multzo bati aplikatzen zaionean.

Semantikoki arau batek honako egitura hau izaten du:

BALDIN `Baldintza<O>` **ORDUAN** `Ekintza<O>`

Hau da, `O` objektuari baldintza (`Baldintza`) bat aplikatzen zaionean eta egia (`true`) bada, ekintza (`Ekintza`) bat egikaritzen da objektuan.

Araua definitu ondoren, elementuz osatutako edukiontzi batean aplika daiteke honako era honetan: `(Container elementuak, Araua araua)`.

Honako hau eskatzen da:

1. Definitu aipatutako `Araua`, JGLren baliabideak erabilita.
2. Nola definituko zenuke algoritmo bat, aipatutako araua elementuz osatutako edukiontzi bati aplikatzeko?

Programa bat eraiki, araua edukiontzi bati aplikatzeko, honako ezaugarriekin: edukiontziak zenbaki osoak ditu biltegiratuta eta, zenbaki guztiak zeinu positiboz jartzen ditu arauak. Hau da, hasieran, edukiontzia honela badago: 4, -5, -6, 7, 3. Bukaeran, honela egon behar du: 4, 5, 6, 7, 3.

oOo

JGL tresnaren liburutegia hedatu nahi da egitura berri bat eraikita. Egitura horrek arau bat egikaritu behar du elementu multzo bati aplikatzen zaionean.

Semantikoki arau batek honako egitura izaten du:

BALDIN `Baldintza<O>` **ORDUAN** `Ekintza<O>`

Hau da, `O` objektuari baldintza (`Baldintza`) bat aplikatzen zaionean eta egia (`true`) bada, ekintza (`Ekintza`) bat egikaritzen da objektuan.

Araua definitu ondoren, elementuz osatutako edukiontzi batean aplika daiteke honako era honetan: (`Container elementuak, Araua araua`).

Arauak honako inplementazio hau izango du:

```
class Araua implements UnaryFunction {
      UnaryPredicate up;
      UnaryFunction  uf;
       Object execute(Object objektu) {
     if(up.execute(objektu) ) then  return(uf.execute(objektu))  ;
                    else return null;
                    }
            }
```

Nola definituko zenuke algoritmo bat, arau bat eta edukiontzi bat emanda emaitza beste edukiontzi batean itzuliko lukeena? Emaitza, jakina, hasierako edukiontziari araua aplikatuta eskuratzen da. Hau da, elementu bat emaitzaren edukiontzira pasatzeko arauaren baldintza-atalak `true` bat itzuli behar du. Adibidez, pertsonaz `(izena, hiria, nan)` osatutako edukiontzi bat arakatu, eta, biltegiratu pertsona horien `nan`-a beste edukiontzi batean, baldin eta `Donostian` bizi badira.

Honako hau eskatzen da:

1. Definitu `Algoritmo.arauAplikatu` algoritmoa.
2. Aplikazio txiki bat programatu araua aplikatzeko.

oOo

7.5. ARIKETAK: EJB (ENTERPRISE JAVABEANS)

EJB osagai bat (sesio-bean bat) programatu nahi da bankuen arteko diru-transferentziak kudeatzeko. Diru-transferentzia lau urratsetan gauzatuko da. Urratsak ez dira aldi berean egikaritzen, sekuentzian baizik:

1. Aukeratu erabiliko dugun iturburu-kontua, diru-transferentzia gauzatzeko.
2. Aukeratu erabiliko dugun kontua, diru-transferentziaren diru kopurua jasoko duena.
3. Aukeratu diru kopurua.
4. Egin diru-transferentzia.

Kontu egin ezen bankuko bezero bakoitzaren kontuaren informazioa (`kontuzenbakia`, `helbidea` eta `balantzea`) datu-base batean gordetzen dela eta JDBC erabili behar dela bezeroaren kontura atzitzeko.

oOo

Akzioekin egiten diren eragiketak kudeatzeko, EJB osagai bat dauka Bilboko Burtsak honako interfazearekin:

```
void erosi(String akzioa, int kopurua)
void saldu(String akzioa, int kopurua)
int lortuBalioa(String akzioa)
```

Osagai hori erabilita, `Akziodun` osagai bat (sesio-bean bat) inplementatu nahi da, goiko osagaia erabilita, beste honako eragiketa hauek gauzatzeko:

1. Akziodunaren kontua hasieratu, izen bat eta kopuru bat biltegiratuta.
2. Handitu akziodunaren kontua, beste balio bat gehituz.
3. Akziodunak saldoa badauka, onartu “x” akzioen erosketa.
4. Onartu “y” akzioen salmenta akzio horien jabea bada.

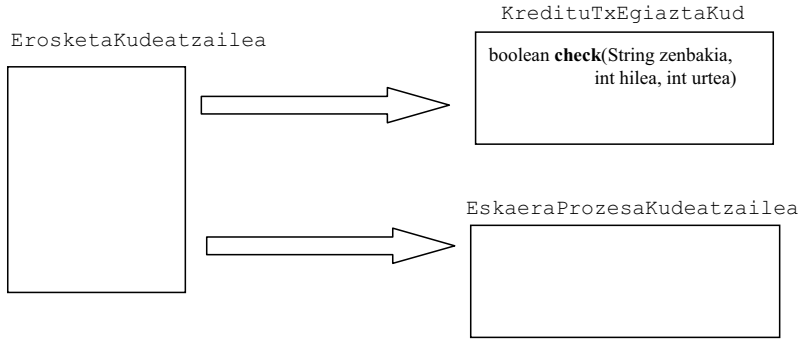
Honako hau eskatzen da:

1. Definitu osagai berriaren interfazeak, bakoitzaren metodoak adierazita.
2. Inplementatu `Akziodun` osagaia.

oOo

Salerosketa elektronikoak kudeatuko dituen sistema informatiko bat eraiki nahi da. Horretarako, hiru osagai diseinatu dira eta EJB teknologia erabilita inplementatu nahi dira. Osagaiak honako hauek dira: `ErosketaKudeatzailea`, `KredituTxEgiaztaKud` eta `EskaeraProzesaKudeatzailea`.

Hasieran, elementuak gehitzen ditu salerosketa elektronikoaren bezeroak bere saskian. Artikuluak aukeratu ondoren, bezeroaren kreditu-txartela egiaztatzen da eta bukatzeko, eskaera prozesatzen da kreditu-txartelaren egiaztapena ondo egin bada.



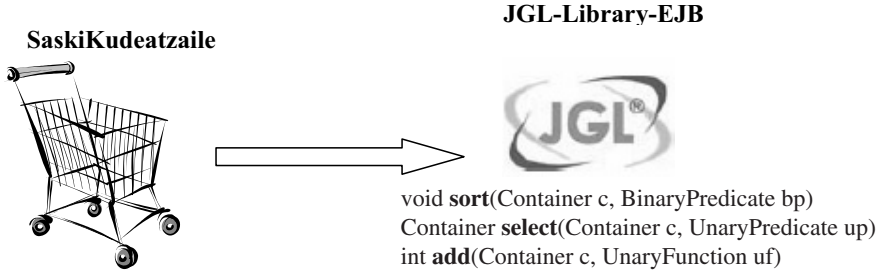
7.15. irudia. Osagaien arteko erlazio funtzionala.

Honako hau eskatzen da: diseinatu eta inplementatu `ErosketaKudeatzailea` osagaia. Kontu egin ezen osagai horrek beste bi osagaien baliabideak erabiliko dituela bere eragiketak gauzatzeko. Eragiketak honako hauek izango dira:

1. Eraiki saskia kudeatuko duten objektuak.
2. Gehitu elementuak (deskripzioa, kopurua, salneurria) saskiari.
3. Txosten bat inprimatu, aukeratu diren artikulu guztiekin deskripzioaren arabera sailkatuta.
4. Eskaera gauzatu txartelaren datuen arabera. Eragiketa honetan txartelaren datuak egokiak diren ala ez egiaztatuko da, eta arazorik ez badago, eskaera prozesatuko da.

oOo

Erosketa elektronikoak kudeatzeko aplikazio informatiko bat diseinatu eta programatzen denean kontuan izaten da nahitaez artikuluen kudeaketa nola gauzatzen den artikuluak saskian jartzen direnean. Diseinatu eta programatu nahi den aplikazioan `SaskiKudeatzaile` osagaiak kudeatuko ditu eragiketa horiek. Eragiketak gauzatzeko, beste EJB osagai batzuk erabiliko ditu aipatutako osagai horrek, artikuluak kudeatzeko. Gure kasuan EJB osagai bat diseinatu eta programatuko dugu aipatutako eragiketak gauzatzeko. Osagaiak ahalmena du JGL liburutegiko baliabideak eskuratzeko.



7.16. irudia. Osagaiak erabiltzen dituen JGL baliabideak.

Honako hau eskatzen da: diseinatu eta programatu `SaskiKudeatzaile` osagaia, JGL osagaia erabiltzen duena eragiketak gauzatzeko. Zehatz-mehatz, aplikazioak honako hau egikaritu beharko luke:

1. Saski motako objektuak eraiki.
2. Artikuluak (`deskripzioa`, `kopurua` eta `salneurria`) gehitu saskira.
3. Txosten bat inprimatu saskiko artikuluekin deskripzioaren arabera sailkatuta. Saskitik txostenera pasatzeko artikuluak, honako baldintza hau bete behar du: bere salneurria `s` baino handiagoa izan behar du.

oOo

7.6. ARIKETAK: DISEINU-PATROIAK

7.17. irudiko pantailek elkarrenganako lotura estuak dituzte. **`AUKERAK`** pantailan balio bat aukeratzen dugunean **`ARTIKULU/BEZERO`** pantaila eguneratu egiten da aukeratutako artikulu edo bezeroaren informazioa azalduz. Aldi berean, **`ESKAERAK`** pantaila ere eguneratu egiten da eta artikulu edo bezeroaren eskaerak agertzen dira zerrenda batean. Aukera berria egiten dugunean **`AUKERAK`** pantailan, eguneratu egiten dira beste bi pantailak.

Diseinatu eta programatu goian azaltzen den erabilera-kasua



7.17. irudia. Ariketaren hiru pantailak.

oOo

Aplikazio informatiko bat garatu nahi da EuskalEnkante erakundearentzat enkantean jartzen diren artikuluak kudeatuko dituena. Enkante bakoitzak honako informazio hau dauka: artikuluaren deskribapena, hasierako salneurria, une horretan artikuluak duen salneurria, salneurri onena eskaini duen erabiltzailearen izena eta artikuluaren jabearen izena.

Gauzatu nahi diren eragiketak honako hauek dira:

1. Eraiki enkante bat, goian adierazitako informazioa jasoko duena.
2. Enkantean parte hartu, eskaintza handiagoa eginez. Horretarako, honako hau zehazten da: zein den erabiltzailea eta zein den enkantearen salneurri berria. Salneurria handiagoa izango denez, enkantearen informazioa eguneratuta geldituko da.

Enkantearen prozesua ondo egikaritzeko, enkante batek eguneraketa bat izaten duenean, eguneraketaren berri, enkante horretan parte hartu duten erabiltzaile guztiei bidaliko zaie ohartaraziz enkantearen salneurri berria.

Honako hau eskatzen da:

1. Diseinatu eta programatu UMLren Klase Diagrama egokia enuntziatuak adierazten duen informazio guztia jasoko duena.
2. Enkantearen garapenaren berri emateko, adierazi egiten dio erabiltzaileak EuskalEnkante erakundeari zein komunikazio-kanaletatik (posta elektronikoa, telefono mugikorra, bilatzailea...) jaso nahi dituen enkantearen garapenari buruzko berriak. Sistemak prestatuta egon behar du etorkizunean komunikazio-kanal berriak kudeatzeko.

   Atal honetan azaldu diren betekizun berriek hasierako Klase Diagramaren diseinua aldatuko dute. Adierazi aldaketa horiek eta inplementatu `Erabiltzailea` eta `Enkantea` klase berriak.
3. Zer aldatuko zenioke Klase Diagramari, erabiltzaileak jasotzen duen mezua enkantearen salneurri berria adierazteko, hizkuntza guztietan jasotzea nahi bagenu? Adibidez, honako mezu hauek sortuko lituzke aplikazio informatikoak:

   Euskara: `Enkantearen salneurri berria honako hau da: XXXX`
   Ingelesa: `The new price of the auction is: XXXX`
   Frantsesa: `Le nouveau prix de la vente aux enchères est XXXX`
   Gaztelania: `El nuevo precio de la subasta es: XXXX`

oOo

Gure herriko jai-batzordeak aplikazio informatiko bat diseinatzeko eta inplementatzeko eskatu digu. Jai-batzordeak jaietan herrian antolatzen diren ekitaldien pertsonala eta pertsonal horren ezaugarriak kudeatu nahi ditu.

Udalak pertsonala hiru motatan sailkatu du: langile arruntak, ekitaldien arduradunak eta herriko auzoetako arduradunak. Auzoko arduradunak bere menpe ditu ekitaldi-arduradunak eta langileak, eta ekitaldi-arduradunak, langile arruntak.

Diseinatu eta inplementatu nahi den aplikazio informatikoa etorkizunean osatu eta jaietako beste arlo batzuetara hedatu nahi denez, erabat modularra eta berrerabilgarria izan behar du eta horretarako ahal den neurrian diseinu-patroi ezagun bat erabiliko da pertsonala hierarkikoki sailkatzeko. Etorkizunean, gainera, pertsonal mota berriak itsatsiko dira hierarkian. Oraingoz, aplikazioak, pertsona bakoitzaren menpe dagoen pertsonala bistaratu nahi du bere kategoria adieraziz.

Jakinda:

- Jaien egitarauak ikuskizun (eguna, tokia, izena) desberdinak dituela.
- `Print.print(Enumeration e)` metodoak `Enumeration` edukiontzian biltegiratuta dauden objektu guztiak inprimatzen dituela.

Zein patroi erabiliko zenuke jaien egitaraua inprimatzeko goian aipatzen den metodoa erabilita? Ebazpena lortzeko bi egoera aztertu beharko dituzu:

a. Lehenengo egoera honetan, atzipena duzu `JaienEgitaraua` klasera eta iturburu-kodea alda dezakezu.
b. Bigarren egoeran, ezin da `JaienEgitaraua` klasea aldatu ebazpena lortzeko.

Diseinatu eta inplementatu egoera bakoitzak behar duen diseinu-patroi ezaguna.

oOo

Aplikazio informatiko bat eraiki nahi dugu enpresa baten eskaera nazionalak eta nazioartekoak kudeatzeko. Eskaerak zatika banatzen dira eta, hala, eskaera bakoitzak entregak izango ditu, bakoitza data batean. Entrega bakoitzean gorde eta kudeatu behar duzun informazioa egituratzeko balioko du `Zatia` klaseak:

```
class Zatia {
 String eskaKod, artKod, data;
 int kopurua;

 public Zatia(String eskaKod, String artKod,
              int kopurua, String data){
 this.eskaKod= eskaKod;
 this.artKod= artKod;
 this.kopurua= kopurua;
 this.data= data;
```

```
}
public void ikuskatu() {
System.out.println();
System.out.print("Eskaera kodea: "+eskaKod);
System.out.print("   Artikulu kodea: "+artKod);
System.out.print("   Artikulu kopurua: "+kopurua);
System.out.print("   Entrega data: "+data);
          }}
```

Eskaera nazionalek eta nazioartekoek izango duten portaera erabat desberdina izango da. Adibidez, garraioa antolatzeko orduan, batek, `GarraioNazionala` klasearen baliabideak erabiliko ditu, eta besteak, `NazioartekoGarraioa`renak. Era berean, bai batak eta bai besteak, entregak eskuratzeko, `eskuratuEntregak()` metodoa erabiliko dute baina batek (eskaera nazionalak) zatiak `array` batean biltegiratuko ditu, eta besteak, `Vector` erako objektu batean.

Programa nagusiak honako lau eragiketa hauek gauzatu behar ditu:

1. Bi eskaera sortu, bata nazionala eta bestea nazioartekoa.
2. Entrega desberdinak gehitu eskaera bakoitzari, `gehituBalioa()` metodoa erabilita:

```
eskaera1.gehituBalioa("1100","1111",5,"050212");
eskaera1.gehituBalioa("1100","1122",7,"050312");
eskaera1.gehituBalioa("1100","1133",8,"050425");

eskaera2.gehituBalioa("2200","2211",4,"050212");
eskaera2.gehituBalioa("2200","2222",6,"050312");
eskaera2.gehituBalioa("2200","2233",10,"050425");
```

3. Eskaeren garraioa antolatu.
4. Eskaeren entregak eskuratu, `eskuratuEntregak()` metodoarekin eta irteera estandarretik atera. Gogoratu ezen entregak, eskaera mota bakoitzean, egitura desberdinetan biltegiratzen direla eta ezaugarri horrek gardena izan behar duela aplikazioarentzat.

Aplikazioa egikaritu ondoren, honako irteera hau ikusi behar du erabiltzaileak irteera estandarretik:

```
Eskaera Nazionalaren garraioa antolatuta

Nazioarteko eskaeraren garraioa antolatuta

Eskaera Nazionalak:

Eskaera-kodea: 1100   Artikulu-kodea: 1111   Artikulu kopurua: 5   Entrega-data: 050212

Eskaera-kodea: 1100   Artikulu-kodea: 1122   Artikulu kopurua: 7   Entrega-data: 050312

Eskaera-kodea: 1100   Artikulu-kodea: 1133   Artikulu kopurua: 8   Entrega-data: 050425
```

```
Nazioarteko Eskaerak:
Eskaera-kodea: 2200   Artikulu-kodea: 2211   Artikulu kopurua: 4   Entrega-data: 050212
Eskaera-kodea: 2200   Artikulu-kodea: 2222   Artikulu kopurua: 6   Entrega-data: 050312
Eskaera-kodea: 2200   Artikulu-kodea: 2233   Artikulu kopurua: 10  Entrega-data: 050425
```

Honako hau eskatzen da:

1. Aplikazioaren betekizunak aztertuta, zein edo zeintzuk diseinu-patroi erabiliko zenuke/zenituzke aplikazioa programatzeko?
2. Marraztu aplikazioaren Klase Diagrama nagusia eta adierazi klase/interfaze bakoitzaren metodoen signatura.
3. Programatu aplikazio informatikoa aipatutako betekizunak inplementatzeko.

oOo

Hezkuntza Sailak, bere egoitza nagusian gauzatuko diren bilerak antolatzeko, aplikazio informatiko bat eraiki nahi du. Hezkuntza Sailak, Lurralde Ordezkaritzetako irakasleekin egiten ditu bilerak, eta horretarako honako prozesu hau erabiltzen du:

- Lurralde Ordezkaritza batek bilera bat antolatu nahi duenean, honako informazio hau zehaztu behar du: noiz egin nahi duen bilera eta zein irakasle mota eta zenbat bidali behar dituen Lurralde Ordezkaritza bakoitzak egoitza nagusira.

  Hezkuntza Sailaren betebeharrak kudeatzeko `HezkuntzaSaila` klasea izango du aplikazio informatikoak. Lurralde Ordezkaritza bakoitzaren kudeaketa `LurraldeOrdezkaritza` klaseak gauzatuko du.

  Lurralde Ordezkaritza batetik bilera-eskaera egin nahi denean, honako mezu hau bidaliko dio `HezkuntzaSaila`-ri `LurraldeOrdezkaritza` klaseak:

```
boolean BileraEskaera(String Nor, Date Noiz, String IrakasleMota, int Zenbat)
```

Mezua jasoko du `HezkuntzaSaila` motako objektuak eta `LurraldeOrdezkaritza` motako objektu guztiei beste honako mezu hau bidaliko die:

```
Vector IrakasleEskaera(Date Noiz, String IrakasleMota, int Zenbat)
```

Modulu berri bat erabiltzen du `LurraldeOrdezkaritza` motako objektuak irakasleengana atzitzeko. Lan horretarako, moduluak, ez dizkio bere inplementazioaren zehaztasunak ematen bezeroari, hau da, objektuen bildumara bere barne-errepresentazioa erabili gabe atzitzen da.

Gauzak ondo badoaz, `LurraldeOrdezkaritza` motako objektu bakoitzak, bektore bat itzultzen du adieraziz zeintzuk irakasle joango diren bilerara.

Honako hau eskatzen da:

1. Aplikazioaren betekizunak aztertuta, zein edo zeintzuk diseinu-patroi erabiliko zenuke/zenituzke aplikazioa programatzeko?
2. Marraztu aplikazioaren Klase Diagrama nagusia eta adierazi klase/interfaze bakoitzaren metodoen signatura.
3. Programatu aplikazio informatikoa aipatutako betekizunak inplementatzeko.

oOo

Liburutegi bateko aldizkariak kudeatuko dituen aplikazio informatiko bat diseinatu eta programatu nahi dugu. Betekizunen azterketa egin du analistak eta erabaki du aldizkari probintzialen informazioa serializatuta biltegiratuko duela. Aldiz, aldizkari nazionalak asko direnez, datu-base erlazional batean biltegiratuko dira. Honako hau eskatzen da:

- Funtzionalitate berri bat programatu nahi da `Liburutegia` klasean, `Aldizkariak` motako egitura bat itzultzen duena. Egitura horretan dauden aldizkariek baldintza bat beteko dute: urteko harpidetzak 100 € baino gehiago balio izatea. `Aldizkariak` egitura, edukiontzi ezezagun bat erabiltzeko prestatuko da.
- `Aldizkariak` egitura aztertu nahi da honako baldintza hau betetzen duten aldizkariak aukeratzeko eta idazteko: aldizkaria formatu elektronikoan ere eskura daiteke.

oOo

UPV-EHUk Gipuzkoako Campusean dituen foroak (musika, kirolak, antzerkia...) kudeatu nahi dira. Horretarako, aplikazio informatiko bat diseinatu eta programatu nahi da, partaideek bidaltzen dituzten mezuak automatikoki kudeatzeko. Honako hauek izan daitezke partaideak: irakasleak, ikasleak, bekadunak eta AZPko langileak. Partaide mota bakoitzak bere atributu eta funtzionalitateak izango ditu. Mezu bat foro batera bidaltzen denean, sistema informatikoak automatikoki informatzen die foroko beste partaide guztiei.

Foroak kudeatzeko eraiki dugun egituran beste funtzionalitate bat txertatu nahi dugu irakasleen irakasgaien tutoretzak kudeatzeko. Horretarako, irakasleak irakasgaian gauzatu behar dituen tutoretzak, atal eta azpiataletan antolatuko dira. Horrela, ikasle batek tutoretza-eskaera elektronikoki bidaltzen duenean, sistema informatikoak, bilatu eta automatikoki eraikitzen du atalari/azpiatalari dagokion ariketa teorikoa edota praktikoa.

Hona hemen adibide bat: X ikasleak tutoretza-eskaera bidali dio Y irakasleari laguntza eske Z irakasgaian honako hau jarrita, `tutoretza-eskaera=2.3.1`. Hau da, irakasgaiaren 2. ikasgaiko 3. ataleko lehenengo azpiataleko laguntza behar du ikasleak.

Honako hau eskatzen da:

1. Aplikazioaren Klase Diagrama eraiki eta klase bakoitzaren metodoak definitu.
2. Diseinatutako klaseak programatu Campuseko foroak eta tutoretzak kudeatzeko.

oOo

7.7. ARIKETAK: UML (UNIFIED MODELING LANGUAGE)

Aztertu nahi dugun informazio-sistemaren erabilera-kasuak honako ezaugarri hauek ditu:

- Gure aplikazioak Euskaltelek kontratuei esleitutako telefono-matxuren eskaerak kudeatu nahi ditu.
- Matxurako, honako informazio hau gorde nahi da: zenbat egun beharko dituen matxura konpontzeko eta egun bakoitzeko, zenbat langile behar dituen. Halaber, egun bakoitzeko, langileak matxurak konpontzeko beharko dituen materialak/tresnak/aparatuak kontrolatu nahi dira. Egin kontu ezen aparatuak erreserbatuta egon daitezkeela egun horretarako eta ordezkoak bilatu egin beharko direla. Gauza bera gertatuko da langile motarekin, hau da, eskatzen den langile mota bat ezin denean erreserbatu egun horretarako, ordezkoa bilatuko da. Beraz, aparatu eta langile moten ordezkoak ere, biltegiratuta eduki beharko ditugu.
- Urteko egun bakoitzeko, langile bakoitzaren lana biltegiratuta egongo da adieraziz egun horretan libre dagoen ala ez. Urtean zehar langileek ez dituzte bakarrik matxurak konpontzen.

Honako erabilera-kasu hau garatu nahi da: `Langileak eta materialak esleitu matxurei`. Horretarako, Euskalteleko administrariak aste-bukaeran, menu nagusian, ***`Matxurak_Esleitu`*** botoia zapalduko du. Sistemak, matxurako eta egun bakoitzeko, aztertu egiten du langileak eta materialak esleituta dauden ala ez. Esleituta ez daudenentzat, langileak edota materialak aukeratzen ditu eta, egun bakoitzeko, beharrezkoak eta ahal diren esleipenak egiten ditu.

Hurrengo egunean, langile bakoitzak badaki zein matxura konpondu behar duen eta zein materialekin.

Hauek dira, gaur egun, Euskaltelek erabiltzen dituen agiriak, matxurak eta iradokizunak jasotzeko:

Matxurak

EUSKALTELek aukera ematen die bezeroei zerbitzuak izan ditzakeen matxurak azkar konpontzeko, Euskaltelen posta elektronikoaren bidez.

-> Zure matxura zein produkturi buruzkoa da?

-> Zu zara...

Partikular bat Enpresa bat

-> Lehen abizena edo sozietatearen izena

-> NAN / IFK / IFZ

-> Linearen telefono-zenbakia

-> Harremanetarako telefonoa

-> Posta elektronikoa

-> Gertaera:

Iradokizunak

Euskaltelek gutunontzi bat jartzen du bezeroen eskura, Euskaltelek eskaintzen dituen ekoizkinen gainean bezeroek dituzten iradokizunak, iruzkinak eta zalantzak jasotzeko.

-> Zure iradokizuna zein produkturi buruzkoa da?

-> Zu zara...

Partikular bat Enpresa bat

-> Lehen abizena edo sozietatearen izena

-> NAN / IFK / IFZ

-> Harremanetarako telefonoa

-> Posta elektronikoa

-> Idatzi hemen zure iradokizuna

oOo

FIFAk antolatzen duen World Cup 2006 (2006ko Munduko Kopa) abian dago. Txapelketaren jarraipen zehatza egiteko, EuskalBerriak komunikabide ezagunak aplikazio informatiko bat erosi nahi du txapelketaren ezaugarriak eta gertaerak ezagutzeko eta kudeatzeko. Hori dela-eta, zu kontratatu zaitu aplikazioa diseinatzeko eta inplementatzeko.

Komunikabideko arduradunarekin bildu eta elkarrizketatu ondoren, honako betekizunak jaso dira aplikazioa eraikitzeko:

- *Zozketak*. Aplikazioak gauza izan beharko du mota honetako galderei erantzuteko:
  - C taldeko lehen partida non eta noiz jokatzen da?
  - F taldeko txapeldunak noren aurka jokatu behar du final-zortzirenetan?
- *Taldeak*. Aplikazioak gauza izan beharko du mota honetako galderei erantzuteko:
  - Nortzuek jokatzen dute inaugurazio-partida?
  - Nortzuk dira B taldeko partaideak?
- *Partidak*. Aplikazioak gauza izan beharko du mota honetako galderei erantzuteko:
  - Nork irabazi zuen B taldeko hirugarren partida?
  - Nork sartu zuen 3. partidako gola?
- *Jokalariak*. Aplikazioak gauza izan beharko du mota honetako galderei erantzuteko:
  - Nork jokatzen du Brasileko selekzioan?
  - Zein jokalarik du 18 urte edo gutxiago?

Galdera orokor horiek aztertzeko eta txapelketaren beste ezaugarri batzuk ezagutzeko FIFAk duen web gune ofizialarekin (`fifaworldcup.com`) jarri gara harremanetan aplikazioaren betekizun berriak eskuratzeko. Eta honako hau aurkitu dugu:

1.- FIFAk txapelketa hasi baino lehen, etiketa batzuk sortu ditu, selekzioak identifikatzeko. Horrela etiketa batzuek zuzen-zuzenean identifikatzen dute talde bat, adibidez, ENG etiketak Ingalaterrako selekzioa identifikatzen du. Baina badaude beste batzuk, adibidez, A1 edo W61, final-zortzirenak, final-laurdenak, finalaurrekoak eta finala, adierazteko. Horrela, A1ek adierazten du A taldean lehenengo gelditu den selekzioa; eta W61ek nork irabazi duen 61. partida. Finaleko partida, W61en eta W62ren artean jokatuko da.

2.- Partida bakoitzak zenbakizko identifikadorea du. Hala, inaugurazio-partidak 1 izango du eta finalak 64. Partidak, halaber, beste honako informazio hau izango du biltegiratuta: data eta hasiera-ordua, hiria eta zein fasetan jokatzen den (zortzirenak, laurdenak...). Biltegiratu nahi dira ere, jakina, partidaren golak.

3.- Selekzio bakoitza hiru letrarekin identifikatuko da (adibidez, ENG Ingalaterrarentzat edo GER Alemaniarentzat). Selekzioaren izena eta entrenatzailearen izena ere biltegiratuko dira.

4.- Jokalariaren informazio pertsonala eta zein taldetatik datorren biltegiratuta edukiko dira.

5.- Partiden kronika idazteko eta gol-egileen sailkapena lortzeko biltegiratuta eduki nahi da partida bakoitzean nork sartu duen gol bat eta zein minututan izan den. Kronika idaztean partidan izandako jokalari-aldaketak jaso nahi direnez, informazio hori ere biltegiratuta eduki nahi da.

6.- Jokalari batek hainbat postutan joka dezake eta partidaren arabera postu zehatz baterako aukeratuko du entrenatzaileak. Biltegiratuta eduki nahi da non jokatu duen partida bakoitzean.

Honako hau eskatzen da:

Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Halaber, itsatsi deskribapen txiki bat bakoitzari.

oOo

1.- Aurreko galderan azaldu den domeinuarekin jarraituz, Erabilera Kasu Diagrama eraiki nahi dugu FIFAren FAIRPLAY (joko garbia) Batzordearen arautegian oinarrituta. Aipatuko diren inprimakiak aplikazio informatiko bat eraikita beteko dira.

2.- Partida bakoitzean, antidopin kontrola egongo da, eta, horretarako, zozketan aukeratutako 8 jokalarien izenak sartuko ditu ikuskatzaileak sisteman.

3.- Partida hasi baino lehen, ***Sendagaien Aitorpena*** inprimakia bete behar du talde bakoitzeko sendagileak adieraziz: sendagaiaren izena, zergatik hartzen duen jokalariak, dosia eta noiz eta zenbat denboran egon behar duen hartzen. Biltegiratuta gelditukо dira informazio horiek eta edozein unetan kontsulta ditzake FIFAko ikuskatzaileak.

4.- Kontsulta horretan, ***Sendagaien Aitorpenean*** agertzen den sendagairen batek debekatuta dagoen substantzia bat badu bere osagaien artean, ikuskatzaileak eskubidea du jokalaria zigortzeko. Zigorrak, partida batean txartel gorriak duen eragin bera izango du, hau da, jokalariak ezin izango du mundialeko partida gehiago jokatu.

5.- Bai ikuskatzaileek baita selekzioetako medikuek ere, FIFAko sendagaien datu-basea kontsulta dezakete sendagaien informazioa eskuratzeko.

6.- Datu-basean dauden sendagaiak zeintzuk diren ezagutzen du ikuskatzaileak gehienetan, baina, eguneraketak egiten direnez, kontsultak gauzatzen ditu. Halaber, behartuta dago, ***Sendagaien Aitorpena*** kontsultatzen duenean, sendagai debekatuen zerrenda egiaztatzen. Batzuetan, gaur debekatuta ez dagoen sendagai bat debekatuta egon baitaiteke bihar.

7.- FAIPLAY (joko garbia) Batzordeak sisteman sartzen du jokalari zigortuari buruzko informazioa, jokalaria kaleratu dutenean partida batean txartel gorria aterata. Txartel gorria ateratzen bazaio jokalari bati, jokalari horrek ezin izango du partida gehiago jokatu. Batzuetan, jokalariak antidopin kontrola pasatu beharko du partidako beste jokalariekin.

8.- Batzuetan jokalari batek zigortuta dagoen sendagai bat hartu behar du hala agindu diolako sendagileak. Horrelako egoeretan taldeko sendagileak edo taldearen ordezkariak ETS (Erabilera Terapeutikoetarako Salbuespenak) inprimakia bete behar du, jokalaria ez zigortzeko. FIFAko ikuskatzaileak sisteman sartu diren ETSak aztertzen ditu ezezkoa edo baiezkoa emateko. Horretarako, ETSan agertzen diren sendagaien informazioa kontsultatu egiten du datu-basean.

9.- Antidopin-kontrola egiteko jokalariaren txiza eta odola aztertzen dira. Ateratako laginak, izenpetu egiten ditu jokalariak eta laborategi batera bidaltzen dira. Analisiaren emaitzak ikuskatzaileari ematen zaizkio zuzenean.

10.- Selekzioetako sendagileek FIFAk biltegiratuta dituen jokalarien fitxa medikoak kontsulta ditzakete zer dagoen biltegiratuta aztertzeko. Hala, jakin dezakete, antidopin kontrolik pasatu duen, egungo ETS zein den, aurreko ETSetan zer adierazi duen ikuskatzaileak... Batzuetan, informazio hori kontsultatu egiten dute sendagileek, ***Sendagaien Aitorpena*** bete baino lehen, jokalariaren fitxa medikoan informazio zehatza baitago sendagaiei buruz eta nahikoa da botoi bat zapaltzea informazio hori guztia zuzenean ***Sendagaien Aitorpenera*** pasatzeko.

Honako hau eskatzen da:

Deskribatutako informazio-sistemaren Erabilera Kasu Diagrama. Ez adierazi erabilera-kasuen gertaera-fluxua. Adierazi, erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den.

oOo

Aztarnak daude Osasuntsu anbulatorioko X izpien makinak ez zuela ondo funtzionatu garai zehatz batean. Hori dela-eta, anbulatorioko datu-basea aztertu nahi da honako hipotesi hau baieztatzeko edo ezeztatzeko: detektatu da, garai horretan, X izpiak hartu zituzten pertsonen seme-alaben artean minbizi-kasuak asko ugaldu direla eta egiaztatu nahi da izpi horiek eraginik izan zuten ala ez.

Anbulatorioak erabilera-kasu bat diseinatu eta inplementatu nahi du aipatutako hipotesia aztertzeko.

Beste proba batzuetan berrerabili ahal izateko, pantaila generikoa izango du erabilera-kasuak eta honako informazio hau sartuko dio erabiltzaileak: proba mota eta aztertu nahi diren hasierako eta bukaerako datak. Informazio hori prozesatuta zerrenda bi eraikiko ditu aplikazio informatikoak adierazitako epean aipatutako proba egin zuten gurasoen seme-alabentzat. Zerrenda batean, semeak egongo dira eta, bestean, alabak.

Lehenengo zerrendan ipiniko diren semeek, nahitaez, honako proba mota hauetariko bat izan behar dute eginda: A, B, C edo D. Aldiz, alaben zerrendan honako proba hauetako bat izan behar dute eginda: E, F, G edo H.

Jarraian, erabilera-kasua egikarituko duen sendagile-erabiltzaileak hautaketa bat egingo du adierazteko sistema informatikoari nork gauzatu behar duen berak kontrolatu eta egikarituko duen XXX motako proba zehatza. Hautaketa hori, jakina, eskuratutako bi zerrendak erabilita egingo da. Amaitzeko, bere egutegia kontsultatuz, behin-behineko txanda (Uuuu-Hh-Ee:Oo-Mm ⑧ 2006-07-04:10-30) esleituko die sendagileak banan-banan aukeratutako pertsonei.

Honako hau eskatzen da:

1. Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Halaber, deskribapen txikia itsatsi bakoitzari.
2. Idatzi deskribatu den erabilera-kasuaren gertaera-fluxua.
3. Idatzi modu berean erabilera-kasuaren Sekuentzia Diagrama (bakarrik ebazpen erlazionala eskatzen da). Azaldu, halaber, Klase Diagrama inplementatzeko erabiliko dituzun eredu erlazionalaren taulak.

oOo

1- Hiriko hotel garrantzitsu batek egunero gauzatzen dituen prozesu batzuk mekanizatu nahi ditu. Hala, aplikazio informatiko bat eraiki nahi du prozesu horiek kudeatzeko. Aplikazioaren eginbeharra hiru arlotara mugatuko da: bezeroen kudeaketara, eskaintzen diren zerbitzuetara eta fakturaziora. Informazio hori guztia barra-barra erabiliko du hoteleko arduradunak azterketa estatistikoak egiteko.

2- Bezeroen honako informazio hau biltegiratu nahi da: NANa, izen-abizenak, helbidea eta harremanetarako telefonoa. Hotelera etortzen den pertsona inoiz hoteleko bezeroa izan bada, aplikazioaren erabiltzailea ohartaraziko du sistemak, berriz alta ez emateko bezeroari. Hotelarentzat pertsona bat bezeroa izango da, inoiz gela bat erreserbatu badu.

3- Hoteleko gelak hiru egoeratan egon daitezke: “hartuta”, “erreserbatuta” edo “libre”. Erreserba bat egitean bezeroaren izena eta harremanetarako telefonoa eskatuko dira. Jakina, erreserbak, beste honako informazio hau ere izan beharko du: zenbat gau igaro nahi diren hotelean, iriste-data eta irteera-data. Faktura

bakoitzean gela bakarra fakturatuko da eta bertan gela horri esleitutako gastu guztiak azalduko dira.

4- Erreserba bat, gauzatu baino lehen, anulatu egin daiteke. Egoera horretan, bezeroaren datu pertsonalak ezabatu egingo dira, baldin eta, bezero horrek ez badu erreserbarik izan lehenago. Halaber, hala eskatuta —bere pribatutasuna gorde nahi duelako— bere datu pertsonalak ezabatzeko eska dezake bezeroak. Faktura ordaindu gabeak baditu bezeroak, aplikazioak ez dio utziko ezabaketa egiten.

5- Segurtasunagatik, harrera-gelan dagoen eta susmagarria den pertsona baten identifikazioa eska dezake hoteleko harreragileak, eta egiaztatu aplikazio informatikoarekin hoteleko bezeroa den ala ez.

6- Hoteleko kudeatzaileei gelen honako informazio hau biltegiratzea interesatzen zaie: gelaren zenbakia, azalera, ohe kopurua, ohe osagarririk onartzen duen, edari-altzaririk eta terrazarik ba ote duen.

7- Jakina, gelaren prezioa ere biltegiratuko da. Bezeroak, ohe osagarria ere eskatzen badu gelarako, kanon txiki bat kobratuko dio hotelak bezeroari eta bera izango da edozein gelatarako.

8- Geletan dauden bezeroei zerbitzu batzuk eskaintzen dizkie hotelak, gero fakturan agertuko direnak. Zerbitzuak honelakoak izan daitezke: tindategia, jatetxea, loradenda... Zerbitzu bakoitzak identifikazio bakarra izango du eta deskribapen txikia, halaber, zerbitzuaren produktua kontsumitu den data eta haren kostua. Kontu egin ezen, kostua, zerbitzuaren produktuaren arabera kalkulatzen dela, hau da, jatetxe-zerbitzuko gosari-produktuak kostu bat izango du eta jatetxe-zerbitzuko 10 afari-produktuk beste bat.

9- Bestalde, hoteleko langileen kudeaketa ere informatizatu nahi dute hoteleko arduradunek. Horretarako, honako informazio hau biltegiratu nahi dute: pertsonalaren identifikadorea, langilearen izena, noiz hasi zen lanean hotelean eta haren soldata. Langile hauek aipatutako zerbitzu batera eta bakarrera daude esleituta. Bi salbuespen daude: geletako garbiketaz arduratzen den pertsonala eta harrera-gelan dagoen pertsonala ez dira zerbitzuetara esleitzen. Jakina, informazio hori guztia biltegiratuta eduki nahi da.

10- Jakin eta biltegiratuta eduki nahi da, nor arduratzen den egunero gela bakoitzaren garbiketaz. Jakina, gela bat baino gehiago garbituko ditu langile batek egunero.

11- Bezeroak hotela uzten duenean, eraikiko zaion fakturak honako informazio hau izango du: bezeroaren NANa, helbidea, iriste- eta irteera-datak, erabili dituen zerbitzuen kostua eta gelaren zenbatekoa (gau kopurua x gelaren prezioa gaueko).

12- Hoteleko zerbitzuek berek ere, zerbitzuak erabiltzen dituzte (adibidez, “jatetxe” zerbitzuak “ikuztegi” zerbitzua erabiltzen du) eta zerbitzu horien kostua

biltegiratuta eduki nahi dute hoteleko arduradunek. Zerbitzuen erabilera-kostua egoera horretan finkoa da (adibidez, "ikuztegi" zerbitzuaren kostua beti da bera hoteleko beste edozein zerbitzurentzat). Biltegiratuta eduki nahi da noiz erabili den zerbitzua.

13- Hotela uzten duenean, faktura eskatzen du bezeroak harrera-gelan. Harreragileak, faktura inprimatzeko agindua sartzen duenean sisteman, bezeroaren urteko faktura guztiak bistaratuko dizkio aplikazioak eta, horrela, jakingo du bezeroaren profila nolakoa den (faktura guztiak ordaindu ote ditu? gure hotela maiz erabiltzen du?...). Inprimagailutik ateratzen ditu sistemak faktura ordaindu gabeak. Edozein unetan eska dezake bezeroak urteko fakturen informazioa. Bistaratutako faktura baten xehetasunak eska diezazkioke harreragileak sistemari.

14- Bikote ezkonberri bat etortzen denean hotelera, fax bat jasotzen da hotelean ohartaraziz noiz helduko den eta zein gela erreserbatu duen bikoteak. Hoteleko mandatu-mutilak faxa hartu eta opari bat uzten du bikotearen gelan.

15- Bezeroak Pay-Per-View (PPV) motako telebista-programak kontrata ditzake gelan ikusteko. Telebista-kanalak paketetan daude antolatuta. Adibidez, kirol-PPV pakete bat eskaintzen da baina baita ere, beste bat haurrentzat, beste bat nagusientzat... eta abar. Pakete bakoitzak, kode bat, bere izena eta aukeran eskaintzen dituen filmak izango ditu. Film batzuk pakete guztietan eskaintzen dira. PPV paketeak, soil-soilik aloka daitezke baina halaber, filmak gehitu daitezke hala nahi izanez gero. PPV pakete bat alokatzen denean filmekin edo filmerik gabe, pasahitza ematen dio hotelak bezeroari.

16- PPV pakete bat baino gehiago kontrata ditzake bezero batek. Halaber, aloka dezake beste data batean PPV pakete bera beste film batzuekin.

17- Ez da biltegiratu nahi noiz eta zenbat aldiz erabili den PPV paketea baina bai noiz kontratatu den.

18- Gerta daiteke bezero bat ados ez egotea postaz heldu zaion faktura batekin. Egoera horretan, hotelera deitzen du eta faktura pantailan bistaratzeko eskatzen dio harreragileak sistemari bezeroak esaten diona egiaztatzeko

Honako hau eskatzen da:

Deskribatutako informazio-sistemaren Erabilera Kasu Diagrama. Ez adierazi erabilera-kasuen gertaera-fluxua. Adierazi, erabilera-kasu bakoitza, enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den.

Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Halaber, deskribapen txiki bat itsatsi bakoitzari.

oOo

Hotel batek egun bateko txangoak antolatzen ditu eta gidari bat esleitzen dio txango bakoitzari. Gidari batek egunean gehienez ateraldi bat izango du.

Gidari bat gaixo dagoenean, hotelera deitzen du eta sistema informatikoak honako hau egiten du: aztertu egiten du ea gaixorik dagoen gidariak bihar txangorik duen eta egoera berriari aurre egiten saiatzen da.

Prozesua sinplifikatzeko gidariaren biharko egoera bakarrik landuko da ariketa honetan. Gidariak, bihar ez badu txangorik, prozesua bukatutzat ematen da. Aldiz, gidariak bihar txango bat badu hoteleko arduraduna beste gidari bat esleitzen saiatuko da.

Arduradunak erabiliko duen aplikazio informatikoak honela funtzionatuko du `esleitu gidari berria` erabilera-kasua gauzatzeko: gidari gaixoaren identifikadorea sartzen du arduradunak eta gidari horrek bihar txangorik duen ala ez egiaztatzen du sistemak. Txangoa badu, bihar libre dauden gidarien zerrenda bistaratzen du aplikazioak. Egoera horretan, gidari berri bat aukeratzen du arduradunak eta txangoari esleitzen dio. Kontu egin ezen gidariak, eguneroko lanorduetan, ez daudela txangoak gauzatzen bakarrik. Egon daitezke, halaber, bulegoan administrazio-lanak egiten.

Gerta daiteke sistemak ez aurkitzea biharko gidaririk txangoa egiteko. Egoera berri horretan, data berria bilatu behar zaio txangoari gidaria esleitzeko. Horretarako, datozen 30 egunetarako gidari libreen zerrenda bistaratzen du sistemak eta gidari bat aukeratzen du arduradunak data zehatz baterako. Txangoaren data aldatu egin denez, bezeroei ohartarazi behar zaie, txangoa egunez aldatu dela eta ea txangoa egin nahi duten galdetu.

Txangoaren data berria hoteleko idazkariak kudeatuko du `txangoaren data berria` erabilera-kasuaren bidez. Horretarako, arduradunak telefonoz pasatu dion txangoaren kodea sartuko du sistemara eta izena eman duten bezeroei deituko die jakiteko ea interesatzen zaien txangora joatea data berrian.

Txangoa ez da egingo bezero kopuru minimo bat lortzen ez bada. Txango bakoitzak bere kopuru minimoa izango du.

Bukaeran, baiezkoa eman duten bezeroei, deituko die idazkariak txangoa egingo dela edo anulatu egin dela adierazteko.

Honako hau eskatzen da:

1. Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Halaber, deskribapen txikia itsatsi bakoitzari.
2. Idatzi, `esleitu gidari berria` eta `txangoaren data berria` erabilera-kasuen gertaera-fluxua.

3. Idatzi `Esleitu gidari berria` eta `txangoaren data berria` erabilera-kasuen Sekuentzia Diagrama (bakarrik ebazpen erlazionala eskatzen da). Azaldu, halaber, Klase Diagrama inplementatzeko erabiliko dituzun eredu erlazionalaren taulak.

oOo

1- Guggenheim Museoak informazio-sistema bat eraiki nahi du bere artelanak kudeatzeko. Artelan bakoitzaren egilea eta zein garaitakoa den izango ditu biltegiratuta. Horrez gain, artelan bakoitzerako honako beste informazio hau biltegiratuko da: balio ekonomikoa, sarrera-data, sortze-data eta azken zaharberritzearen data.

2- Artelanen informazio zehatzaren datuen sarrera informazio-sisteman, katalogoen arduradunak gauzatuko du museoaren idazkariak sartutako informazioan oinarrituta. Idazkariak sartzen duen informazioa, artelanen hornitzaileek beraien zerbitzari informatikoak eraikitako txostenak kontsultatuta lortzen du.

3- Artelanen zaharberritze-prozesua ere kudeatu nahi da. Horretarako, edozein unetan zaharberritzaileen arduradunak erabakitzen du artelan batzuk zaharberritu egin behar dituztela hondatuta daudelako. Horretarako, ordenagailua erabilita, artelanaren egoera aztertzen du eta erabakia hartzen du zaharberritzeko. Artelan bat, beraz, zaharberritzen edo ikusgai egongo da museoan.

Bost urtean behin, artelan bakoitza zaharberritu egiten da eta, horregatik, egunero, prozesu bat martxan jartzen da erabakitzeko zein artelan zaharberrituko den.

Artelan bakoitzari aplikatzen zaizkion zaharberritze-eragiketa guztiak biltegiratuta edukitzea interesatzen zaio museoari. Hau da, honako informazio hau eduki nahi du: noiz eta zein zaharberritzailek zaharberritu duten artelana eta zaharberritzaile bakoitzak erabili dituen materialak artelana zaharberritzeko. Halaber, biltegiratuta eduki nahi da artelanaren zaharberritze bakoitzaren kostu nagusia.

4- Aldiko, Guggenheim Museoak, artelan batzuk uzten dizkie beste museo batzuei epealdi baterako. Museo horiek idatziz bidalitako eskaerak, museoko idazkariak sartzen ditu informazio-sisteman. Museoko zuzendariak eman behar du oniritzia artelanak uzteko. Horretarako, museoen eskaerak aztertzen ditu eta ordenagailu bidez adierazten du zeini ematen dion oniritzia. Artelan bat uzten denean, honako informazio hau biltegiratu nahi da: noiz utzi zaion artelana museoari eta noiz arte.

5- Lapurretak ekiditeko, aldiro, aretoz aldatzen dira artelanak. Areto bakoitzari zaintzaileak esleitzen zaizkio eta horretarako honela antolatuko da informazioa ordenagailuaren datu-basean:

1. Areto bakoitzak zaintza motak edukiko ditu.

2. Zaintzaile bat zaintza mota batzuetan bakarrik da aditua eta dago homologatua.
3. Areto batek zaintzaileak izango ditu esleituak baina aretoak onartzen dituen zaintza motakoak onartuko ditu bakarrik.

Aretoen eta zaintzaileen esleipen hori museoko zuzendariak gauzatzen du.

6- Artelan bat zein aretotan dagoen paratuta jakiteko, museoko atarian kokatutako artelanen listatuak kontsulta ditzakete museoko bisitariek.

7- Eraiki nahi den informazio-sistemak kontsulta-zerbitzuak izan behar ditu inplementatuta honako erabiltzaile hauentzat:

1. Zaharberritzaileen arduradunak edozein unetan kontsulta dezake museoko artelanen egoera.
2. Jakin badakigu artelanak beste museoei uzten zaizkienean eskaera bat sortzen dela. Museoko zuzendariak kontsultak gauzatu ditzake eskaera horien edukiak aztertzeko. Kontsulta horiek egiten ari dela eskaera egin duen museoaren profila informatikoki azter dezake zuzendariak. Azken aukera hori, zuzendariak, edozein unetan egikaritu dezake.

Honako hau eskatzen da:

Deskribatutako informazio-sistemaren Erabilera Kasu Diagrama. Ez adierazi erabilera-kasuen gertaera-fluxua. Adierazi erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den.

Marraztu azaldutako informazioaren Klase Diagrama. Adierazi erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak.

oOo

Erabilera-kasu hau aztertu nahi da ariketa honetan: artelan bat esleitu areto berezi bati eta biltegiratu, halaber, zein izango den artelanaren arduraduna areto berezi horretan dagoen bitartean. Erabilera-kasu hau ere, Guggenheim Museoaren testuinguruan kokatuko da.

Guggenheim Museoak erakusketa bat antolatzen duenean garai edo mugimendu zehatz baten inguruan (errenazimentua, kubismoa, espresionismoa...) bere bildumako eta beste museo batzuetako artelanak erabiltzen ditu. Artelana beste museo batekoa denean areto berezietan jartzen da ikusgai. Halakoetan, aldiro, artelana aretoz areto aldatzen da segurtasuna bermatzeko.

Artelan bakoitza mota zehatz batekoa izango da eta areto berezi bakoitzak mota zehatz batzuetako artelanak bakarrik onartzen ditu. Areto berezi bakoitzak

arduradunak ditu esleituak eta arduradun horiek areto berezi batean baino gehiagotan arduradun izan daitezke.

Artelan mota batzuk bakarrik kudea ditzake arduradun bakoitzak. Artelan kopuru zehatza kudeatzen du arduradun batek. Kopuru zehatz hori, arduraduna sortzen denean datu-basean adierazten da eta finkatzen da.

Aipatutako informazio hori guztia jakinda, honako erabilera-kasu hau diseinatu nahi dugu: artelan bat esleitu areto berezi bati eta biltegiratu, zein izango den artelanaren arduraduna areto berezi horretan dagoen bitartean. Kontu egin ezen artelana aurretik beste areto berezi batean egon zitekeela.

Erabilera-kasuak sarrera bakarra izango du: `artelanaren identifikadorea`.

Oharra: Artelan bat, behin egon bada areto berezi batean ez da berriro ikusgai jarriko areto berezi horretan. Artelanari, alta, beste erabilera-kasu batean ematen zaio.

Honako hau eskatzen da:

1. Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Horrez gain, itsatsi deskribapen txikia bakoitzari.
2. Adierazi `Artelanak eta arduradunak esleitu` erabilera-kasuaren Sekuentzia Diagrama (bakarrik ebazpen erlazionala eskatzen da). Adierazi, halaber, Klase Diagrama inplementatzeko erabiliko dituzun eredu erlazionalaren taulak.

oOo

Trafikoko Zuzendaritza Nagusiak (TZN) eta autoeskolen elkarteak informazio-sistema bat eraiki nahi dute, puntu bidezko gidabaimenaren sistema kudeatzeko. Sistemak inplementatu beharko dituen betekizunak honako hauek dira:

1. Lehenengo eta behin, gidabaimenen esleipena kudeatu nahi du. Gidabaimena eskuratzeko bi azterketa mota gainditu ko dira: test erako azterketa teorikoa eta horren ondoko azterketa praktikoa. Azterketa bat (teorikoa edo praktikoa) egin baino lehen, ikasleen datu pertsonalak (NANa, izenabizenak), data eta azterketa mota sartu eta bidaltzen dituzte autoeskolek TZNra. TZNk jasotzen ditu datu horiek eta egiaztapen batzuk egin eta gero, zerrenda bat osatzen du azterketara joango direnekin. Zerrendak honako informazio hau jasoko du: azterketaren data eta ordua, eraikina eta gela (azterketa teorikoetarako) edo abiapuntua (azterketa praktikoetarako). Zerrenda horiek postaz bidaltzen dira autoeskoletara.

2. Azterketara aurkezten denak gidabaimena galdu badu zigor larri bat eginda, autoeskolak bidalitako datuak egiaztatzen dituenean, ziurtatu egin behar du

TZNk pertsona hori ikastordu teorikoetara joan dela azterketara aurkeztu baino lehen.

3. Azterketak egin eta gero, kalifikazioen zerrendaren emaitzak sartzen dituzte zuzentzaileek adieraziz, aurkeztutako bakoitzarentzat, bere NANa, azterketa mota eta emaitza (GAI/EZ GAI). Informazio hori guztia biltzen du TZNk eta, autoeskolen arabera zatitzen du. Ikasleei bidaltzen zaizkie emaitzak. Gidabaimena eskuratu duenak 10 puntuko saldoa edukiko du hasieran.

4. Gidabaimenaren saldoa jaisten joango da gidariak trafikoko arau-hausteak egiten dituen heinean. TZNk hiru arau-hauste mota sortu ditu: oso larria, larria eta arina. Eta honako zigor hauek izango dituzte: 8, 5 eta 2 puntu.

5. Gidariek arau-hausteak egiten dituztenean, PDAk eta inprimagailu eramangarriak erabiltzen ditu Ertzaintzak zigorra komunikatzeko eta aplikatzeko. Hori dela-eta, oharra inprimatzen du ertzain batek eta ibilgailuaren haizetakoari itsatsi. Edozein unetan ere, inprima ditzake ertzainak oharrak.

6. Salbuespen-egoeretan, Ertzaintzak ibilgailua ibilgetu dezake. Egoera horietan TZNa ohartarazten da eta oharra inprimatzen zaio gidariari.

7. Gidariak arau-hauste bat egiten duenean eta Ertzaintzak gelditzen badu, honako informazio hau bidaliko du ertzainak TZNra: gidariaren NANaren zenbakia, arau-hauste mota eta kodea (arau-haustea zehazteko), deskribapena, eta, aukeran, azalpen gehigarriak. Informazio hori guztia jasoko du sistema informatikoak eta bertatik bertara, gidariaren saldoa eguneratzen du eta ertzainari bidaltzen dio informazio eguneratua. Saldoa zero edo txikiagoa bada, gidariari gidabaimena kentzen dio ertzainak eta ibilgailua ibilgetu egiten du lehen aipatu dugun eran. Egoera horretan, gidari ohiak azterketak gainditu behar ditu gidabaimena berriz eskuratzeko. Gidariak arau-haustea egiten duenean eta gidabaimena sistema zaharrekoa bada, arau-haustea biltegiratu baino lehen bere datu-basean sartuko du sistemak eta 10 puntuko (gidabaimenak 10 urte baino gutxiago badu) edo 12 puntuko (gidabaimenak 10 urte baino gehiago badu) saldoa itsatsiko dio. Gidabaimenaren informazio hori TZNren sistema informatiko zaharreko datu-basetik eskuratzen da.

8. Oinarrizko estatistikak atera ditzake TZNk adieraziz zein estatistika mota eskuratu nahi den eta zein datatan. Hori egiteko, aplikazioaren komandolerroa erabiliko da. Halaber, egiazta dezake bete beharreko ikastordu teorikoetara nortzuk joan diren eta eskuratu azterketara aurkeztuko direnen zerrenda.

Honako hau eskatzen da:

Deskribatutako informazio-sistemaren Erabilera Kasu Diagrama. Ez adierazi erabilera-kasuen gertaera-fluxua. Adierazi erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den.

oOo

Azterketa teorikorako autoeskolek ikasleen datu pertsonalak eta data egokia sartzen dituzte sistema informatikoan. Urte hasieran urteko azterketa teorikoak biltegiratzen dira datu-basean adieraziz zein egunetan egingo diren. Azterketa teoriko berean gidabaimen mota batera baino gahiagotara aurkez daiteke ikaslea. Ikasleen datuak sartzen direnean, adierazi egin behar duzu argi eta garbi zenbat gidabaimen motatara aurkeztuko den ikaslea.

Azterketa teorikoa egiteko, gela desberdinak prestatzen dira. Gela bakoitzean gidabaimen mota bakarreko ikasleak egongo dira. Horrela, gidabaimen bat baino gehiago eskuratu nahi baditu ikasleak azterketa berean, gela desberdinetara joan beharko du.

Ikasleak atera duen nota biltegiratuta eduki nahi da azterketa bakoitzerako eta gidabaimen mota bakoitzerako.

Autoeskolako ikasleak gidabaimena ateratzeko, azterketa teorikoa gainditu ondoren, azterketa praktikoa egin behar du ibilgailuarekin. Azterketa bakoitza data zehatz batean egingo da eta gidabaimen mota bat ateratzeko balioko du. Bere ibilgailuak prestatzen ditu autoeskola bakoitzak azterketa baterako eta arazoak badaude ibilgailu batekin, bere ordezkoetako bat aukeratzen da. Ikastordu praktikoetan erabili duen ibilgailu bera erabiliko du ikasleak azterketa praktikoa egiteko. Ez badu gainditzen azterketa, hurrengo txandarako beste ibilgailu batekin aurkez daiteke.

Irakasleen ordutegia sistema informatiko bidez egituratu nahi du autoeskolak eta, horretarako, bere irakasleak izango ditu esleituak ikasle bakoitzak. Edozein unetan sistema informatikoa kontsulta dezakegu jakiteko irakasle bat ordu horretan libre dagoen ala ez, eta zein ikaslerekin dagoen ikastordua (teorikoa edo praktikoa) egiten. Ikastordu praktikoa badu irakasleak, ibilgailuaren ordutegian egongo da esleitua. Irakasle bat gidabaimen mota zehatzak emateko gai da bakarrik.

Honako hau eskatzen da:

Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Horrez gain, deskribapen txikia itsatsi bakoitzari.

oOo

Argitaletxeen argitalpenak banatzen ditu BANANDU enpresa-kooperatibak. Horretarako biltegi nagusia Lezon dauka banaketak egiteko.

1. Kontratuak sinatzen ditu enpresa banatzaileak herrialdeko egunkari eta aldizkariekin eta etxez etxe banatzen dizkie argitalpenak harpidedunei. Enpresa banatzailearen harpidedun batzuk kioskoak izan daitezke.
2. Enpresaren bezero nagusiak, argitalpenen (egunkaria edota aldizkaria) argitaletxeak dira. Argitalpen desberdinak izango ditu argitaletxe bakoitzak banatzeko eta enpresa banatzailea arduratuko da banatzeaz herrialdean zehar.
3. Harpidedunak erakartzeko triptikoak argitaratzen ditu enpresa banatzaileak.
4. Egunkariak, astegunetan bakarrik, asteburuetan bakarrik edota astegunetan eta asteburuetan jasotzen ditu harpidedunak. Aldizkariak, ordea, beti jasotzen dira jarraian. Enpresa banatzaileak kontrolatuta eduki nahi du harpidedun batek argitalpen bat nola eta noiz jaso nahi duen. Harpidedunak urteetan edo hilabeteetan hutsuneak eduki ditzake baja eman diolako argitalpenari.
5. Edozein unetan 4. paragrafoan deskribatutako datuak alda ditzake harpidedunak. Horretarako, telefonoz deituko dio enpresa banatzaileari edo enpresa banatzailera joaten da.
6. Ezer ez bada adierazten harpideduna harpidedun arrunta izango da. Batzuetan, baina, harpidedunari alta ematen zaionean, lehentasunezko harpidedun gisa biltegiratzen da, ezaugarri berezi batzuk adierazita. Edozein unetan, harpidedunak baldintza zehatz batzuk betetzen dituenean, lehentasunezko harpidedunen kategoriara pasatzen du enpresa banatzaileak. Aipatutako bi prozesuak, enpresako kontratazio-sailak gauzatzen ditu.
7. Honako era honetan banatzen ditu egunero enpresa banatzaileak argitalpenak: goizean goiz banatzaile bat bere furgoneta hartu eta enpresa banatzailearen biltegira joaten da. Biltegian, argitalpenak kargatu, eta emate-agiria jasotzen du ordenagailutik. Banaketa, herriz herri egiten da ibilbide jakina erabilita. Ibilbide bat banatzaile bati etengabe esleitzen zaio aldatu arte. Aldaketa egiteko, urte hasieran aplikatzen den prozesu bera egikaritzen da ibilbidea esleitzeko.
8. Egoera zehatz batzuetan, baja ematen zaio banatzaileari. Egoera horietan honako prozesu hau egikaritzen da: banatzaile horrek esleituta zituen ibilbideak beste banatzaile batzuei esleitzen zaizkie.
9. Banaketa egin baino lehen, badaezpada, ibilbidea ohikoa dela egiaztatzen dute banatzaileek ordenagailua erabilita.
10. Harpidedunak askatasuna du zehazteko non jaso nahi duen argitalpen bakoitza. Hala, aldizkariak, etxean jaso ditzake eta egunkariak, lanean. Gainera, aukera ematen dio enpresak, finkatzeko, noiz arte eta non jaso nahi duen argitalpen bakoitza.

11. Aldian-aldian, saldu ezin izan dituzten aldizkariak itzuli egiten dituzte kioskoek eta informazio hori bere datu-basean biltegiratuta eduki nahi du enpresa banatzaileak data eta guzti zehaztuta.
12. Halaber, aldian behin, bai egunkariek eta bai aldizkariek, promozioak egiten dituzte bezeroen leialtasuna ziurtatzeko edota salmentak ugaltzeko. Horretarako, bilduma motak proposatzen dizkie bezeroei: DVD filmak, bideokaseteak edo opariak. Promozio horiek kioskoetan bakarrik egoten dira.
13. Horma-irudiak argitaratzen dituzte argitaletxeek marketina bultzatzeko. Horma-irudi horiek kioskoetara bidaltzen dira bakarrik.

Honako hau eskatzen da:

Deskribatutako informazio-sistemaren Erabilera Kasu Diagrama. Ez adierazi erabilera-kasuen gertaera-fluxua. Adierazi erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den.

Marraztu azaldutako informazioaren Klase Diagrama. Adierazi erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak.

oOo

Demagun Euskaltel enpresaren domeinua aztertzen ari garela. Honako erabilera-kasu hau garatu nahi da: `Promozioak esleitu bezeroei`. Horretarako, Euskalteleko administrariak, menu nagusian, ***`Promozioak_Esleitu`*** botoia zapaldu baino lehen bezeroaren NANa, kontratu motaren ezaugarri bat eta esleitu nahi den promozioa tekleatzen ditu. Jarraian, sistema martxan jartzen da. Honako hau izango da sartutako informazioa tratatzeko prozesua: bezeroaren kontratuak bilatu eta kontratu bakoitzaren motara atzituz mota horren ezaugarriak aztertu. Sartutako ezaugarria ezaugarri horietan badago, promozio berria esleitzen zaio bezeroari.

Kontu egin ezen promozio berria bezero bati baino gehiagori eslei dakiokeela eta kontratu mota baten ezaugarria beste kontratu mota batean ere egon daitekeela.

Honako hau eskatzen da:

Marraztu goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Halaber, itsatsi deskribapen txikia bakoitzari.

Adierazi, `Promozioak esleitu bezeroei` erabilera-kasuaren Sekuentzia Diagrama (bakarrik ebazpen erlazionala eskatzen da). Adierazi, Klase Diagrama inplementatzeko erabiliko dituzun eredu erlazionalaren taulak.

oOo

1. Giharrak kirol-elkarteak honako partaide mota hauek ditu: Pertsonal Saila, Finantza Saila, arduradun bat, diruzain bat, administrari bat, kobratzaileak eta bazkideak.
2. Elkarteko pertsonalaren ordenagailuak sarean daude konektatuta beharrezko informazio guztia transmititzeko.
3. Elkarteko kobratzaileak bezeroarengana joaten dira kuotak kobratzera. Egunero, kontsulta bat egiten du kobratzaileak jakiteko zein bezerorengana joan behar duen egun horretan. Kobratzaileak hiru motatakoak izan daitezke bezeroaren motaren arabera:
   - A motakoek, AA motako bezeroak bisitatzen dituzte, 10 € kobratzeko hilero.
   - B motakoek, BB motako bezeroak bisitatzen dituzte, 15 € kobratzeko hilero.
   - C motakoek, CC motako bezeroak bisitatzen dituzte, 25 € kobratzeko hilero.
4. Kobratzaileek egunero egin behar dituzten bisitak Pertsonal Sailak antolatzen ditu. Bisitak antolatzeko, kobratzaileen agenda kontsultatzen da eta hurrengo eguneko bisitak esleitzen dizkie.
5. Eguneroko bisitak bukatuta, jasotako dirua, kirol-elkartean uzten dute kobratzaileek adieraziz, zein bazkideri kobratu zaion kuota eta zeini ez.
6. Denbora arazoak baditu kobratzaileak eta ezin bada hurbildu kirol-elkartera dirua uztera, beste kobratzaile bati eman diezaioke, berak egin dezan eragiketa. Gerta daiteke bigarren kobratzaileak ez izatea dirurik egun horretan diru-sarrerak egiteko, hau da, ez du bere kuotarik kobratu. Orduan, ordezkatzen duen kobratzailearen diru-sarrera egingo du bakarrik.

   Kobratzailea, egoera horretan badago baina berak ere diru-sarrerak baditu, lehenengo, bereak sartuko ditu eta gero, ordezkatzen duenarenak.

   Beste kobratzaile baten diru-sarrerak kudeatzen dituenean, adierazi egin behar dio kobratzaile batek sistema informatikoari ordezkatzen duen kobratzaileak zergatik ezin izan duen diru-sarrera egin.

   Sisteman biltegiratuta gelditu behar du nork egin duen bisita eta nork esleitu dion bisita sistema informatikoari.
7. Eguna amaitzean jasotako diru kopurua Finantza Sailera bidaltzen du kirol-elkarteko administrariak eskuz. Azken sail horretan, diru kopurua sistema informatikoan biltegiratzen da.
8. Kirol-elkarteko diruzainak bakarrik atera dezake dirua Finantza Sailetik, eragiketa informazio-sisteman gelditukо da adierazita. Halaber, pantaila bidez, kontsulta dezake nahi badu, elkartean dagoen diru kopurua.

9. Kirol-elkarteko diruzainak bakarrik kontsulta dezake nahi badu pantaila bidez, elkartean dagoen diru kopurua.
10. Kirol-elkarteko arduradunak dirua eska diezaioke Finantza Sailari administrazio-gastuak ordaintzeko.
11. Kirol-elkarteko arduradunak proposatzen dio Pertsonal Sailari zein kobratzaileri igo behar zaion maila.
12. Kobratzaile berri bat kontratatu nahi duenean, Finantza Sailari dirua eskatzen dio kirol-elkarteko arduradunak administrazio-gastuak ordaintzeko eta jarraian Pertsonal Sailari eskatzen dio gauzatzeko kontratazioa.
13. Kirol-elkarteko arduradunak kobratzaile bat kaleratu nahi duenean Pertsonal Sailari eskatzen dio gauzatzeko kaleratzea.
14. Pertsonal Sailak egiten du kontratazioa.
15. Kontratazio-prozesuan, kobratzailearen alta, inplizituki gauzatzen da. Era berean, kaleratze-prozesuan, kobratzailearen baja, inplizituki gauzatzen da.
16. Ordainketa-egunean, kobratzaileei ordaindu beharreko nominak inprimatzen ditu kirol-elkarteko diruzainak eta eskuz egiten ditu ordainketak.
17. Nominak inprimatu ondoren, ordaindu behar den diru kopurua automatikoki deskontatzen da Finantza Sailean dagoen diru kopurutik. Pentsatzekoa da beti egongo dela dirua nominak ordaintzeko.

Kontsultak, eskaerak eta esleipen guztiak ordenagailua erabilita egiten dira.

Honako hau eskatzen da:

Deskribatutako informazio-sistemaren Erabilera Kasu Diagrama. Ez adierazi erabilera-kasuen gertaera-fluxua. Adierazi erabilera-kasu bakoitza enuntziatuko hitzak erabilita eta zein paragrafotatik lortu den.

oOo

7.8. ORDENAGAILUAREKIN EGITEKO LABORATEGIAK

Laborategi honetan Javaren klaseekin eta objektuekin lan egingo dugu. Horretarako, aplikazio garatu bat aztertuko dugu eta funtzionalitate berriak itsatsiko dizkiogu oinarrizko iturburu-kodeari.

Oinarrizko iturburu-kodea eta `txt` motako fitxategiak, honako helbide honetatik eskuratu ditzakezu:

`http://ji.ehu.es/isw/Web_ISO.html`

Iturburu-kodea probatzeko, `txt` motako fitxategiak erabiliko dira. Fitxategi horiekin aplikazioaren hasierako informazioa eraikiko dugu.

Aplikazioak bezeroaren eskaerak kudeatuko ditu eta horretarako hainbat klase eratuko ditu eskaera horiek gauzatzeko.

Datuen biltegiratzea, goian aipatu dugun bezala, `txt` motako fitxategiak erabilita eratzen denez, klase berezi bat (`Bezeroak`) erabiliko dugu datu-basearen funtzioak beteko dituena.

Gauzak horrela, oinarrizko iturburu-kodea erabilita, honako hiru lan hauek gauzatu behar dituzu laborategian:

1. Marraztu aplikazio informatikoaren Klase Diagrama.
2. Osatu oinarrizko iturburu-kodean agertzen diren hutsuneak eta probatu aplikazio informatikoa, hots, egiaztatu zuk idatzitako iturburu-kodea ondo integratzen dela aplikazio informatikoan.
3. Orain dagoen bezala antolatuta aplikazioa, eskaera bat egiten denean, ez da kontuan hartzen eskatzen den artikulu kopurua zerbitza daitekeen ala ez (**Hautazkoa**).

   Aplikazioa egokitu egingo dugu eta bertsio berrian kontuan hartuko dugu kopuru hori. Horrela, eskatzen den kopurua handiagoa bada biltegiratuta duguna baino, eskaera bat egingo zaio hornitzaileari arazoa konpontzeko. Hornitzaileko, eskaera bakarra eraikiko da.

Bi bertsio diseinatzea eta programatzea proposatzen da:

a. Artikuluak hornitzaile bakarra du eta hari eskatzen zaio produktua.

b. Artikuluak hornitzaile bat baino gehiago ditu eta fidagarrienari eskatzen zaio artikulua.

Praktika gauzatzeko kontuan hartu honako ohar hauek:

1.- `Vector` motako objektu batetik elementu bat ezabatu nahi bada, honako metodo hau egikaritu behar duzu:

```
removeElement(Object obj)
```

2.- Konparazioak gauzatzeko erabili : `equals()` metodoa aldagaientzat eta == adierazlea objektuentzat.

3.- `lortuEskaera()` eta `lortuEskaerak()` metodoetan eskaeraren hartzailea aztertu behar duzu parametroak hala eskatzen duenean. Arazoak daudenean `null` erreferentzia itzuliko da.

4.- Kontu egin ezen bezeroaren `eskaerak` array-a, `null` erreferentziatzen egon daitekeela.

5.- Hornitzaileari egingo zaion eskaeran artikuluaren kopurua eskuratzeko, bezeroaren eskaeraren kodea eta artikuluarena adieraziko dira. Horrela, beti izango dugu kontrolatuta, artikulu kopuru bat heltzen zaigunean, zein eskaerari lotu behar diogun. Kontu egin ezen bezeroaren eskaerari aldagai berri bat sartu behar diozula hura identifikatzeko.

6.- Hornitzailearen eskaerak, `Hornitzailea` motako objektuan biltegiratuko dira.

oOo

Bigarren laborategi hau lehenengoaren bilakaera naturala izango da. Gehiegi ez aldrebesteko laborategiaren joan-etorria, aurreko laborategian erabilitako domeinu bera erabiliko dugu bigarren honetan ere. Gogoratu, aplikazioak, bezeroaren eskaerak kudeatzen zituela eta horretarako hainbat klase eratzen zirela eskaera horiek kudeatzeko.

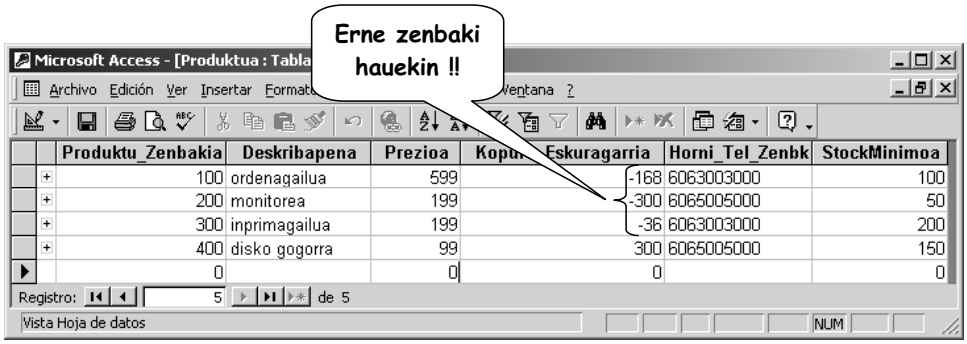
Aurreko laborategian datuen biltegiratzea `txt` motako fitxategiak erabilita eratzen zenez, klase berezi bat (`Bezeroak`) erabiltzen zen datu-basearen funtzioak betetzen zituena. Orain, laborategi honetan, informazioaren biltegiratzea datu-base batean (`eskaera2000.mdb`) egingo da Javaren JDBC baliabidea erabilita. Gauzak horrela, inplementatu nahi dugun aplikazioak **7.20. irudia**n agertzen diren erlazioak erabiliko ditu domeinuaren informazio iraunkorra kudeatzeko. Bestalde, aplikazioak, **7.19. irudia**n agertzen den interfaze grafikoa erabiliko du informazioa bistaratzeko eta harremanetan egoteko erabiltzailearekin.

Hutsetik ez hasteko, aplikazioaren irteera grafikoaren funtzionalitate gehiena inplementatuta dago eta **7.21. irudia**n agertzen den Klase Diagramari jarraitzen dio orpoz orpo. Kontu egin ezen klase bakoitzak bere funtzioa betetzen duela aplikazioaren arkitektura orokorrean eta hala izan dela diseinatua hiru mailako arkitektura bat inplementatzeko. Hala, `EskaeraGUI` eta `NirePanela` klaseak *aurkezpen maila* egituratzeko eraiki dira. *Datuen maila* antolatzeko, hau da, aplikazioaren informazio-sistemaren iraunkortasuna kudeatzeko, `EskaeraDatuBasea` klasea inplementatu da. Horrenbestez, aplikazioaren eta datu-basearen arteko komunikazioa klase hori erabilita gauzatuko da. Konturatu, halaber, ezen aplikazioaren *domeinuaren logika* `EskaeraSistema` eta `Eskaera` klaseak erabilita eratuko dela.

Bukatzeko, azken atal honetan azalduko dugu nola funtzionatzen duen orain arte inplementatuta dagoen aplikazioak:

Lehenengo eta behin, aplikazioaren erabiltzaileak telefono-zenbaki bat aukeratzen du eta ***Bezeroa Sartu*** botoia zapaltzen du; dena ondo badoa bezeroaren izena agertuko da alboko testu-eremuan. Orduan, erabiltzaileak erosi nahi dituen produktuak aukeratzen ditu: produktu bakoitzari bere kopurua itsatsita eta ***Produktua Sartu*** botoia zapalduta. Produktu guztiak aukeratu dituenean

eskaera bukatutzat ematen da ***Eskaera Bukatu*** botoia zapalduta. Eskaera erabat bukatzeko ordainketa gauzatzen da eskatzen diren ordainketa-eremuak beteta eta ***Ordaindu*** botoia zapalduta.



7.18. irudia. `Produktua` taularen eremuak eta eremuen balioak.

Oinarrizko iturburu-kodea eta erabiliko den datu-basea, honako helbide honetatik eskura ditzakezu:

`http://ji.ehu.es/isw/Web_ISO.html`

Gauzak horrela, oinarrizko iturburu-kodea erabilita, honako lan hau egin behar duzu laborategian:

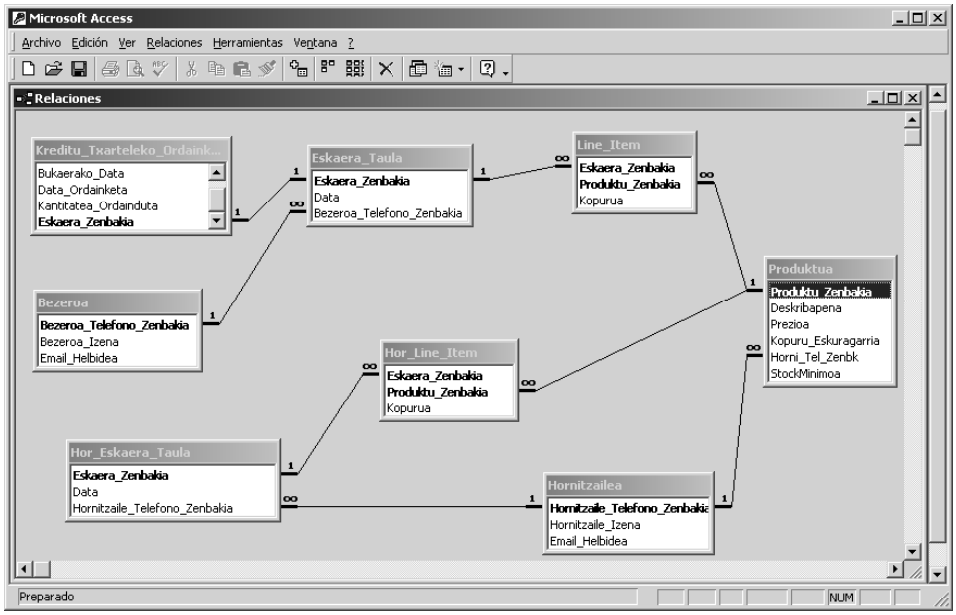
- **7.18. irudia**n agertzen den `Produktua` taularen eremuen balioei erreparatzen badiegu, berehala konturatzen gara eskaerak ugaltzen doazen heinean, produktu bakoitzaren stocka jaisten joango zaigula. Batzuetan, beherakada hain handia da, ezen stock minimora heltzen baita edo areago, stock minimo hori igaro eta zenbaki negatiboetan gelditzen baita produktuaren stocka. Halako egoeretan, aplikazioaren jabeak arazoak izango ditu eskaera osatzeko produktuak falta baitzaizkio biltegian.

  Aipatutako arazo larri horri aurre egiteko, funtzionalitate berri bat diseinatu eta programatu behar duzu orain arte eraikitako arkitekturan (hiru mailako arkitektura, alegia) oinarrituta. Horretarako, aplikazioaren menu nagusiari beste botoi (***Eskaera Hornitz***) bat gehituko zaio aipatutako funtzionalitatea inplementatzeko. Botoi hori zapalduta, aplikazioak, honako hau egikaritu beharko luke:

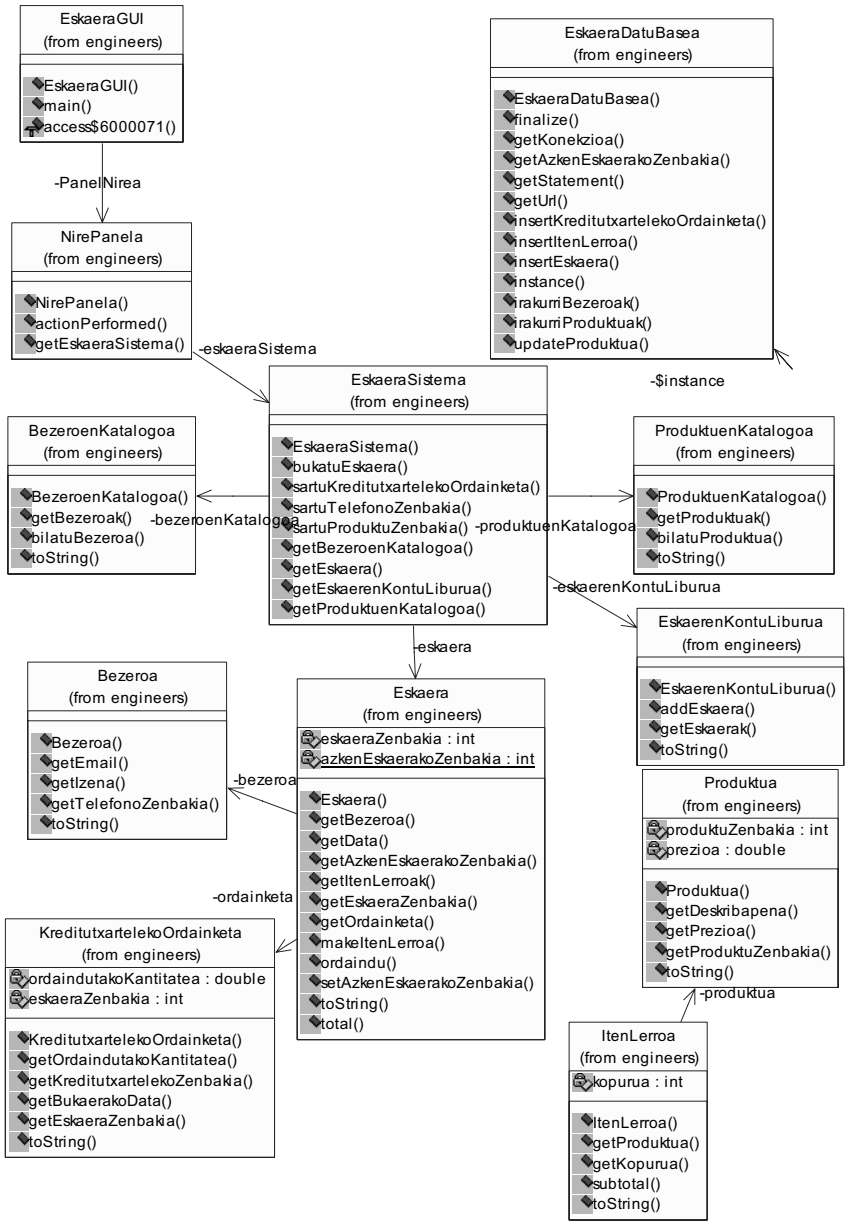
  – Kontuan izanda produktu bakoitzak bere hornitzailea duela, aplikazioak, eskaerak sortu behar ditu hornitzailearentzat. Hala behar denean, hornitzaile bati eskaera *bakarra* egingo zaio eta, bertan, produktuaren stock minimoa gehi behar den kopurua eskatuko zaio.



7.19. irudia. Aplikazioaren irteera grafikoa.



7.20. irudia. Aplikazioan erabiltzen diren taulak eta haien arteko erlazioak.



7.21. irudia. Aplikazioaren Klase Diagrama nagusia.

oOo

Laborategi honetan kutxazain automatiko baten softwarea eraikiko da ordenagailuan. Talde bakoitzak diseinatu behar duen aplikazioak **7.24. irudia**n agertzen den Klase Diagramari jarraitzen dio orpoz orpo eta, kutxazainak, **7.22. irudia**n agertzen den interfaze grafikoa aterako du bere pantailan.

Kutxazainak, lehenengo eta behin, kontu-zenbakia eta PIN pertsonalaren zenbakia eskatuko dizkio bezeroari. Bezeroak, eskatutako zenbakiak tekleatuko ditu eta `Sartu Txartela` botoia zapalduko du. Kontu-zenbakia eta PINa bat badatoz gainerako botoi eta testu-eremu guztiak gaitu egingo ditu aplikazio informatikoak eta gauzatu nahi dituen eragiketak egin ahal izango ditu bezeroak. Eragiketa guztiak eginda, `Bukatu` botoia zapaltzen du bezeroak saioa amaitutzat emateko.

Hiru mailako arkitektura (**7.24. irudia**) ezaguna inplementatzen du laborategi honetan garatu nahi den aplikazio informatikoak.

Laborategian ez da klase berririk diseinatu eta programatu behar, bakarrik, aplikazioaren iturburu-kodea jaitsi behar duzu web gunetik eta kodean agertzen diren zenbait hutsune bete. Hutsuneak, batez ere, datuen mailan (`KontuDatuBasea` klasea) eta domeinuaren mailan kokatu dira.

Laborategian gauzatu nahi ditugun eragiketak honako hauek dira:

1. Kontua eta PINa egiaztatu.

Bezeroaren kontu-zenbakia eta PIN zenbakia ondo egiaztatzen badira, doitu egiten dira interfaze grafiko nagusiaren osagaiak erabilgarri bihurtuz erabiltzailearentzat.

Bezeroaren kontu-zenbakia eta PIN zenbakia egiaztatzeko, `Kutxazaina` klasearen `sartuTxartela()` metodoa egiaztatzen da. Horretarako, `KontuNagusia` motako objektura bideratzen da programaren kontrola, `aurkituKontua()` metodoa egikarituta. `HashMap` motako objektu batean ditu `KontuNagusia` motako objektuak kontu guztiak. `KontuNagusia` motako objektua eraikitzen denean eskuratzen dira kontu horiek.

Kontua eskuratzen badu, `NirePanela` objektuak botoiak eta testu-eremuak gaitu egiten ditu, bestela, errore-mezua ateratzen du irteera estandarretik.

Funtzionalitate honetan, honako hau osatu behar duzu:

– Programatu `aurkituKontua()` metodoa `Kontua` motako objektua itzuliko duena.

  Konturatu hori egiteko kontuak eskuratu behar dituzula datu-basetik `KontuDatuBasea` klasearen `irakurriKontuak()` metodoa egikarituta. Eragiketa ondo gauzatzen bada, honako informazio hau ateratzen da irteera estandarretik:

```
Kontua: kontuZenbakia = 100 | pinZenbakia = 111 | saldoZaharra =
1000.0 | bezeroaIzena = Xabier Lamarka | bezeroaTelefonoZenbakia
= 9093453245 |
```

```
Kontua: kontuZenbakia = 200 | pinZenbakia = 222 | saldoZaharra =
2000.0 | bezeroaIzena = Jon Elorriaga | bezeroaTelefonoZenbakia
= 6193454356 |

Kontua: kontuZenbakia = 300 | pinZenbakia = 333 | saldoZaharra =
3000.0 | bezeroaIzena = Ane Garate | bezeroaTelefonoZenbakia =
7144571654 |

Kontua: kontuZenbakia = 400 | pinZenbakia = 444 | saldoZaharra =
4000.0 | bezeroaIzena = Nerea Iturgaitz |
bezeroaTelefonoZenbakia = 9093455693 |

Kontua: kontuZenbakia = 500 | pinZenbakia = 555 | saldoZaharra =
5000.0 | bezeroaIzena = Amaia Lekaroz | bezeroaTelefonoZenbakia
= 8187142547 |
```

Horrez gain, kontu egin ezen `HashMap` motako objektuan, kontuak honelako formatuan gordeko direla: (`Integer kontuZenbakia, Kontua kontua`), honako agindu hau, jarrita:

```
put(kontuZenbakia, kontua)
```

Gogoratu `HashMap` motako objektu batetik `get(kontuZenbakia)` agindua egikarituta eskuratzen direla elementuak.

2. Dirua sartzeko eta ateratzeko eragiketak antolatu.

Hemen, `Kutxazain` klaseko `sartuGordailua()` eta `sartuDirua()` metodoak programatu nahi dira `sartuKontsulta()` metodoa eredu gisa erabilita. Bai batak eta bai besteak, `Kontua` klasearen metodoak erabiltzen dituzte: `eginGordailua()` eta `ateraDirua()`. Bi metodo horiek programatu behar dituzu, `eginKontsulta()` metodoa eredu gisa hartuta.

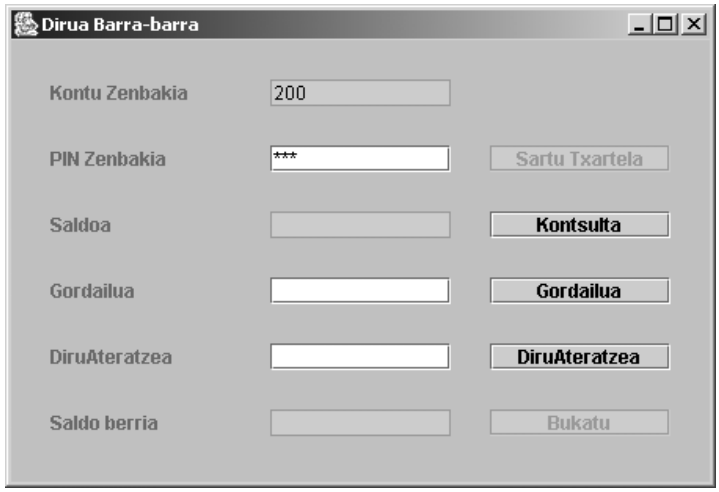
Kontu egin ezen bai batak eta bai besteak, `KontuDatuBasea` klasearen `sartuTransakzioa()` metodoa erabiltzen dutela.

3. Hasierako transakzio-zenbakia eskuratu.

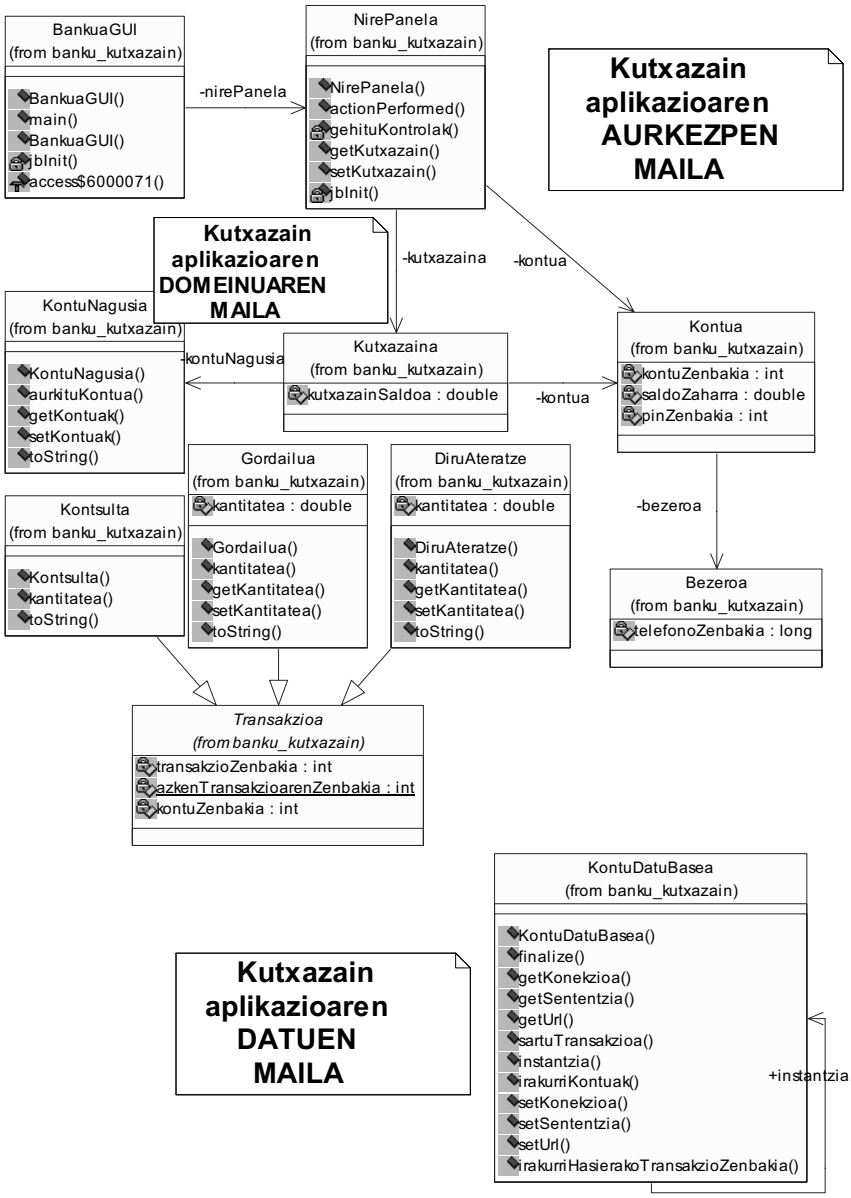
Inplementatu `irakurriHasierakoTransakzioZenbakia()` metodoa `KontuDatuBasea` klasean transakzio berriak ondo txertatzeko. Metodoak hutsik aurkitzen badu taula, lehenengo transakzioa 30000 zenbakiarekin hasieratzen du.



7.22. Aplikazioaren hasierako interfaze grafikoa.



7.23. irudia. Kutxazain automatikoaren interfaze grafikoa osagaiak doituta.



7.24. irudia. Aplikazioaren Klase Diagrama.

oOo

Jatetxe bateko menuak kudeatu nahi ditugu aplikazio informatiko bat eraikita. Jatetxeak bazkarietan eta afarietan eskaintzen dituen jakiak ateratzen ditu aplikazioak lehendabizi irteera estandarretik. Hori eginda, barazkijaleentzat bereziki prestatu dituen jakiak ateratzen ditu. Bukatzeko, kontsulta bat egiten

badio bezero batek zerbitzariari jaki bati buruz, lagundu egiten dio aplikazioak adieraziz barazkijaleentzat den ala ez.

Laborategi honetarako prestatu diren datuak erabilita, aplikazio informatikoak honako irteera hau emango du:

```
MENUAK
——
BAZKARIAK
Txitxirioak, 2.99 — Txitxirioak ziazerba eta kukubiltxoekin
Hegaluzea, 2.99 — Hegaluze entsalada chatkarekin
Sagarra, 3.49 — Sagar erreak marrubi naturalekin
Marrubia, 3.59 — Marrubi natural irabiakia

AFARIAK
Barazkijale-1, 2.99 —  Berenjena barrubeteak
Txitxarro, 2.99 — Txitxarroa labean pikilo saltsarekin
Indaba, 3.29 — Indaba zuriak odolostearekin
Solomoa, 3.05 — Solomoa plantxan gazta saltsarekin
Kardaberak, 3.99 — Kardaberak bizkaitar saltsan
Barazkiak, 3.89 — Barazkitxo salteatuak

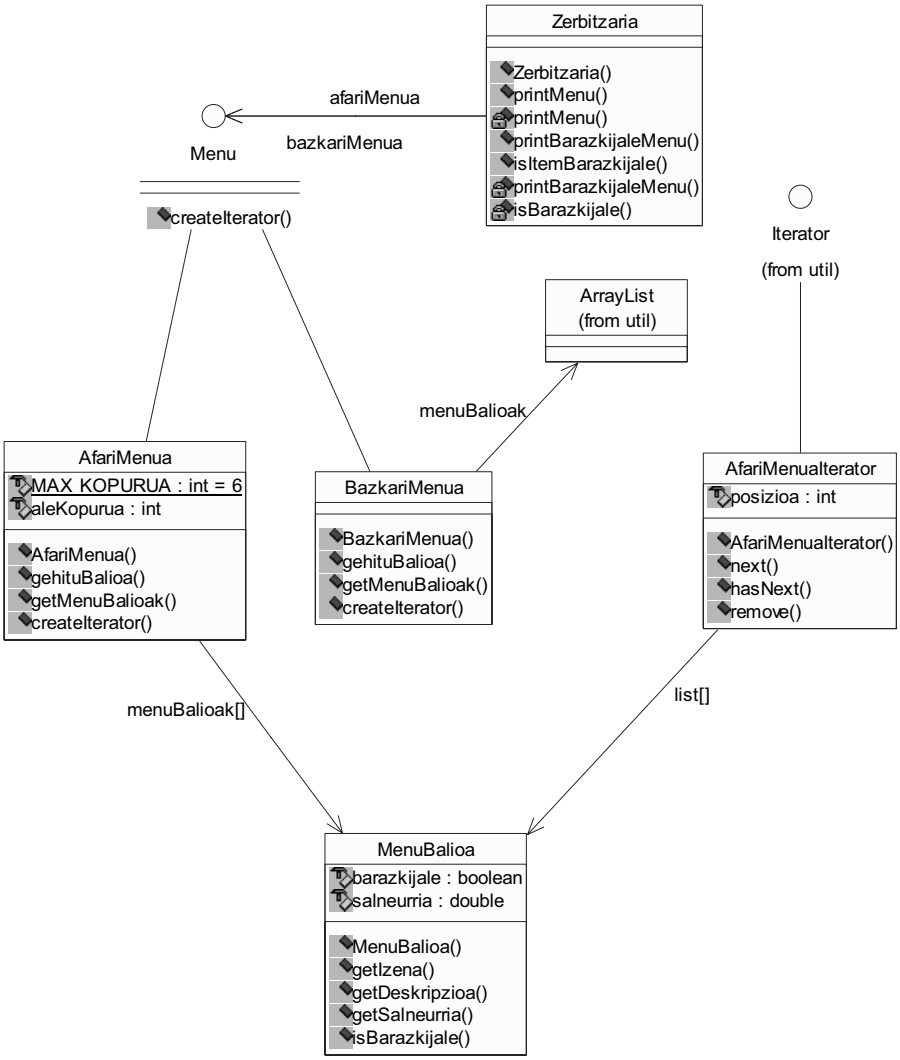
BARAZKIJALE MENUA
——
BAZKARIA
Txitxirioak         2.99
       Txitxirioak ziazerba eta kukubiltxoekin
Sagarra             3.49
       Sagar erreak marrubi naturalekin
Marrubia            3.59
       Marrubi natural irabiakia

AFARIA
Barazkijale-1       2.99
        Berenjena barrubeteak
Kardaberak          3.99
       Kardaberak bizkaitar saltsan
Barazkiak           3.89
       Barazkitxo salteatuak

Bezeroak galdetzen du, Barazkiak barazkijaleentzat?
Zerbitzariak dio: Bai

Bezeroak galdetzen du, Txitxirioak barazkijaleentzat?
Zerbitzariak dio: Bai
```

Aplikazio informatikoak ondoko irudian agertzen den Klase Diagrama erabiliko du diseinu-patroi ezagun bat inplementatzeko.



7.25. irudia. Aplikazioaren Klase Diagrama.

```
import java.util.*;
import java.util.Iterator;

public class MenuIteratzailea {
       public static void main(String args[]) {
              BazkariMenua bazkariMenua = new BazkariMenua();
              AfariMenua afariMenua = new AfariMenua();
    Zerbitzaria Zerbitzaria = new Zerbitzaria(bazkariMenua,
afariMenua);
              Zerbitzaria.printMenu();
              Zerbitzaria.printBarazkijaleMenu();
```

```
              System.out.println("\nBezeroak galdetzen du, Barazkiak
barazkijaleentzat?");
              System.out.print("Zerbitzariak dio: ");
              if (Zerbitzaria.isItemBarazkijale("Barazkiak")) {
                     System.out.println("Bai");
              } else {
                     System.out.println("Ez");
              }
   System.out.println("\nBezeroak galdetzen du, Txitxirioak
barazkijaleentzat?");
              System.out.print("Zerbitzariak dio: ");
              if (Zerbitzaria.isItemBarazkijale("Txitxirioak")) {
                     System.out.println("Bai");
              } else {
                     System.out.println("Ez");
              }  }   }

                                   oOo
public class Zerbitzaria {
       Menu bazkariMenua;
       Menu afariMenua;

       public Zerbitzaria(Menu bazkariMenua, Menu afariMenua) {
              this.bazkariMenua = bazkariMenua;
              this.afariMenua = afariMenua;
       }

       public void printMenu() {

      // metodo honek beheko printMenu(Iterator) metodoa erabiliko du
      //   menuak idazteko irteera estandarrean









       }

       private void printMenu(Iterator iterator) {









       }
```

```
        public void printBarazkijaleMenu() {
                System.out.println("\nBARAZKIJALE MENUA\n--\nBAZKARIA");
        //.............
                System.out.println("\nAFARIA");
        //.............
        }

        public boolean isItemBarazkijale(String izena) {









        }


        private void printBarazkijaleMenu(Iterator iterator) {









        }

        private boolean isBarazkijale(String izena, Iterator iterator) {









        }   }
```

oOo

```
import java.util.Iterator;

public interface Menu {
        public Iterator createIterator();
}
```

oOo

```
public class MenuBalioa {
       String izena;
       String deskripzioa;
       boolean barazkijale;
       double salneurria;

       public MenuBalioa(String izena,
                       String deskripzioa,
                       boolean barazkijale,
                       double salneurria)
       {
              this.izena = izena;
              this.deskripzioa = deskripzioa;
              this.barazkijale = barazkijale;
              this.salneurria = salneurria;
       }

       public String getIzena() {
              return izena;
       }

       public String getDeskripzioa() {
              return deskripzioa;
       }

       public double getSalneurria() {
              return salneurria;
       }

       public boolean isBarazkijale() {
              return barazkijale;
       }
}
                                   oOo
 import java.util.Iterator;

public class AfariMenua implements Menu {
       static final int MAX_KOPURUA = 6;
       int aleKopurua = 0;
       MenuBalioa[] menuBalioak;

       public AfariMenua() {
              menuBalioak = new MenuBalioa[MAX_KOPURUA];

      gehituBalioa("Barazkijale-1"," Berenjena barrubeteak", true,
2.99);
       gehituBalioa("Txitxarro",
                   "Txitxarroa labean pikilo saltsarekin", false, 2.99);
       gehituBalioa("Indaba",
              "Indaba zuriak odolostearekin", false, 3.29);
```

```
        gehituBalioa("Solomoa", "Solomoa plantxan gazta saltsarekin",
                    false, 3.05);
        gehituBalioa("Kardaberak",
                    "Kardaberak bizkaitar saltsan", true, 3.99);
        gehituBalioa("Barazkiak", "Barazkitxo salteatuak",
                    true, 3.89);
        }




        public void gehituBalioa(String izena, String deskripzioa,
                             boolean barazkijale, double salneurria)
        {









        }

        public MenuBalioa[] getMenuBalioak() {
        // .........
        }

                      public Iterator createIterator() {
        // ..........
                     }
        }
oOo
 import java.util.Iterator;

public class AfariMenuaIterator implements Iterator {
        MenuBalioa[] list;
        int posizioa = 0;

        public AfariMenuaIterator(MenuBalioa[] list) {
        // ..............
        }

        public Object next() {




        }
```

```
        public boolean hasNext() {




        }

        public void remove() {
                if (posizioa <= 0) {
                        throw new IllegalStateException
        ("Ezin duzu elementu bat ezabatu next() bat egin baino lehen");
                }
                if (list[ posizioa-1]  != null) {
                     for (int i = posizioa-1; i < (list.length-1); i++) {
                             list[ i]  = list[ i+1] ;
                     }
                     list[ list.length-1]  = null;
                } }    }
                                    oOo
import java.util.ArrayList;
import java.util.Iterator;

public class BazkariMenua implements Menu {
        ArrayList menuBalioak;

        public BazkariMenua() {
                menuBalioak = new ArrayList();

                gehituBalioa("Txitxirioak",
                           "Txitxirioak ziazerba eta kukubiltxoekin",
                        true,
                        2.99);

                gehituBalioa("Hegaluzea",
                        "Hegaluze entsalada chatkarekin",
                        false,
                        2.99);

                gehituBalioa("Sagarra",
                        "Sagar erreak marrubi naturalekin",
                        true,
                        3.49);

                gehituBalioa("Marrubia",
                                "Marrubi natural irabiakia",
                             true,
                             3.59);

                }
```

```
        public void gehituBalioa(String izena, String deskripzioa,
                                 boolean barazkijale, double salneurria)
        {



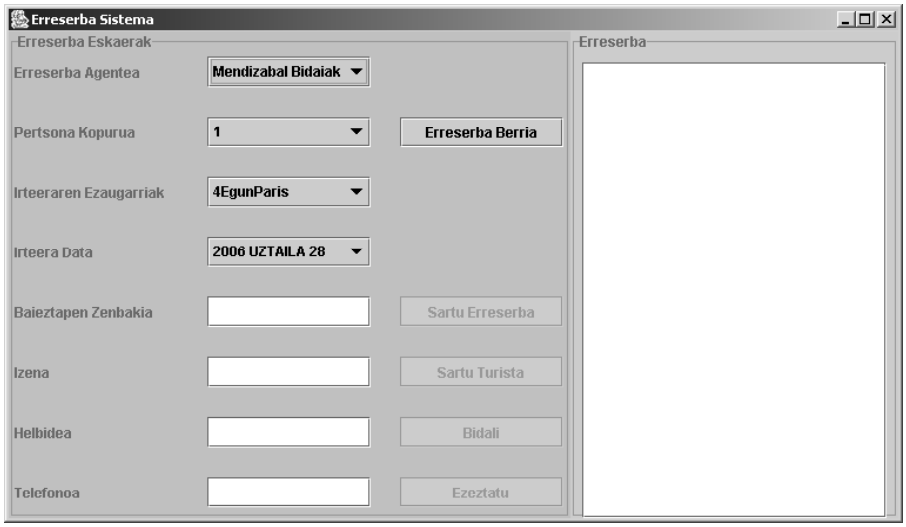

        }

        public ArrayList getMenuBalioak() {
// .........
        }

        public Iterator createIterator() {
        // .........
        }
        }
```

oOo

Bidaiak antolatzen dituen enpresa batek bere informazio-sistema mekanizatu nahi du zerbitzua hobetzeko eta kudeaketa-kostuak jaisteko. Horretarako beteki-zunen analisia egin du gurekin eta **7.26. irudia**n agertzen den interfaze grafikoa adostu du bidaien erreserbak kudeatzeko. Aurreko laborategietan egin den bezala praktika honetan ere aplikazio informatikoa hiru mailako arkitektura erabilita eraiki behar duzu, hau da, argi eta garbi bereiziz aplikazioaren aurkezpen maila, domeinuaren maila eta datuen maila.



7.26. irudia. Erreserba-sistemaren interfaze grafiko nagusia.

Interfaze grafikoak honako funtzionalitate hau izango du:

Erreserba bat antolatu nahi denean bulegoko langileak, lehendabizi, agente bat aukeratzen du eta zenbat pertsona joango diren antolatu nahi den irteerara. Kontu egin ezen erabiltzaileak agentea aukeratzen duenean interfaze grafikoan, `Irteera Ezaugarriak` eta `Irteera Data` eremuak eguneratu egiten direla. Hau da, `Mendizabal Bidaiak` aukeratzen bada lehenengo lerroan, beste bi eremu horietan, agente horrek eskaintzen dituen irteerak azalduko dira. Orduan, erabiltzaileak irteeraren ezaugarriak eta irteera-data aukeratu behar ditu.

Hori guztia eginda, erabiltzaileak, ***`Erreserba Berria`*** botoia zapaltzen du eta, egoera erreal batean, komunikazioa sortuko litzateke agentearen ordenagailuarekin ikusteko eta aztertzeko ea erreserba hori posible den aurrera eramatea. Egin badaiteke, baieztapen-zenbaki bat bidaltzen digu eta pantailan agertzen da. Bestela, mezu bat bidaltzen digu ohartaraziz irteera hori bertan behera utzi dela edo beteta dagoela.

Demagun baieztapen-zenbakia bidali digula. Orduan, interfaze grafikoko botoiak doitu egiten dira eta erreserba gauzatu egiten da adieraziz nor izango den erreserbaren erantzulea ordainketak egiteko orduan.

Konturatu pantailak beti ematen digula aukera erreserba ezeztatzeko. Transakzioaren gorabehera guztiak eskuineko testu-arean azalduko dira.

Honako hau eskatzen da:

- Erabilera-kasuaren deskribapena.

  Deskribapenak honako zehaztasunak izan behar ditu:

  Erabilera-kasuaren azalpena.
  Nola hasi eta bukatzen den erabilera-kasua.
  Gertaeren fluxu normala.
  Gertaeren fluxu ezohikoa.

  Erabilera-kasuak erabiltzen dituen erabiltzailearen interfaze grafikoen diseinua.

  Aurkitutako arazoen deskribapena.

- Domeinuaren Klase Diagrama.

  Atal honetan honako informazio hau eraiki eta egituratu behar duzu:

  Diagrama orokorra.
  Klaseak identifikatu.
  Klaseen arteko erlazioak atera.
  Klaseen atributuak eratu

Eredu erlazionalaren taulak sortu.
`Access` datu-basea eraiki.
Metodoak erantsi klaseei.

- Sekuentzia Diagrama.

  Sekuentzia Diagrama antolatu eredu erlazionala erabilita.

  Grafikoa.
  Azalpena.

  Erabilera-kasuan erabilitako SQL aginduak zehaztu.

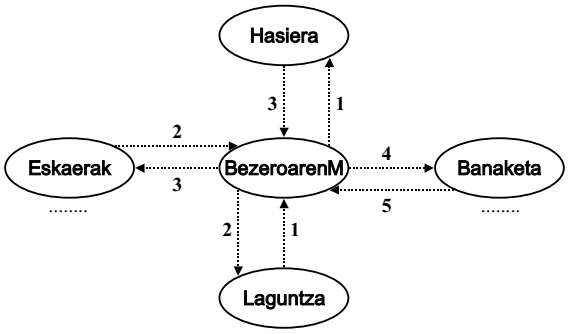
- Inplementazioa.

  Erabilera-kasuaren interfaze grafikoak eraiki (Aurkezpen Maila).
  Erabilera-kasuaren kudeatzaileak programatu (Domeinuaren Maila).
  `DBKudeatzailea` klasea diseinatu eta programatu (Datuen Maila).
  `SQLEmaitza` klasea diseinatu eta programatu (Datuen Maila).

oOo

Xabierrek eta Nereak hainbat urte daramate aplikazio informatikoak garatzen *XaNex* erakunde informatikoan. Aztertzen dituzten aplikazio informatikoek, behin eta berriz, gauza bera eskaini behar diote erabiltzaileari: interfaze grafiko ugari (**7.28. irudia**), gehienetan, antzeko diseinua dutenak. Interfaze grafikoaren goiko aldean, aplikazioaren eta, zehazki, domeinuaren informazioa agertzen da. Beheko aldean, hainbat botoi eskaintzen zaizkio erabiltzaileari pantailen arteko trantsizioak gauzatzeko. Pantailen formatu hori, behin eta berriz errepikatzen da aplikazioan eta garatzen diren aplikazio gehienetan.

Hori dela-eta, erabaki dute aplikazioen pantailen kudeaketa eta pantailen trantsizioak antolatzeko aplikazio txiki bat garatuko dutela egoera-makina bat simulatuko duena. Adibidez, jarraian itsatsi den interfaze grafikoak (**6.28. irudia**) honako trantsizio hauek izango lituzke aipatutako egoera-makinan:



7.27. irudia. Interfaze grafikoaren trantsizioak.

Irudiko biribil bakoitzak aplikazioaren egoera bat islatzen du eta, zenbaki bakoitzak, egoera horretan egonda hautatzen den aukera. Trantsizioa nondik nora gauzatzen den adierazteko balioko du aukerak eta pantaila grafikoaren beheko botoiarekin gauzatuko da aplikazioa egikaritzen denean.

Hala, aplikazioaren domeinua Euskaltel izango balitz, honako metodoak izan beharko lituzke gure aplikazio txikiak, trantsizioak kudeatzeko:

```
Euskaltel euskaltel = new Euskaltel();
euskaltel.eraiki(egoeraKopuru. aukeraKopuru);
euskaltel.eraikiEgoera(egoera, indizea);
euskaltel.txertatuTrantsizioa(egoeraI, aukeraI, balioa);
```

Konturatu ezen metodoek aplikazioaren egoera-makina kudeatzen dutela trantsizioak gauzatzeko eta egituratzeko.

Kontu egin, halaber, ezen aplikazio txikiak berdin-berdin funtzionatu beharko lukeela domeinua aldatuko bagenu, hau da, Euskaltelen ordez beste bat aukeratuko bagenu, metodoei dei egiteko era ez litzateke aldatuko:

```
Panasonic panasonic = new Panasonic();
panasonic.eraiki(egoeraKopuru. aukeraKopuru);
panasonic.eraikiEgoera(egoera, indizea);
.........
Fagor fagor = new Fagor();
fagor.eraiki(egoeraKopuru. aukeraKopuru);
fagor.eraikiEgoera(egoera, indizea);
.........
```



7.28. irudia. Aplikazioaren menu nagusia.

Honako hau eskatzen da:

1. Adierazi zein izango litzatekeen aplikazioaren Klase Diagrama. Aplikazioaren interfaze grafikoak eraikitzeko `Dialog` klase ezaguna erabili. Klase bakoitzarentzat adierazi bere atributuak eta metodoen signatura.

2. Programatu enuntziatuan agertzen diren hiru metodoen (`eraiki()`, `eraikiEgoera()` eta `txertatuTrantsizioa()`) iturburukodea. Gogoratu, aplikazioaren egoera-makina programatzen ari zarela.

3. Programatu `Bezeroaren Marko Nagusia` (**7.28. irudia**) interfaze grafikoa adieraziz batez ere egoeren eta aukeren arteko trantsizioak. Batik bat, arreta, interfaze grafikoen arteko trantsizioetan jarri behar duzu.

oOo

Eroski erakunde kooperatibak dendetan saltzen ditu bere produktuak. Produktuak motetan sailkatzen dira eta produktu bakoitzak bere identifikadore bakarra dauka. Diseinatu nahi den erabilera-kasuak (`produktuen salmenta`) honako ezaugarri hauek izango ditu: bazkideak, produktuen erosketa burutzen duenean denda batean, sistemak, informazio hori biltegiratu egin behar du. Erosketa berean bazkideak produktu bat baino gehiago eros ditzake.

Produktu bakoitzaren stocka ezagutzen du sistemak eta produktuko, atributu bat izango du: `StockBerritzea`, zeinak adierazten baitu zenbat alez hornitu behar duen berriro stocka. Sistemak beti gauzatuko du eragiketa hori, stockak 20 unitate baino gutxiago badu.

Erosketa egiten den bakoitzean stocka eguneratu egiten da. Nahiz eta stocka zerora heldu, erosketa egin egiten da. Kasu horietan eskaera bat sortzen da bazkideen erosketak kudeatzeko. Eskaerak ondo jaso behar du eta kontrolatua izan behar du eskaeraren zati bakoitza zein erosketarekin erlazionatzen den erosketari bukaera egokia emateko. Denda bakoitzeko eskaerak, automatikoki[78] eraikitzen dira astelehenero lehen orduan. Horrela, prestatuta daude bezeroek erosketak egiten dituztenerako.

Kontu egin ezen stockean honako hau gerta daitekeela: stocka < 20 baino gutxiago izan, baina stocka oraindik ez da heldu zerora. Kasu horretan ez duzu egin behar eskaerarik bazkidearentzat.

Gertaera-fluxu nagusia (zehaztasunak zeuk adierazi):

78. Dendak, astero, eskaera bakarra eraikitzen du astean gertatu diren arazoak jasotzeko. Gure kasuan arazoa stockarekin gertatzen da eta horretarako sortzen da eskaera hori.

- Saltzaileak dendaren eta bazkidearen kodeak sartzen ditu.
- Saltzaileak, banan-banan, bazkideak erosi nahi dituen produktuen kodea eta nahi den kopurua aukeratzen ditu. Salmenta bakoitzean, produktuko, gehienez, 10 unitate eros daitezke.
- Pantaila bat bistaratzen da egindako erosketak eta ordainduko den kopurua adierazten dituena.

Edozein unetan kontsulta daitezke bezero baten salmentak eta urteko eskaeren egoera.

Laborategi honetan honako hau eskatzen da:

- Goian azaldutako informazioaren Klase Diagrama. Beharrezkoa da adieraztea klase bakoitzaren nahitaezko atributuak eta erlazioak. Halaber, deskribapen txikia itsatsi bakoitzari.
- Bazkideen produktuen salmenta erabilera-kasuaren Sekuentzia Diagrama (bakarrik ebazpen erlazionala eskatzen da). Adierazi, halaber, Klase Diagrama inplementatzeko erabiliko dituzun eredu erlazionalaren taulak.
- Inplementatu behin betiko Sekuentzia Diagrama.

oOo

Bibliografia

Adamson, C. eta Marinacci, J. (2005): *Swing Hacks*, O´Reilly.

Agile Alliance (2001a): *Manifesto for Agile Software Development*, <http://www.agilealliance.org/>.

––––––––––, (2001b): *Principles: The Agile Alliance*, <www.agilealliance.org/principles.html>.

Ahern, D.; Clouse, A. eta Turner, R. (2003): *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*, Addison Wesley.

Aho, A. V.; Kernighan, B. W. eta Weinberger, P. J. (1979): "Awk: a pattern scanning and processing language", *Software Practice and Experience*, **9(4)**, 267-79.

Alexander, C. (1979): *The Timeless Way of Building*, Oxford University Press.

Alexander, C.; Ishikawa, S. eta Silverstein, M. (1977): *A Pattern Language*, Oxford University Press.

Alur, D.; Malks, D.eta Crupi, J. (2003): *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall.

Ambler, S. W. (1995): *The Object Primer: Application Developer's Guide to Object Orientation*, Cambridge University Press.

————————, (1997): *Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology*, Cambridge University Press, <www.ambysoft.com/buildingObjectApplications.html>.

————————, (1998): *Process Patterns: Building Large-Scale Systems Using Object Technology*, Cambridge University Press, <www.ambysoft.com/processPatterns.html>.

————————, (1999): *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*, Cambridge University Press, <www.ambysoft.com/moreProcessPatterns.html>.

––––––––––, (2001a): *The Object Primer, Second Edition: The Application Developer's Guide to Object Orientation*, Cambridge University Press, <www.ambysoft.com/theObjectPrimer.html>.

––––––––––, (2001b): *Enterprise Unified Process White Paper*, <www.roninintl.com/publications/unifiedProcess.htm>.

––––––––––, (2001c): *Agile Modeling Home Page*, <http://www.agilemodeling.com/>.
––––––––––, (2001d): *The Design of a Robust Persistence Layer*, <www.ambysoft.com/persistenceLayer.html>.
––––––––––, (2001e): *Mapping Objects to a Relational Database*, <www.ambysoft.com/mappingObjects.html>.
Arnold, K.; Gosling, J. eta Holmes, D. (2000): *The Java™ Programming Language*, Addison Wesley.
Arnow, D. eta Weiss, G. (2000): *Introducción a la programación con Java. Un enfoque orientado a objetos*, Addison Wesley.
Ashmore, D. (2004): *The J2EE Architect's Handbook*, Dvt Press.
Avondolio, D.; Richardson, W. C.; Schrager, S.; Mitchell, M. W. eta Scanlon, J. (2007): *Professional Java JDK 6 Edition*, Wrox.
Bass, L.; Clements, P. eta Kazman, R. (1998): *Software Architecture in Practice*, Addison Wesley .
Bauer, C. eta King, G. (2006): *Java Persistence with Hibernate*, Manning.
Beck, K. (2000): *Extreme Programming Explained: Embrace Change*, Addison Wesley.
Beck, K. eta Cunningham, W. (1989): *A Laboratory for Teaching Object-Oriented Thinking*, Proceedings of OOPSLA'89, 1-6.
Beck, K. eta Fowler, M. (2001): *Planning Extreme Programming*, Addison Wesley.
Beedle, M. eta Schwaber, K. (2001): *Agile Software Development With SCRUM*, Prentice Hall.
Beck, K. (2002): *Una explicacion de la Programación Extrema. Aceptar el cambio*, Addison Wesley.
————, (2003): *Test Driven Development: By Example*, Addison Wesley.
————, (2004): *JUnit Pocket Guide*, O`Reilly.
Begin, C.; Goodin, B. eta Meadors, L. (2007): *iBATIS in Action*, Manning.
Bennett, D. (1997): *Designing Hard Software: The Essential Tasks*, Manning Publications.
Bezroukov, N. (1999a): *Open Source Software Development as a Special Type of Academic Research*, FirstMonday, Volume 4 - Number 10.
————, (1999b): *A Second Look at the Cathedral and Bazaar*, FirstMonday, Volume 4 - Number 12.
Bishop, J. (1999): *Java. Fundamentos de programación*, Addison Wesley.
Black, D. A. (2006): *Ruby for Rails. Ruby techniques for Rails developers*, Manning.
Boehm, B. W. (1981): *Software Engineering Economics*, Prentice Hall.
––––––––––, (1988): "A Spiral Model Of Software Development and Enhancement", *IEEE Computer*, **21(5)**, 61-72.

Booch, G. (1994): *Object-Oriented Analysis and Design with Applications*, Addison Wesley.

Booch, G.; Rumbaugh, J. eta Jacobson, I. (1999): *El Lenguaje Unificado de Modelado*, Addison Wesley.

Bremer, M. (1999): *UnTechnical Writing: How to Write About Technical Subjects and Products So Anyone Can Understand*, UnTechnical Press.

Brooks, F. P. (1995): *The Mythical Man Month: Essays on Software Engineering Anniversary Edition*, Addison Wesley.

Brown, W. J.; McCormick, H. W. III, eta Thomas, S. W. (2000): *AntiPatterns in Project Management*, John Wiley & Sons.

Bruegge, B. eta Dutoit, A. (2003): *Object-Oriented Software Engineering: Using UML, Patterns and Java*, Prentice Hall.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P. eta Stal, M. (1996): *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons.

Caballe, S. (2007): *Aplicaciones distribuidas en Java con tecnologia RMI*, Delta Publicaciones.

Campione, M.; Walrath, K.; Zakhour, S. eta Bodoff, S. (2005): *The Java™ Tutorial: A Short Course on the Basics*, Addison Wesley.

Ceballos, F. J. (2000): *Java 2. Curso de programación*, Rama.

Chrissis, M.; Konrad, M. eta Shrum, S. (2003): *CMMI: Guidelines for Process Integration and Product Improvement*, Addison Wesley.

Christel, M. G. eta Kang, K. C. (1992): *Issues in Requirements Elicitation*, Software Engineering Institute (SEI) Technical Report CMU/SEI-92-TR-12, <http://www.sei.cmu.edu/>.

Coad, P.; Lefebvre, E. eta DeLuca, J. (1999): *Java Modeling in Color with UML: Enterprise Components and Process*, Prentice Hall.

Coady, Y.; Kiczales, G.; Feeley, M. eta Smolyn, G. (2001): *Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code*, Proceedings of the Joint European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9), Vienna, Austria, September, 78-88.

Cockburn, A. (1998): *Surviving Object-Oriented Projects: A Manager's Guide*, Addison Wesley.

————, (2001a): *Writing Effective Use Cases*, Addison Wesley, Boston, MA.

————, (2001b): *Crystal Clear: A Human-Powered Software Development Methodology for Small Teams*, <http://members.aol.com/humansandt/crystal/clear/>.

————, (2001c): *Characterizing People as Non-Linear, First-Order Components in Software Development*, <members.aol.com/humansandt/papers/nonlinear/ nonlinear.htm>.

————, (2002): *Agile Software Development*, Addison Wesley.
Cohen, F. (2004): *Java Testing and Design : From Unit Testing to Automated Web Tests*, Prentice Hall.
Colyer, A.; Clement, A.; Harley, G. eta Webster, M. (2005): *Aspect-Oriented Programming in Eclipse with AspectJ and the AspectJ Development Tools*, Addison Wesley.
Constantine, L. L. eta Lockwood, L. A. D. (1999): *Software For Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, ACM Press.
Coplien, J. O. (1997): *Idioms and Patterns as Architectural Literature*, IEEE Software Special Issue on Objects, Patterns, and Architectures, January.
————, (1998): *Software Design Patterns: Common Questions and Answers*.
Rising, L. (ed.), *The Patterns Handbook: Techniques, Strategies, and Applications*, Cambridge University Press, January, 311-320.
Coplien, J. eta Harrison, N. (2001): *Organizational Patterns Site*, <www.belllabs.com/cgi-user/OrgPatterns/OrgPatterns>.
Crawford, W. eta Kaplan, J. (2003): *J2EE Design Patterns*, O´Reilly.
Date, C. J. eta Darwen H. (2006): *Databases, Types and the Relational Model*, Addison Wesley.
Davis, A. M. (1995): *201 Principles of Software Development*, McGraw Hill.
Deitel, H. eta Deitel, P. (2004): *Cómo Programar en Java*, Prentice Hall.
Dijkstra, E. W. (1976): *A Discipline of Programming*, Prentice Hall.
————, (1982): *Selected writings on computing: a personal perspective*, Springer Verlag.
Douglass, B. P. (1999): *Doing Hard Time: Developing Real-Time Systems With UML, Objects, Frameworks, and Patterns*, Addison Wesley.
D'Souza, D. F. etaWills, A. C. (1999): *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison Wesley.
Edlich, S.; Hörning, H.; Hörning, R. eta Paterson, J. (2006): *The Definitive Guide to db4o*, Apress.
El Eman, K.; Drouin, J. eta Melo W. (1998): *SPICE. The theory and practice of Software Process Improvement and Capability Determination*, IEEE Computer Society Press.
Elliott, J.; Loy David, M. eta Brian Cole, W.(2002): *Java Swing*, O´Reilly.
Elmasri R. eta Navathe S. B.(2007): *Fundamentals of Database Systems*. Addison Wesley.
Elrad, T.; Aksit, M.; Kiczales, G.; Lieberherr, K. eta Ossher, H. (2001): *Discussing Aspects of AOP*, Communications of the ACM, October, 33-38.
European Space Agency. *Ariane 5: Flight 501 failure. Report by the enquiry board*, <www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html>.
Evans, G. (2001): *Palm Sized Process: Point of Sale Gets Agile*, Software Development, September.
Fowler, M. (1997): *Analysis Patterns: Reusable Object Models*, Addison Wesley.

————————, (1999): *Refactoring: Improving the Design of Existing Code*, Addison Wesley.
————————, (2001a): *The New Methodology*, <www.martinfowler.com/articles/newMethodology.html>.
————————, (2001b): *Is Design Dead?* <www.martinfowler.com/articles/designDead.html>.
————————, (2002): *Patterns of Enterprise Application Architecture*. Addison Wesley.
Fowler, M. eta Scott, K. (1999): *UML Distilled Second Edition: A Brief Guide to the Standard Object Modeling Language*, Addison Wesley .
Fowler, C. (2006): *Rails Recipes*, Pragmatic Bookshelf.
Frankel, D. (2003): *Model Driven Architecture: Applying MDA to Enterprise Computing*, John Wiley & Sons.
Freeman, E, eta Freeman, E. (2004): *Head First Design Patterns*, O´Reilly Media.
Gane, C.eta Sarson, T. (1979): *Structured Systems Analysis: Tools and Techniques*, Prentice Hall.
Froufe, A.(2005): *Java 2. Manual de usuario y tutorial*, RA-MA.
Gamma, E.; Helm, R.; Johnson, R. eta Vlissides, J. (1995): *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.
Gilb, T. eta Graham, D. (1993): *Software Inspection*, Addison Wesley.
Goldberg, A. (1984): *Smalltalk-80: The Interactive Programming environment*, Addison Wesley.
Goldberg, A. eta D. R. (1983): *Smalltalk-80: The Language and Its Implementation*, Addison Wesley.
Gottesdiener, E. (2001): *Specifying Requirements With a Wall of Wonder*, <http://www.therationaledge.com/content/nov_01/t_wallOfWonder_eg.html> .
Gradecki, J. eta Lesiecki, N. (2003): *Mastering AspectJ*, Wiley.
Graham, I.; Henderson-Sellers, B. eta Younessi, H. (1997): *The OPEN Process Specification*, ACM Press.
Grand, M. (1998): *Patterns in Java. Vol I. A catalog of reusable Design Patterns illustrated with UML*, John Wiley & Sons.
————————, (1998): *Patterns in Java, Vol II*, John Wiley & Sons.
Greenbaum, J. eta Kyng, M. (1991): *Design At Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates.
Grosso, W.(2001): *Java RMI*. O´Reilly.
Henderson-Sellers, B. (1992): *A book of object-oriented knowledge*, Prentice Hall.
Highsmith, J. A. III (2000): *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House.
Hightower, R.; Onstine, W.; Visan, P.; Payne, D. eta. Gradecki, J. (2004): *Professional Java Tools for Extreme Programming: Ant, XDoclet, JUnit, Cactus, and Maven (Programmer to Programmer),* Wrox.

Hock, D. W. (2000): *Birth of the Chaordic Age*, Berrett-Koehler.
Hohmann, L. (1996): *Journey of the Software Professional: The Sociology of Computer Programming*, Prentice Hall.
Holzner, D. (2006): *Beginning Ruby on Rails*, Wrox.
Humphrey, W. S. (1995): *A discipline for Software Engineering*, Addison Wesley.
————————, (1999): *Introduction to the Team Software Process*, Addison Wesley.
————————, (2001): *Introducción al Proceso del Software Personal*, Addison Wesley.
Hunt, J. (2003): *Guide to the Unified Process Featuring UML, Java and Design Patterns*, Springer.
Hunt, A. eta Thomas, D. (2000): *The Pragmatic Programmer: From Journeyman to Master*, Addison Wesley.
Hunt, A. eta Thomas, D. *Pragmatic Unit Testing in Java with JUnit,* The Pragmatic Programmers.
Jacobson, I.; Booch, G. eta Rumbaugh, J. (1999): *The Unified Software Development Process*, Addison Wesley.
————; ———— eta ————, (2000): *El Proceso Unificado de Desarrollo de Software*, Addison Wesley.
Jacobson, I.; Christerson, M.; Jonsson, P. eta Overgaard, G. (1992): *Object-Oriented Software Engineering: A Use Case Driven Approach*, ACM Press.
Jacobson, I.; Griss, M. eta Jonsson, P. (1997): *Software Reuse: Architecture, Process, and Organization for Business Success*, ACM Press.
Jacobson, I. eta Pan-Wei Ng. (2005): *Aspect-Oriented Software Development with Use Cases*, Addison Wesley.
Jeffries, R.; Anderson, A. eta Hendrickson, C. (2001): *Extreme Programming Installed*, Addison Wesley, Boston, MA.
Jeffries, R. (2001a): *Essential XP: Card, Conversation, Confirmation*, <www.xprogramming.com/xpmag/expCardConversationConfirmation.htm>.
————————, (2001b): *Essential XP: Documentation*, <www.xprogramming.com/xpmag/expDocumentationInXp.htm>.
Johnson, R. eta Hoeller, J. (2004): *Expert One-on-One J2EE Development without EJB,* Wrox.
Johnson, R.; Hoeller, J.; Arendsen, A.; Risberg T. eta Sampaleanu C. (2005): *Professional Java Development with the Spring Framework*, Wrox.
Jones, C. (2000): *Software Assessments, Benchmarks, and Best Practices*, Addison Wesley.
Kasse, T. (2004): *A Practical Guide to CMMI Implementation*, Artech House.
Keith, M. eta Schincariol, M. (2006): *Pro EJB 3: Java Persistence API*, Apress.
Kerievsky, J. (2001): *Patterns and XP. Extreme Programming Examined*, 207-220, ed. G. Succi eta M. Marchesi, Addison Wesley.
Kerth, N. (2001): *Project Retrospectives: A Handbook for Team Reviews*, Dorset House.

Kiczales, G.; Rivières, J. eta Bobrow, D. G. (1991): *The Art of the Metaobject Protocol*, MIT Press.

Kiczales, G. (1992): *Towards a New Model of Abstraction in the Engineering of Software*, Proceedings of the International Workshop on New Models for Software Architectures (IMSA): Reflection and Metalevel Architecture, Tokyo, Japan, November, 1-11.

Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J. eta Griswold, W. (2001): *An Overview of AspectJ*, European Conference on Object-Oriented Programming (ECOOP), LNCS 2072, Springer-Verlag, Budapest, 327-353.

Kiselev, I. (2002): *Aspect-Oriented Programming with AspectJ*, Sams.

Koenig, D.; Glover, A.; King, P.; Laforge, G. eta Skeet, J. (2007): *Groovy in Action*. Manning.

Kruchten, P. (1995): "The 4+1 View Model of Architecture", *IEEE Software*, **12(6)**, 42–50.

––––––––––, (2000): *The Rational Unified Process, Second Edition: An Introduction*, Addison Wesley.

Kuwabara, K. (2000): *Linux: A Bazaar at the Edge of Chaos*, FirstMonday, Volume 5 - Number 3.

Kuchana, P. (2004): *Software Architecture Design Patterns in Java*, Auerbach Publications.

Laddad, R. (2002): *I Want My AOP! Parts1-3*, Java World, <www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html>.

————————, (2003): *AspectJ in Action*, Manning Publications.

Lane, D. (2004): *JUnit: The Definitive Guide*, O`Reilly.

Lano, K. (2005): *Advanced Systems Design with Java, UML and MDA*, Butterworth-Heinemann.

Lans, R. F. van der (2006): *Introduction to SQL: Mastering the Relational Database Language*, Addison Wesley Professional.

Larman, C. (2002): *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall.

————————, (2002): *UML y Patrones*, Prentice Hall.

Lee, R. eta Tepfenhart, W. (2003): *Practical Object-Oriented Development with UML and Java*, Prentice Hall.

Lesieki, N. (2002): *Test Flexibly with AspectJ and Mock Objects*, IBM DeveloperWorks, <www-106.ibm.com/developerworks/java/library/j-aspectj2/>.

Leuf, B. eta Cunningham, W. (2001): *The Wiki Way: Quick Collaboration on the Web*, Addison Wesley .

Li, L. (1998): *Java. Data Structures and Programming*, Springer-Verlag.

Lima Díaz, F. (2000): *Manual avanzado de Java 2*, Anaya Multimedia.

Link, J. eta Fröhlich, P. (2003): *Unit Testing in Java: How Tests Drive the Code*, Morgan Kaufman.

Linthicum, D. S. (2000): *Enterprise Application Integration*, Addison Wesley .

Martin, R. (2001): *The Process*, <www.objectmentor.com/publications/RUPvsXP.pdf>.

Massol, V. (2004): *JUnit in Action*, Manning Publications.

Mayhew, D. J. (1992): *Principles and Guidelines in Software User Interface Design*, Prentice Hall.

McConnell, S. (1993): *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press.

McGovern, J.; Tyagi, S.; Stevens, M. eta Mathew, S. (2003): *Java Web Services Architecture*, Elsevier Science.

McNiff K. eta Pitt. E. (2001): *Java RMI: The Remote Method Invocation Guide*, Addison Wesley.

Meyer, B. eta Dino Mandrioli, D. (1992): *Advances in Object-Oriented Software Engineering*, Prentice Hall.

Meyer, B. (1992): "Applying Design by Contract", *IEEE Computer,* **25(10)**, 40-51.

————, (1995): "On formalism in specifications", *IEEE Software,* **2(1)**, 6-26.

————, (1997): *Object-Oriented Software Construction*, Prentice Hall, [2. argitalpena].

————, (1998): *Construcción de Software Orientado a Objetos*, Prentice Hall.

Microsoft Corporation (1995): *The Windows Interface Guidelines for Software Design: An Application Design Guide*, Microsoft Press.

––––––––––, (2001): *The Microsoft Solutions Framework (MSF)*, <www.microsoft.com/business/services/mcsmsf.asp>.

Miles, R. (2005): *AspectJ Cookbook*, O´Reilly Media.

Monson-Haefel R.; Burke, B. eta Labourey S. (2001): *Enterprise JavaBeans*, O´Reilly.

Monson-Haefel, R. (2003): *J2EE Web Services*, Addison Wesley Professional.

Musser, D. eta Saini, A. (1995): *STL Tutorial and Reference Guide: C++Programming with the Standard Template Library*, Addison Wesley.

Musser, D. R.; Derge, G. J. eta Saini, A. (2001): *STL Tutorial and Reference Guide, Second Edition: C++ Programming with the Standard Template Library*, Addison Wesley.

Myers, T. eta Nakhimovsky, A. (2003): *Google, Amazon, and Beyond: Creating and Consuming Web Services*, Apress.

Naiburg, E. J. eta Maksimchuk, R. A. (2001): *UML for Database Design*, Addison Wesley, Boston, MA.

Newcomer, E. eta Lomow, G. (2005): *Understanding SOA with Web Services*, Addison Wesley Professional.

Newkirk, J. eta Martin, R. C. (2001): *Extreme Programming in Practice*, Addison Wesley.

Niemeyer, G. eta Poteet, J. (2003): *Extreme Programming with Ant: Building and Deploying Java Applications with JSP, EJB, XSLT, XDoclet, and JUnit*, Sams.

Öberg, R.(2001): *Mastering RMI: Developing Enterprise Applications in Java and EJB,* John Wiley & Sons.

Object Management Group (2001a): *The Unified Modeling Language (UML) Specification*, <www.omg.org/technology/documents/formal/uml.htm>.

––––––––––, (2001b): *Model Drive Architecture (MDA)*, <ftp.omg.org/pub/docs/ormsc/01-07-01.pdf>.

Page-Jones, M. (1988): *Practical Guide to Structured Systems Design 2/e*, Prentice Hall.

–––––––––, (2000): *Fundamentals of Object-Oriented Design in UML*, Dorset-House Publishing.

Panda, D.; Rahman, R. eta Lane, D. (2007): *EJB 3 in Action*, Manning.

Parnas, D. (1972): *On the Criteria To Be Used in Decomposing Systems into Modules*, Communications of the ACM, December, 1053-1058.

Parsian, M. (2005): *JDBC Recipes: A Problem-Solution Approach*, Apress.

–––––––––, (2006): *JDBC Metadata, MySQL, and Oracle Recipes: A Problem-Solution Approach*, Apress.

Paulk, M.; Weber, C.; Curtis, B. eta Chrissis, M. (1995): *The Capability Maturity Model for Software: Guidelines for Improving the Software Process*, Addison Wesley.

Pawlak, R.; Retaillé, J-P. eta Seinturier, L. (2005): *Foundations of Aop for J2ee Development*, Apress.

Piattini, M.; Calvo-Manzano, J. A.; Cervera, J. eta Fernández, L. (1996): *Análisis y diseño detallado de Aplicaciones Informáticas de Gestión*, Rama.

Pressman, R. S. (2002): *Ingeniería del Software. Un enfoque práctico*, McGraw Hill, [5. argitalpena].

Pugh, E. eta Gradecki, J. D. (2004): *Professional Hibernate*, Wrox.

Pulier, E. eta Taylor. H. (2005): *Understanding Enterprise SOA*, Manning.

Rainsberger, J. B. (2004): *JUnit Recipes : Practical Methods for Programmer Testing*, Mannig Publications.

Raskin, J. (2000): *The Human Interface: New Directions for Designing Interactive Systems*, Addison Wesley.

Rational Corporation (2001): *Rational Unified Process Home Page*, <www.rational.com/products/rup/index.jsp>.

Raymond, E. (1998): *The Cathedral and the Bazaar*, FirstMonday, Volume 3 – Number 3.

–––––––––, (2001): *The Cathedral and the Bazaar*, O`Reilly.

Reingruber, M. C. eta Gregory, W. W. (1994): *The Data Modeling Handbook: A Best-Practice Approach to Building Quality Data Models*, John Wiley & Sons.

Richardson, C. (2006): *POJOs in Action*, Manning.
Riel, A. J. (1996): *Object-Oriented Design Heuristics*, Addison Wesley.
Robinson, M. eta Vorobiev, P. (2003): *Swing*, Manning.
Rocher, G. (2006): *The Definitive Guide to Grails,* Apress.
Roman, E.; Ambler, S. W.; Jewell, T. eta Marinescu, F. (2002): *Mastering Enterprise Java Beans, Second Edition*, John Wiley & Sons.
Rosenberg, D. eta Scott, K. (1999): *Use Case Driven Object Modeling With UML: A Practical Approach*, Addison Wesley.
Ross, R. G. (1997): *The Business Rule Book, Second Edition*, Business Rules Solutions.
Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F. eta Lorensen, W. (1991): *Object-Oriented Modeling and Design*, Prentice Hall.
Rumbaugh, J.; Jacobson, I. eta Booch, G. (1999): *The Unified Modeling Language Reference Manual*, Addison Wesley.
————; ———— eta ————, (2000): *El Lenguaje Unificado de Modelado Manuel de Referencia*, Addison Wesley.
Sam-Bodden, B. (2006): *Beginning POJOs: Lightweight Java Web Development Using Plain Old Java Objects in Spring, Hibernate, and Tapestry*, Apress.
Sierra, K. eta Bates, B. (2003): *Head First EJB*, O´Reilly.
Software Engineering Institute (1995): *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley, Inc.
Sperko, R. (2003): *Java Persistence for Relational Databases*, Apress.
Stapleton, J. (1997): *DSDM: Dynamic Systems Development Method*, Addison Wesley.
Stroustrup, B. (1997): *The C++ Programming Language*, Addison Wesley, [3. argitalpena].
Sullins, B. G. eta Whipple, M. B. (2003): *EJB Cookbook*, Manning.
Tarr, P.L. (1999): *N Degrees of Separation: Multi-Dimensional Separation of Concerns*, Proceedings of the International Conference on Software Engineering (ICSE), ACM, Los Angeles, 107-119.
Tate, B. eta Hibbs, C. (2006): *Ruby on Rails: Up and Running*, O´Reilly.
Thomas, J.; Young, M.; Brown, K. eta Glover, A. (2004): *Java Testing Patterns*, Wiley.
Thomas, D. eta Heinemeier, D. (2006): *Agile Web Development with Rails*, Pragmatic Bookshelf.
Vermeulen, A.; Ambler, S. W.; Bumgardner, G.; Metz, E.; Misfeldt, T.; Shur, J. Eta Thompson, P. (2000): *The Elements of Java Style*, Cambridge University Press.
Wake, W. C. (2002): *Extreme Programming Explored*, Addison Wesley.
Walnes, J.; Abrahamian, A.; Cannon-Brookes, M. eta Lightbody, P. (2003): *Java Open Source Programming: with XDoclet, JUnit, WebWork, Hibernate*, Wiley.

Warmer, J. eta Kleppe, A. (1999): *The Object Constraint Language: Precise Modeling With UML*, Addison Wesley.

Weiss, E. H. (1991): *How To Write Usable User Documentation*, The Oryx Press.

Wells, J. D. (2001): *Extreme Programming: A Gentle Introduction*, <www.extremeprogrammning.org/>.

Wiegers, K. (1999): *Software Requirements*, Microsoft Press.

Wilkinson, N. M. (1995): *Using CRC Cards: An Informal Approach to Object-Oriented Development*, Cambridge University Press.

Williams, L.; Kessler, R. R.; Cunningham, W. eta Jeffries, R. (2000): "Strengthening the Case for Pair Programming", *IEEE Software*, July/August, 19-25.

Williamson, A. R. (1999): *Java Servlets by Example*, Manning.

Wood, J. eta Silver, D. (1995): *Joint Application Development Second Edition*, John Wiley & Sons.

Xerox (2001): *Aspect-Oriented Programming*, <http://www.parc.xerox.com/csl/projects/aop/>.

Yourdon, E. (1989): *Modern Structured Analysis*, Prentice Hall.

————, (1997): *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice Hall.

Sailean argitaratu diren beste liburu batzuk

Algoritmika
Rosa Arruabarrena
1997an argitaratua
ISBN: 84-86967-82-1

Ordenadore bidezko irudigintza
Joseba Makazaga, Asier Lasa
1998an argitaratua
ISBN: 84-86967-90-2

Oinarrizko programazioa. Ariketa-bilduma
Arantza Diaz de Illarraza, Kepa Sarasola
1999an argitaratua
ISBN: 84-8438-002-5

Zirkuitu elektriko eta elektronikoen oinarrizko analisia
Olatz Arbelaitz, Txelo Ruiz
2001ean argitaratua
ISBN: 84-8438-018-1

Linux sistemaren eta sarearen administrazioa
Iñaki Alegria
2003an argitaratua
ISBN: 84-8438-040-8

Sistema digitalen diseinu-hastapenak. Oinarrizko kontzeptuak eta adibideak
Olatz Arbelaitz eta beste
2005ean argitaratua
ISBN: 84-8438-069-6

Softwarearen ingeniaritza [I. ATALA: Softwarearen garapenaren zenbait arlo]
Jose Ramon Zubizarreta Aizpuru
2006an argitaratua
ISBN: 84-8438-085-8