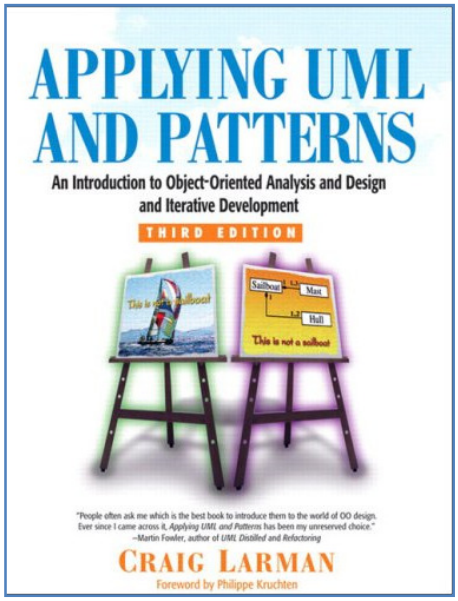
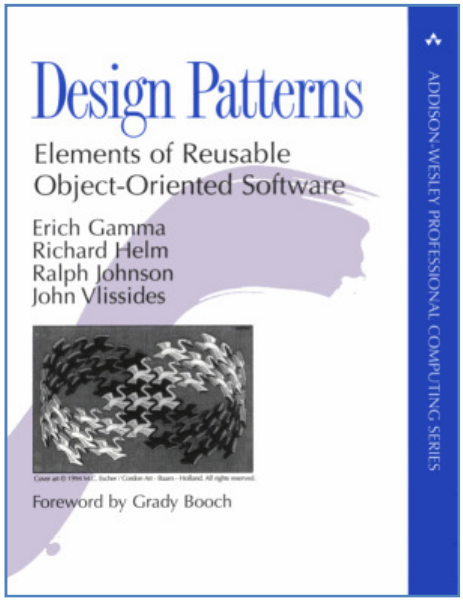
Craig Larman

Hyrje në Paternat e Dizajnit GoF

pjesa e dytë nga libri i mirënjohur për OOAD
që i referohet librit bazë të paternave





Përktheu dhe përshtati
Ridvan Bunjaku

Prishtinë
Gusht 2009

Përmbajtja

Parathënia e librit “Applying UML and Patterns”

Programimi është argëtues, por zhvillimi i softuerit cilësor është i vështirë. Ndërmjet ideve të mira, kërkesave apo "vizionit", dhe një produkti softuerik, ka shumë më shumë se programim. Analiza dhe dizajni, definimi se si me zgjidhë problemin, çka me programu, me mbërthy këtë dizajn në mënyra që janë të lehta për me komuniku, me rishiku, me implementu, dhe me evolu - është ajo që qëndron në thelb të këtij libri. Kjo është ajo që do ta mësoni.

Gjuha e Unifikuar e Modelimit - Unified Modeling Language (UML) është bërë gjuhë e pranuar universalisht për planet e dizajnit të softuerit. UML është gjuha vizuale që përdoret për m'i shprehë idetë e dizajnit nëpër këtë libër, që e thekson se si i aplikojnë zhvilluesit vërtet elementet e përdorura shpesh të UML, e jo veçoritë e errëta të gjuhës.

Rëndësia e paternave në përpunimin e sistemeve komplekse është pranuar moti në disciplinat tjera. Paternat e dizajnit softuerik janë ato që na lejojnë m'i përshkru fragmentet e dizajnit, dhe m'i ripërdorë idetë e dizajnit, duke i ndihmu zhvilluesit me përfitu nga ekspertiza e të tjerëve. Paternat i japin emër dhe formë eksperimenteve abstrakte, rregullave dhe praktikave më të mira të teknikave të orientuara kah objektet. Asnjë inxhinier i arsyeshëm nuk don me ia fillu nga një cung i thatë, dhe ky libër e ofron një paletë të paternave të dizajnit të gatshme m'i përdorë.

Mirëpo dizajni i softuerit duket paksa i thatë dhe misterioz kur nuk prezentohet në kontekst të një procesi të inxhinierimit të softuerit. Dhe në këtë temë, jam i impresionuar se për edicionin e tij të ri, Craig Larman ka zgjedhë me përqafu dhe me prezentu Procesin e Unifikuar (Unified Process), duke tregu se si mundet m'u përdorë ai në mënyrë relativisht të thjeshtë dhe pa ceremoni të mëdha. Duke i prezentu studimet e rasteve (case studies) në proces iterativ, të udhëhequr nga rreziqet, dhe arkitekturë-qendror, këshillat e Craig'ut kanë kontekst realistik; ai e ekspozon dinamikën e asaj që vërtet ndodh në zhvillimin e softuerit, dhe i tregon forcat e jashtme që janë në lojë. Aktivitetet e dizajnit janë të lidhura me punët tjera, dhe më nuk shfaqen si aktivitet i pastër mendor i transformimeve sistematike apo intuitës kreative. Dhe Craig dhe unë jemi të bindur në benefitet e zhvillimit iterativ, që do t'i shihni të ilustruara bujshëm nëpër libër.

Pra për mua, ky libër e ka miksin e duhur të përbërësve. Do ta mësoni një metodë sistematike për me bë Analizë dhe Dizajn të Orientuar kah Objektete (Object-Oriented Analysis and Design - OOA/D) nga një mësues i madh, një metodologjist briliant, dhe një "guru OO" që ua ka mësuar atë mijëra njerëzve nëpër botë. Craig e përshkruan metodën në kontekst të Procesit të Unifikuar (UP). Ai gradualisht i prezenton paternat më të sofistikuar të diztajnit - kjo do ta bëjë librin shumë të përdorshëm kur ballafaqoheni me sfida të dizajnit në botën reale. Dhe ai e përdor notacionin më të pranuar.

Jam i nderuar që e kam pasë shansin me punu drejtpërdrejt me autorin e këtij libri madhor. Jam kënaqë duke e lexu edicionin e parë, dhe isha i impresionuar kur më luti me rishiku dreftin e këtij edicioni të ri. Jemi taku disa herë dhe kemi shkëmby shumë e-maila. Kam mësuar shumë nga Craig, bile edhe për vetë punën tonë të procesit në Procesin e Unifikuar dhe si me përmirësu atë dhe me

pozicionu në kontekste të ndryshme organizative. Jam i sigurtë se edhe ju do të mësoni shumë, gjatë leximit të këtij libri, edhe nëse tashmë jeni familiarë me OOA/D. Dhe, sikur unë, do ta gjeni veten duke iu kthy librit, për me fresku memorjen tuaj, apo për me fitu mprehtësi të mëtejme nga shpjegimet dhe përvoja e Craig'ut.

Lexim të këndshëm!

Philippe Kruchten

Profesor i Inxhinierimit të Softuerit, Universiteti i British Columbia

më parë,

Partner i Rational dhe Drejtor i Zhvillimit të Procesit për Softuerin RUP Rational

Vancouver, British Columbia

Komente të përkthyesit

Këtu është kapitulli 26 dhe disa pjesë nga kapituj tjerë ku përmenden paternat GoF. Në libër përmenden edhe paterna tjerë, por këtu janë përfshi vetëm paternat GoF.

Preferohet që ky publikim të lexohet pas publikimit "GRASP: Dizajnimi i Objekteve me Përgjegjësi" (botimi i dytë), i cili e përmban një hyrje në OOAD dhe në paterna të dizajnit. Ai publikim po ashtu është përkthim i imi që e përmban kapitullin 17 dhe 25, ku mbulohen të gjitha paternat/principet GRASP.

Edhe këtu gjatë përkthimit e kam përdorë një stil gjuhësor që është më i thjeshtë dhe më i afërt për lexuesin shqiptar në përgjithësi. Në shumë raste i kam thjeshtu fjalitë duke konsideru se lexuesi është programer e jo gjuhëtar, dhe se atij/asaj i intereson informata dhe kuptimi e jo saktësia gjuhësore sintaksore e fjalisë. Dmth. për hir të semantikës më të mirë ka shmangie nga sintaksa dhe drejtshkrimi standard.

Në fund është një shtojcë që, ndër të tjerat, e përmban një fjalor të publikimit ku janë dhënë shpjegime të shkurtra të shkurtesave (akronimeve). Akronimet i kam lënë origjinal për shkak të kuptueshmërisë më të mirë meqë literatura dërmuese është në anglisht.

Pas fjalorit është përmbledhja e principeve/paternave GRASP që mund të shërbejë si referencë e mirë dhe e shpejtë.

Dhe për fund, kur e kam kërku lejen e publikimit nga autori e kam përjetu një befasi të këndshme. Autori, pos që është përgjigjë shpejt (brenda 4 minutave), është përgjigjë shqip:

> (singapore)
>
> dear ridvan,
>
> Po, ju lutemi përdorni atë. Faleminderit për bërjen e përkthimit.
>
> regards, craig
> craig@craiglarman.com
> www.craiglarman.com
> author of:
> *-LATEST: Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*
> *-COMING SOON: Practices for Scaling Lean & Agile Development: Large, Multisite & Offshore Products with Large-Scale Scrum*
> *-Agile and Iterative Development: A Manager's Guide*
> *-Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*

Në vijim është pamja e korrespondencës së shkurtë por efektive. Vëreni se Singapori është 6 orë para nesh.

Re: Permission for translation Monday, July 27, 2009 12:01 PM

From: "Craig Larman" <craig@craiglarman.com>

To: "Ridvan Bunjaku" <rbunjaku@yahoo.com>

(singapore)
dear ridvan,

Po, ju lutemi përdorni atë. Faleminderit për bërjen e përkthimit.

regards, craig
craig@craiglarman.com
www.craiglarman.com
author of:
-LATEST: Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum
-COMING SOON: Practices for Scaling Lean & Agile Development: Large, Multisite & Offshore Products with Large-Scale Scrum
-Agile and Iterative Development: A Manager's Guide
-Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design

On Jul 27, 2009, at 5:57 PM, Ridvan Bunjaku wrote:

> Dear mr. Larman,
> I was delighted to read and study your book, Applying UML and Patterns, Third Edition.
> I found it most interesting chapter 17, GRASP: Designing Objects with Responsibilities, which I translated in Albanian for my personal studying purposes.
> Now, since it's already translated, I ask your permission to publish it in web, for the Albanian community which, unfortunately, lacks Albanian literature on IT field in general, and particularly in software development.
> Please find attached a draft of the translated chapter. I will yet do paging, will add a foreword and some final tunings.
> Of course, I will mention you as author and myself as a translator, and the source book.
>
> Looking forward for your reply.
>
> Best regards,
> Ridvan
>
> ---
>
> |:(|:)
> Ridvan Bunjaku
> Web: http://kalabuli.googlepages.com/
> Shëndet dhe suksese!
> Health and success!
>
> ---
>
> <GRASP draft.pdf>

Ridvan Bunjaku

Prishtinë, 26 gusht 2009

Kapitulli 26. Aplikimi i paternave të dizajnit GoF

Zhvendosja e fokusit (te paternat) ka me pasë efekt të thellë dhe të qëndrueshëm në mënyrën se si i shkruajmë programet.

Ward Cunningham dhe Ralph Johnson

Qëllimet

- M'i prezentu dhe m'i apliku disa paterna GoF të dizajnit
- M'i tregu principet GRASP si përgjithësim i paternave tjerë të dizajnit

Hyrje

Ky kapitull e eksploron dizajnin OO për realizimet e rasteve të përdorimit për studimin e rastit NextGen, duke ofru përkrahje për serviset e jashtme palë të treta, interfejsat e të cilëve munden me ndryshu, për rregulla më komplekse të çmimeve të produkteve, dhe për rregulla të integrueshme (pluggable) të biznesit. Theksi është me tregu se si m'i zbatu paternat Gang-of-Four (GoF) (Banda e Katërshes) dhe paternat më themelorë GRASP. Kapitulli ilustron se dizajni i objekteve dhe ndarja e përgjegjësive munden m'u shpjegu dhe m'u mësu duke u bazu në aplikimin e paternave - një fjalor i principeve dhe i idiomave që mundet m'u kombinu për me dizajnu objekte.

Disa nga 23 paternat GoF të dizajnit prezentohen këtu, por tjera po ashtu mbulohen në kapitujt e mëvonshëm.

Paternat e dizajnit të Bandës së Katërshes (Gang-of-Four)

Paternat GoF të dizajnit, dhe ndikimi i tyre rrënjësor, së pari janë prezentuar në faqe 280 (në libër). Si përsëritje e shpejtë, ato së pari janë përshkru në librin *"Design Patterns"*, një punim me ndikim në themel dhe jashtëzakonisht i njohur që i paraqet 23 paterna të dobishëm gjatë dizajnit të objekteve.

Jo të gjitha 23 paternat janë përdorë gjerësisht; ndoshta 15 janë më të shpeshta dhe më të dobishme.

Për m'u rritë si dizajner i objekteve rekomandohet një studim i thellë i librit *Design Patterns*, megjithëse ai libër supozon se lexuesi tashmë është dizajner OO me eksperiencë të konsiderueshme - dhe ka background në C++ dhe Smalltalk. Në kontrast të tij, ky libër (Applying UML and Patterns) e ofron një hyrje.

26.1. Adapteri (GoF)

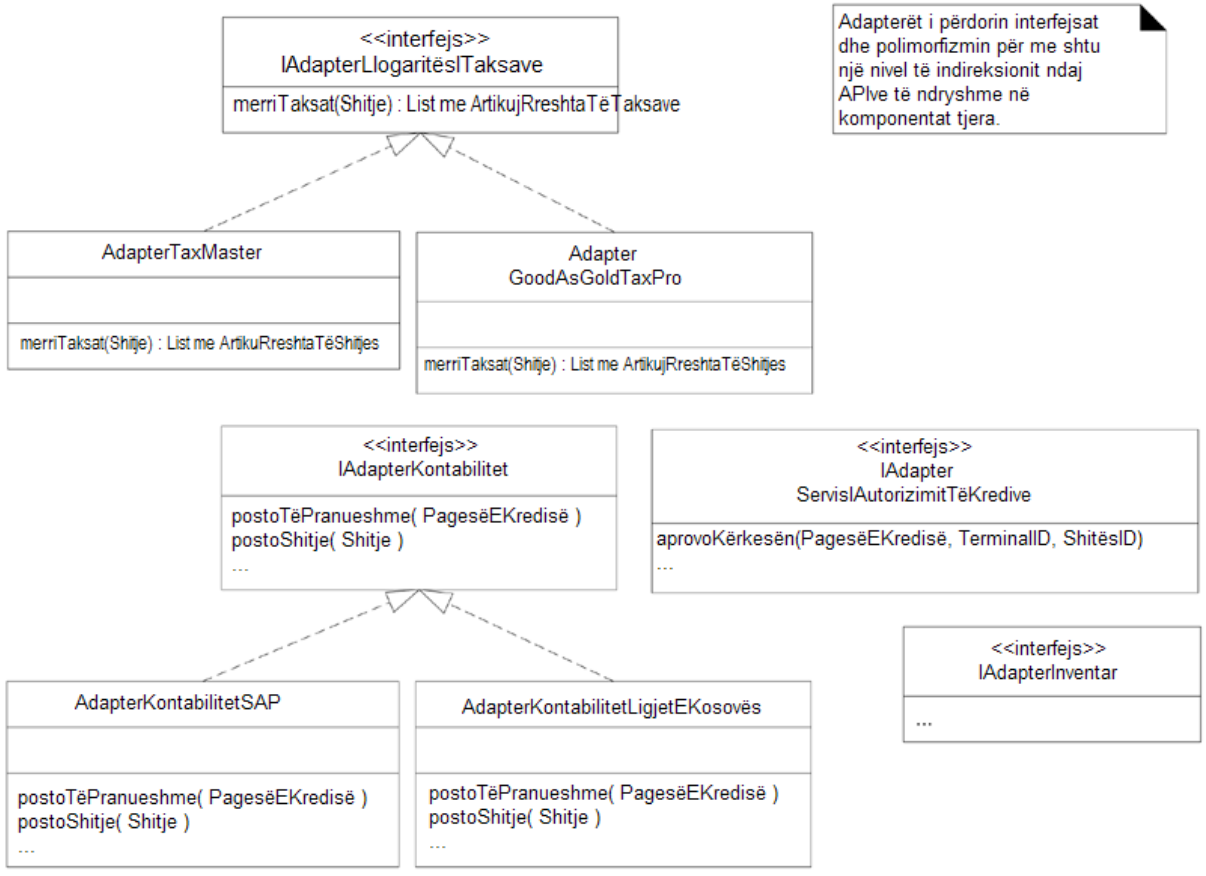
Problemi i NextGen i eksploruar më herët për me motivu paternin Polimorfizmi dhe zgjidhja e tij është më saktësisht shembull i paternit GoF **Adapteri**.

Emri:	**Adapteri**
Problemi:	Si m'i zbërthy interfejsat jokompatibilë, ose me ofru interfejs stabil për komponentat e ngjashme me interfejsa të ndryshëm?
Zgjidhja (këshilla):	Konvertoje interfejsin origjinal të një komponente në një interfejs tjetër, përmes një objekti të ndërmjetëm adapter.

Për me përsëritë: Sistemi NextGen POS ka nevojë m'i përkrahë disa lloje të serviseve palë të treta, duke i përfshi kalkulatorët e taksave, serviset e autorizimit të krediteve, sistemet e inventarit, dhe sistemet e kontabilitetit, ndërmjet tjerash. Secila ka API të ndryshme, që nuk mundet m'u ndryshu.

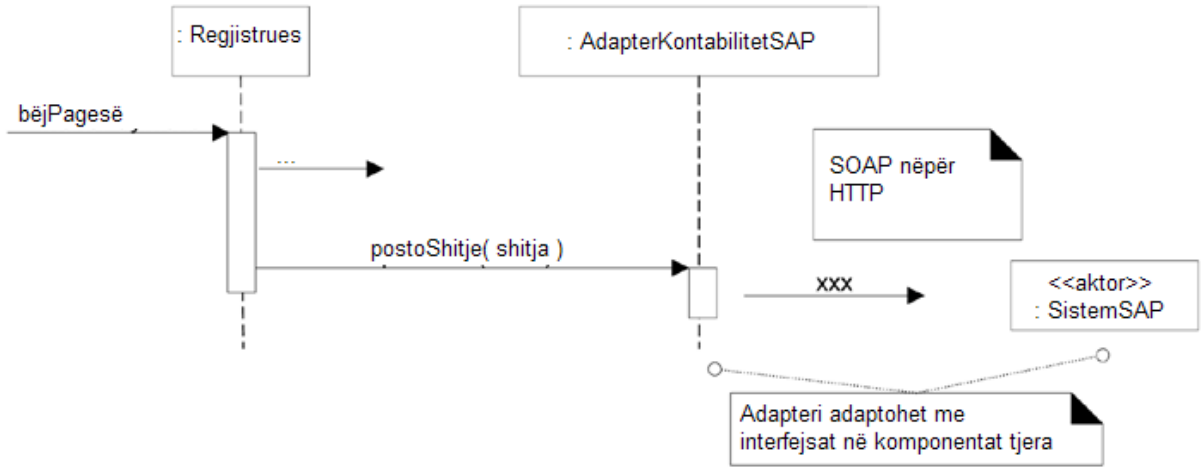
Një zgjidhje është me shtu një nivel të indireksionit me objekte që i adaptojnë interfejsat e ndryshueshëm të jashtëm në një interfejs konsistent që përdoret brenda aplikacionit. Zgjidhja është e ilustruar në Figurën 26.1.

Figura 26.1. Paterni Adapter.



Siç është e ilustruar në Figurën 26.2, do të instancinohet një instancë e veçantë e adapterit për servisin e zgjedhur të jashtëm[1], siç është SAP për kontabilitet, dhe do t'ia adaptojë kërkesën postoShitje interfejsit të jashtëm, siç është një interfejs SOAP XML përmbi HTTPS për një web-servis për intranet të ofruar nga SAP.

Figura 26.2. Përdorimi i një Adapteri.



Udhëzim: Përfshije Paternin në Emrin e Tipit

Vëreje se emrat e tipeve e përfshijnë emrin e paternit "Adapter". Ky është stil relativisht i shpeshtë dhe e ka përparësinë që iu komunikon lehtë të tjerëve që e lexojnë kodin apo diagramet se çfarë paterna të dizajnit po përdoren.

Paternat e ndërlidhur

Një adapter i resurseve që e fsheh një sistem të jashtëm mundet m'u konsideru si objekt Fasadë (një patern tjetër GoF që diskutohet në këtë kapitull), meqë e mbështjell qasjen në nënsistem apo sistem me një objekt të vetëm (gjë që është esenca e Fasadës). Megjithatë, motivimi për me quajtë adapter i resurseve ekziston kur objekti mbështjellës ofron adaptim për interfejsa të ndryshueshëm të jashtëm.

[1] Në Arkitekturën J2EE Connector, këta adapterë për serviset e jashtme quhen më saktësisht adapterë të resurseve.

26.2. Disa principe GRASP si përgjithësim i paternave tjera

Përdorimi i mëhershëm i paternit Adapter mundet m'u shiku si specializim i disa blloqeve ndërtuese GRASP:

> Adapteri i përkrah *Variacionet e Mbrojtura* ndaj ndryshimit të interfejsave të jashtëm apo të paketave palë të treta përmes përdorimit të një objekti *Indireksion* që i aplikon interfejsat dhe *Polimorfizmin*.

Ku është problemi? Mbingarkim me paterna!

Almanaku i Paternave 2000 i liston rreth 500 paterna të dizajnit. Dhe disa qindra më shumë janë publiku prej atëherë. Zhvilluesi kureshtar nuk ka kohë me programu nëse i jipet kjo listë e leximit!

Një zgjidhje: shihi principet themelore

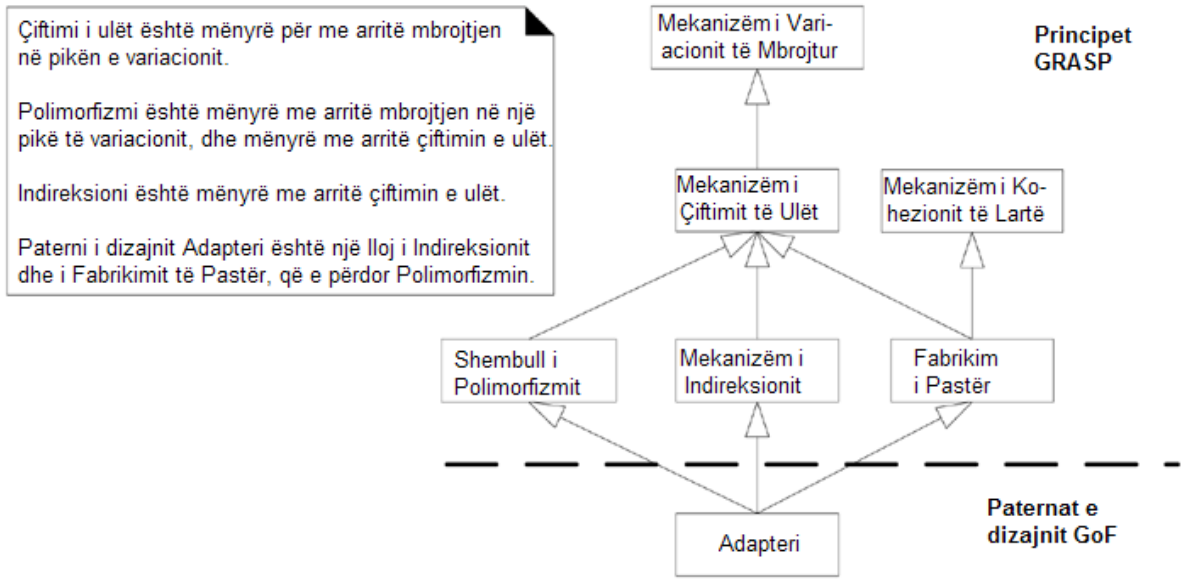
Po, është e rëndësishme për një dizajner me eksperiencë m'i njoftë detalisht dhe përmendësh 50+ nga paternat më të rëndësishëm të dizajnit, por pak nga ne munden m'i mësu apo m'i mbajtë mend 1000 paterna, apo edhe me fillu me organizu atë shumësi të paternave në taksonomi të përdorshme.

Por ka lajme të mira: Shumica e paternave të dizajnit munden m'u pa si specializime të disa principeve themelore GRASP. Edhe pse është vërtet e dobishme m'i studiu paternat e detalizuara të dizajnit për me shpejtu të mësuarit, është edhe më e dobishme m'i pa temat e tyre themelore bazike (Variacionet e Mbrojtura, Polimorfizmi, Indireksioni, ...) për me na ndihmu me depërtu nëpër shumësinë e detaleve dhe me pa "alfabetin" esencial të teknikave të dizajnit që aplikohen.

Shembull: Adapteri dhe GRASP

Figura 26.1. e ilustron poentën time që paternat e detalizuara të dizajnit munden m'u analizu në terma të "alfabetit" themelor të principeve GRASP. Relacionet e përgjithësimit UML janë përdorë për m'i sugjeru lidhjet konceptuale. Në këtë pikë mbase kjo ide duket akademike apo tepër analitike. Por është vërtet rasti ku derisa i kaloni disa vite duke i apliku dhe duke reflektu për shumë paterna të dizajnit, do të ndiheni gjithnjë e më shumë që janë temat themelore ato që janë të rëndësishme, dhe detalet e vogla të Adapterit apo Strategjisë apo të kujtdoqoftë do të bëhen sekondare.

Figura 26.3. Ndërlidhja e Adapterit me disa nga principet thelbësore GRASP



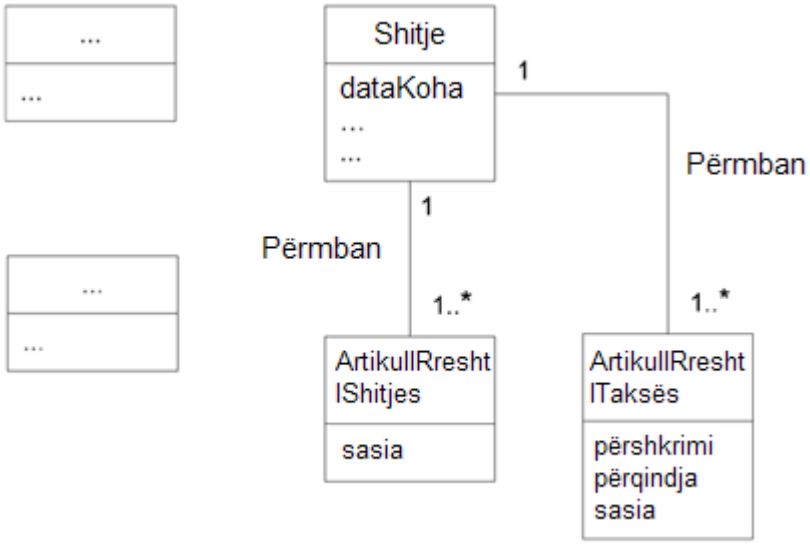
26.3. Zbulimet e "analizës" gjatë dizajnit: modeli i domenit

Vëreni se në dizajnin e Adapterit në Figurën 26.1, operacioni *merriTaksat* e kthen një listë të *ArtikujveRreshtTëTaksave*. Domethënë, gjatë reflektimit më të thellë dhe hulumtimit se si trajtohen taksat dhe si punojnë kalkulatorët e taksave, modeluesi (unë) e kuptoi se një listë e artikujve të rreshtave të taksave janë të asociuara me një shitje, siç është taksa e shtetit, taksa federale, dhe kështu me radhë (gjithmonë ka gjasa se qeveritë do të shpikin taksa të reja!).

Pos që është klasë softuerike e re e krijuar në Modelin e Dizajnit, ky është koncept i domenit. Është normale dhe e shpeshtë me zbulu koncepte të rëndësishme të domenit dhe kuptim të përmirësuar të kërkesave gjatë dizajnit apo programimit - zhvillimi iterativ e përkrah këtë lloj të zbulimit rritës.

A duhet të reflektohet ky zbulim në Modelin e Domenit (apo në Fjalor)? Nëse Modeli i Domenit do të përdoret në të ardhmen si burim i inspirimit për punë të mëvonshme të dizajnit, apo si ndihmë vizuale e të mësuarit për m'i komuniku konceptet kryesore të domenit, atëherë mundet me pasë vlerë me shtu këtë. Figura 26.4 e ilustron një Model të freskuar të Domenit.

Figura 26.4. Modeli i freskuar i pjesshëm i Domenit.



26.4. Fabrika

Kjo quhet edhe **Fabrika e Thjeshtë** apo **Fabrika Konkrete**. Ky patern nuk është patern GoF i dizajnit, por është i shpërndarë jashtëzakonisht. Ai është edhe thjeshtim i paternit GoF Fabrika Abstrakte, dhe shpesh përshkruhet si variacion i Fabrikës Abstrakte, edhe pse kjo nuk është krejt e saktë. Megjithatë, për shkak të popullaritetit të saj dhe asociacionit me GoF, është prezentuar tash.

Adapteri e ngrit një problem të ri në dizajn: Në zgjidhjen e mëparshme të paternit Adapter për serviset e jashtme me interfejsa të ndryshueshëm, kush i krijon adapterat? Dhe si me përcaktu se cilën klasë të adapterit me kriju, siç është *AdapterTaxMaster* apo *AdapterGoodAsGoldTaxPro*?

Nëse i krijon ndonjë objekt i domenit, përgjegjësitë e objektit të domenit po shkojnë përtej logjikës së pastër të aplikacionit (siç janë llogaritjet e totalit të shitjes) dhe nëpër shqetësime tjera të ndërlidhura me lidhshmërinë me komponenta të jashtme të softuerit.

Kjo pikë e nënvizon edhe një princip tjetër fundamental të dizajnit (zakonisht i konsideruar si princip arkitektural i dizajnit): Dizajno për me mirëmbajtë **ndarje të shqetësimeve**. Domethënë, modularizoji apo ndaji shqetësimet e veçanta në zona të ndryshme, ashtu që secila nga to e ka një qëllim koheziv. Në thelb, është aplikim i principit GRASP të Kohezionit të Lartë. Për shembull shtresa e domenit e objekteve të softuerit i thekson përgjegjësitë e logjikës relativisht të pastër të aplikacionit, derisa një grup i tjetër i objekteve është përgjegjës për shqetësimin e lidhshmërisë me sistemet e jashtme.

Prandaj, zgjedhja e një objekti të domenit (siç është një *Regjistrues*) për m'i kriju adapterat nuk e përkrah qëllimin e ndarjes së shqetësimeve, dhe e zvogëlon kohezionin e tij.

Një alternativë e shpeshtë në këtë rast është me apliku paternin **Fabrika**, në të cilën definohet një objekt "fabrikë", që është Fabrikim i Pastër[2], për m'i kriju objektet.

Objektet Fabrikë i kanë disa përparësi:

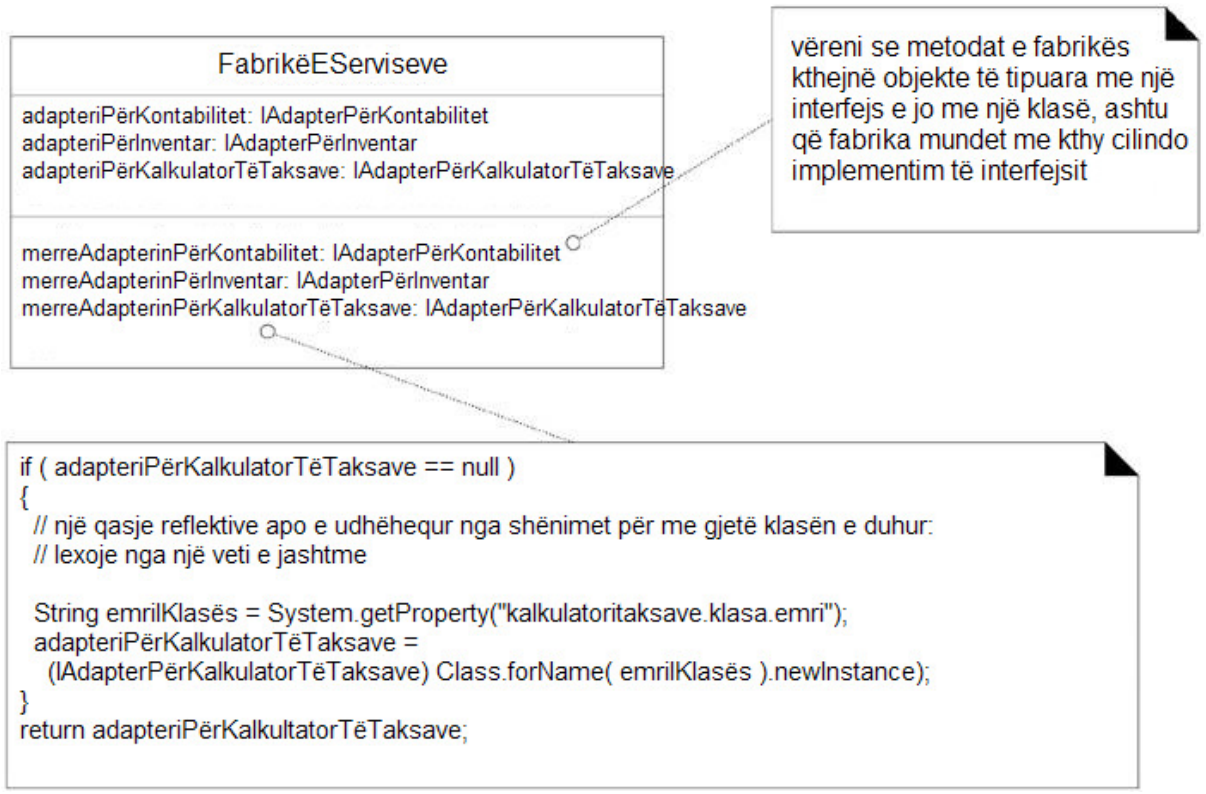
- E veçojnë përgjegjësinë e krijimit kompleks në objekte kohezive ndihmëse
- E fshehin logjikën potencialisht komplekse të krijimit
- Lejojnë futje të strategjive të menaxhmentit të memorjes që e rrit performansën, siç është futja e objekteve në kesh apo reciklimi.

Emri:	**Fabrika**
Problemi:	Kush duhet me qenë përgjegjës për m'i kriju objektet kur ka konsiderata speciale, siç është logjika komplekse e krijimit, një dëshirë për m'i veçu përgjegjësitë e krijimit për kohezion më të mirë, e kështu me radhë?
Zgjidhja (këshilla):	Krijoje një objekt si Fabrikim të Pastër që quhet Fabrikë që e trajton krijimin.

[2] Fabrikë (Factory) = diçka që prodhon, krijon; Fabrikim (Fabrication) = shpikje, trillim (vërejtje e përkthyesit)

Një zgjidhje Fabrikë është ilustruar në Figurën 26.5.

Figura 26.5. Paterni Fabrikë.



Vëreni se në *FabrikëEServiseve*, logjika për me vendosë se cila klasë m'u kriju zbërthehet duke e lexu emrin e klasës nga një burim i jashtëm (për shembull, përmes një vetie të sistemit nëse përdoret Java) dhe pastaj duke e ngarku klasën dinamikisht. Ky është shembull i një **dizajni** pjesërisht **të drejtuar nga shënimet** (data-driven design). Ky dizajn i arrin Variacionet e Mbrojtura në lidhje me ndryshimet në klasën e implementimit të adapterit. Pa e ndryshu kodin burimor në këtë klasë fabrikë, ne mundemi me kriju instanca të klasave të reja adapter duke e ndryshu vlerën e vetisë dhe duke u siguru që klasa e re është e dukshme në Java class path (rruga ku janë klasat e Javës) për ngarkim.

Paternat e ndërlidhur

Fabrikat shpesh qasen me paternin Singleton.

26.5. Singletoni (GoF)

FabrikaEServiseve e ngrit edhe një problem të ri në dizajn: Kush e krijon vetë fabrikën, dhe si i qasemi asaj?

Së pari, vëreni se vetëm një instancë e fabrikës nevojitet brenda procesit. Së dyti, reflektimi i shpejtë sugjeron që metodat e kësaj fabruke mund të nevojitet m'u thirrë nga vende të ndryshme në kod, meqë vendet e ndryshme kërkojnë qasje te adapterët për m'i thirrë serviset e jashtme. Prandaj, është një problem i dukshmërisë: Si me fitu dukshmërinë në këtë instancë të vetme *FabrikëEServiseve*.

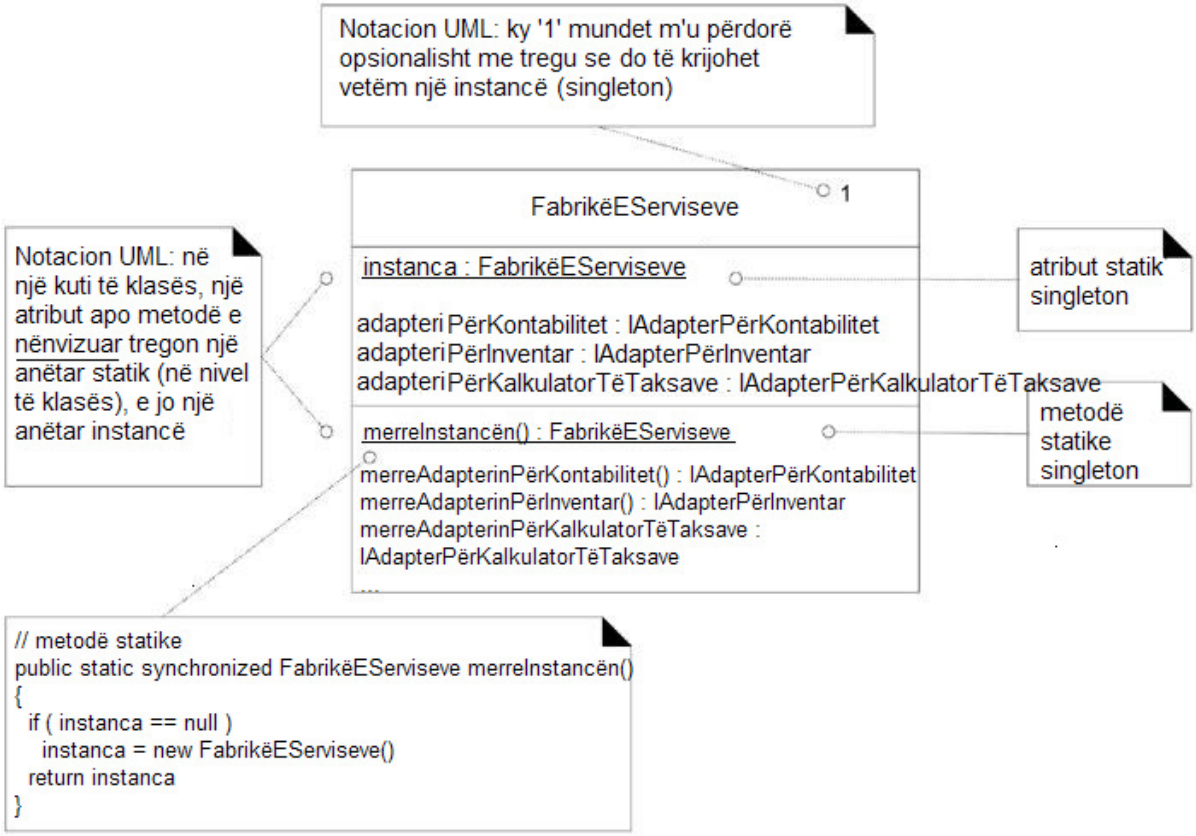
Një zgjidhje është me bartë instancën FabrikëEServiseve përreth si parametër kudo që zbulohet nevojë e dukshmërisë për të, apo m'i inicializu objektet që kërkojnë dukshmëri për të, me një referencë të përhershme. Kjo është e mundur por e papërshtatshme: një alternativë është paterni **Singleton**.

Herë pas here, është e dëshirueshme me përkrahë dukshmëri globale apo një pikë të vetme të qasjes për një instancë të vetme të një klase në vend të ndonjë forme tjetër të dukshmërisë. Kjo është e vërtetë për instancën *FabrikëEServiseve*.

Emri:	**Singleton**
Problemi:	Lejohet saktësisht një instancë e një klase - është "singleton". Objektet e kërkojnë një pikë globale dhe të vetme të qasjes.
Zgjidhja (këshilla):	Definoje një metodë statike të klasës që e kthen singletonin.

Për shembull, Figura 26.6 e tregon një implementim të paternit Singleton.

Figura 26.6. Paterni Singleton në klasën FabrikëEServiseve.



Aplikim i UML: Vëreni se si është ilustru singletoni, me një '1' në këndin e sipërm djathtas të ndarjes së emrit.

Kështu, ideja kryesore është se klasa X e definon një metodë statike *merreInstancën* që vetë e ofron një instancë të vetme të X.

Me këtë qasje, një zhvillues ka dukshmëri globale në këtë instancë të vetme, përmes metodës statike të klasës *merreInstancën*, si në këtë shembull:

```
public class Regjistrues
{
	public void initialize()
	{
		... bëj dicka ...
		// qasja në Fabrikën singleton
		// përmes thirrjes merreInstancën
	adapteriPërKontabilitet =
	FabrikëEServiseve.merreInstancën().merreAdapterinPërKontabilitet();
		... bëj diçka ...
	}

// metoda tjera...

} // fundi i klasës
```

Meqë dukshmëria në klasat publike është globale në skop (në shumicën e gjuhëve), në cilëndo pikë në kod, në cilëndo metodë të cilësdo klasë, dikush mundet me shkru

KlasëSingleton.merreInstancën()

në mënyrë që me siguru dukshmëri në instancën singleton, dhe pastaj me ia dërgu një mesazh, siç është KlasëSingleton.merreInstancën().bëjDiçka. Dhe është vështirë m'i rezistu ndjenjës se mundemi me bëDiçka globalisht!

Çështje të implementimit dhe të dizajnit

Një metodë Singleton merreInstancën thirret shpesh. Në aplikacione me shumë thread'a (fije të ekzekutimit), hapi i krijimit të logjikës së zvarritur të inicializimit është seksion kritik që kërkon kontroll të konkurencës së thread'ave. Kështu, duke supozu se instanca është inicializu në mënyrë të zvarritur (të ngadalshme, përtace, lazy), është e zakonshme me mbështjellë metodën me kontroll të konkurencës. Në Java, për shembull:

```
public static synchronized FabrikëEServiseve merreInstancën()
{
	if ( instanca == null )
	{
		// seksioni kritik nëse është aplikacion me shumë thread'a
		instanca = new FabrikëEServiseve();
	}

	return instanca;
}
```

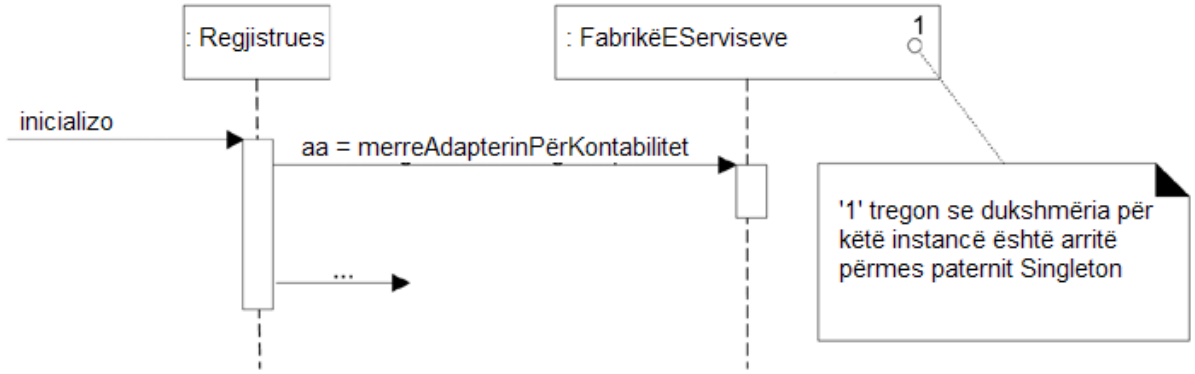
Në temën e inicializimit të zvarritur, pse mos me preferu inicializimin energjik (të shpejtë, të hershëm), si në këtë shembull?

```
public class FabrikëEShërbimeve
{
	// inicializimi energjik
	private static FabrikëEShërbimeve instanca =
		new FabrikëEShërbimeve();

	public static FabrikëEShërbimeve()
	{
		return instanca;
	}

	// metodat tjera...
}
```

Qasja e parë e inicializmit të zvarritur (të vonuar) preferohet zakonisht për të paktën këto arsye:

- Puna e krijimit (dhe ndoshta mbajtja e resurseve "të shtrenjta") shmanget, nëse instanca realisht nuk qaset kurrë.
- Inicializimi i zvarritur merreInstancën nganjëherë përmban logjië komplekse dhe të kushtëzuar të krijimit.

Figura 26.7. Mesazhi implicit merreInstancën i paternit Singleton i treguar në UML për shkak të shenjës '1'



Një tjetër çështje e shpeshtë e implementimit të Singletonit është: Pse mos m'i bë të gjitha metodat e servisit metoda statike të vetë klasës, në vend se me përdorë një objekt instancë me metoda të instancës? Për shembull, çka nëse ia shtojmë një metodë statike të quajtur merreKontabilitetinAdapter klasës FabrikëEServiseve. Për një instancë dhe metodat e instancës zakonisht preferohen për këto arsye:

- Metodat e instancës lejojnë nënklasime dhe përsosje të klasës singleton nëpër nënklasë: metodat statike nuk janë polimorfike (virtuale) dhe nuk e lejojnë mbishkrimin në nënklasa në shumicën e gjuhëve (përjashto Smalltalk).
- Shumica e mekanizmave të komunikimit në distancë të orientuar kah objektet (për shembull, RMI e Java) e përkrahin vetëm aktivizimin në distancë të metodave të instancës, jo të metodave statike. Një instancë singleton mundet m'u aktivizu në distancë, edhe pse kjo pa dyshim bëhet rrallë.
- Një klasë nuk është gjithmonë singleton në të gjitha kontekstet e aplikacionit. Në aplikacionin X, mundet me qenë singleton, por mundet me qenë "multi-ton" në aplikacionin Y. Po ashtu nuk është e pazakonshme me fillu një dizajn duke mendu se objekti ka me qenë singleton, dhe pastaj me zbulu një nevojë për shumë instanca në procesin e njëjtë. Kështu, zgjidhja me instancë e ofron fleksibilitetin.

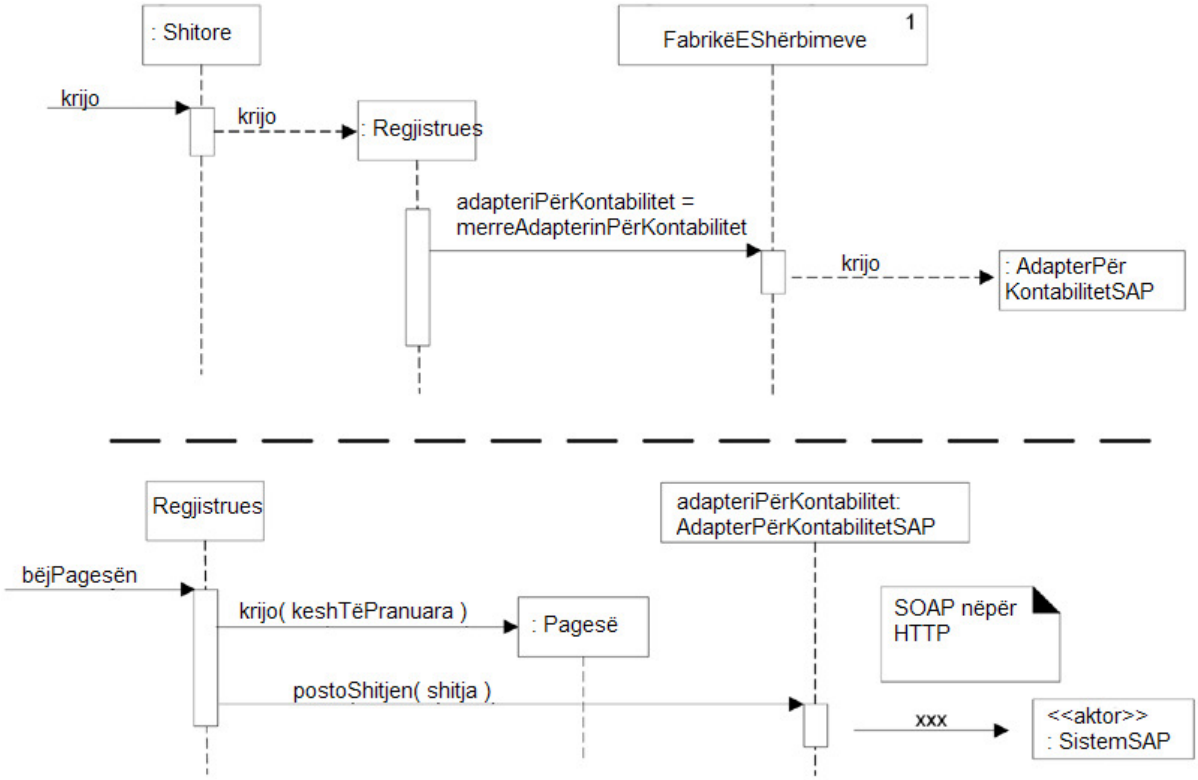
Paternat e ndërlidhura

Paterni Singleton shpesh përdoret për objektet Fabrikë dhe objektet Fasadë - një patern tjetër GoF që do të diskutohet.

26.6. Përmbyllja e problemit të serviseve të jashtme me interfejsa të ndryshueshëm

Një kombinim i paternave Adapter, Fabrikë, dhe Singleton është përdorë për me ofru Variacione të Mbrojtura nga interfejsat e ndryshueshëm të kalkulatorëve të jashtëm të taksave, sistemeve të kontabilitetit, e kështu me radhë. Figura 26.8 e ilustron kontekstin më të gjerë të përdorimit të tyre në realizimin e rasteve të përdorimit.

Figura 26.8. Paternat Adapter, Fabrikë, dhe Singleton të aplikuara në dizajn.



Ky dizajn mundet mos me qenë ideal, dhe gjithmonë ka hapësirë për përmirësim. Por njëri nga qëllimet e dëshiruara në këtë studim të rastit është me ilustru se një dizajn mundet m'u konstruktu nga një grup i principeve apo i "blloqeve të ndërtimit" të paternave, dhe se ka qasje metodike për me bë dhe me shpjegu një dizajn. Është shpresa ime e sinqertë se është e mundur me pa se si dizajni në Figurën 26.8. ka rrjedhë nga rezonimi i bazuar në Kontrolluesin, Krijuesin, Variacionet e Mbrojtura, Çiftimin e Ulët, Kohezionin e Lartë, Indireksionin, Polimorfizmin, Adapterin, Fabrikën, dhe Singletonin.

Vëreni se sa i ngjeshur mundet me qenë një dizajner në bisedë apo në dokumentim kur ka kuptim të përbashkët të paternave. Mundem me thënë, "Për me trajtu problemin e interfejsave të ndryshueshëm për serviset e jashtme, le t'i përdorim Adapterat e gjeneruar nga një Fabrikë Singleton." Dizajnerët e objekteve vërtet kanë biseda që tingëllojnë kështu; përdorimi i paternave dhe i emrave të paternave e përkrah ngritjen e nivelit të abstraksionit në komunikimin e dizajnit.

26.7. Strategjia (GoF)

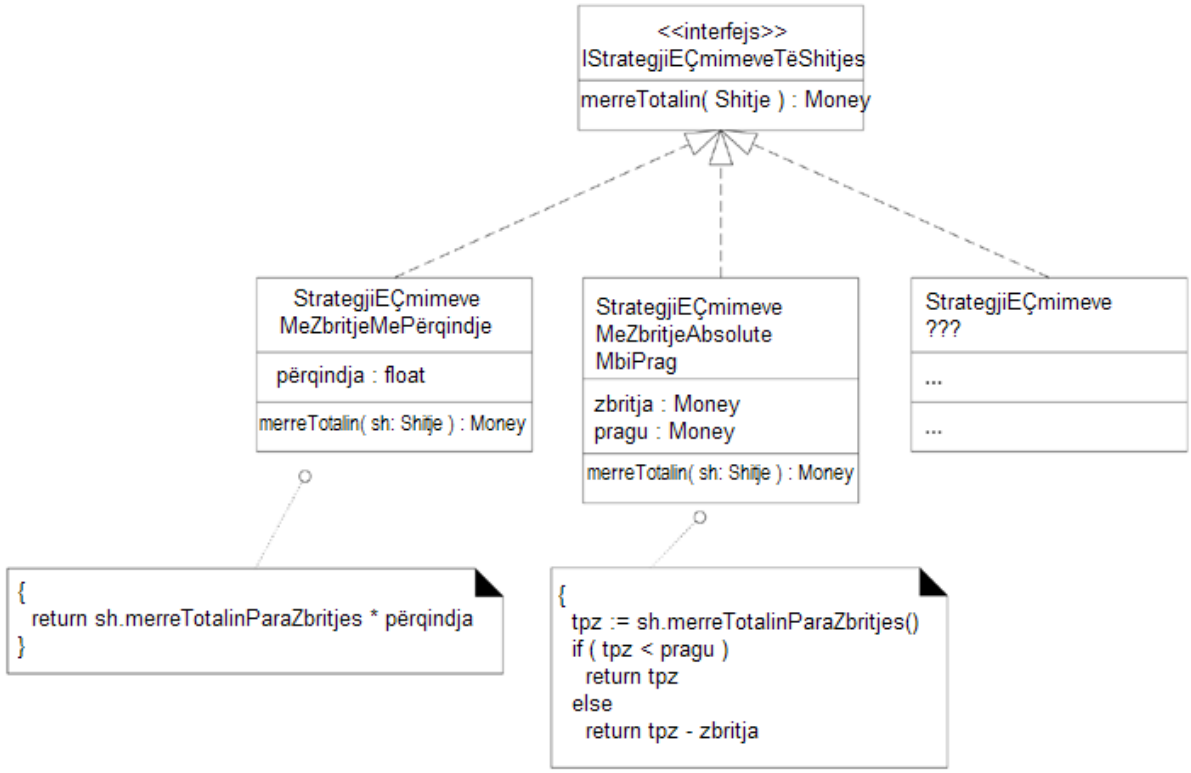
Problemi i ardhshëm i dizajnit që do të zgjidhet është me ofru logjikë më komplekse të çmimeve, siç është zbritja në nivel të shitores për një ditë, zbritja për qytetarët më të moshuar, e kështu me radhë.

Strategjia e çmimeve për një shitje (që mundet m'u quajtë edhe rregull, politikë, apo algoritëm) mundet me ndryshu. Gjatë një periudhe mundet me qenë 10% më lirë për të gjitha shitjet, më vonë mundet me qenë 10 Euro më lirë nëse totali i shitjes është më shumë se 200 Euro, dhe shumë variacione të tjera. Si me dizajnu kësi algoritme të ndryshueshme të çmimeve?

Emri:	**Strategjia**
Problemi:	Si me dizajnu algoritme apo politika të ndryshueshme por të ndërlidhura? Si me dizajnu mundësinë m'i ndërru këto algoritme apo politika?
Zgjidhja (këshilla):	Definoje secilin algoritëm/politikë/strategji në një klasë të veçantë, me një interfejs të përbashkët.

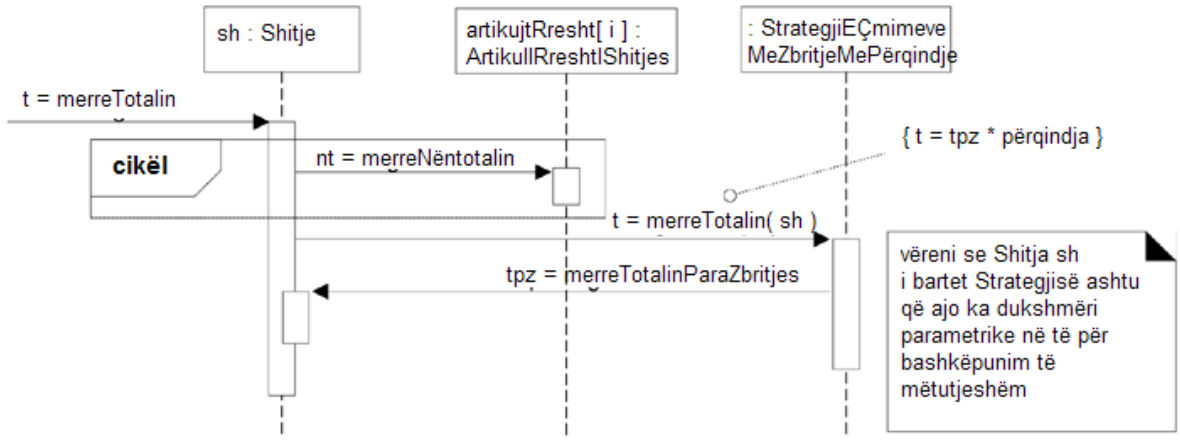
Meqë sjellja e çmimeve ndryshon sipas strategjishë (apo algoritmit), ne krijojmë shumë klasë *StrategjiEÇmimeveTëShitjes*, secila me një metodë polimorfike *merreTotalin* (shih Figurën 26.9). Secila metodë *merreTotalin* e merr objektin *Shitje* si parametër, ashtu që objekti i strategjisë së çmimeve mundet me gjetë çmimin para-zbritjes nga Shitja, dhe pastaj me apliku rregullën e zbritjes. Implementimi i secilës metodë merreTotalin do të jetë i ndryshëm: *StrategjiEÇmimeveMeZbritjeMePërqindje* do të zbresë sipas një përqindjeje, e kështu me radhë.

Figura 26.9. Klasat e Strategjisë së çmimeve.



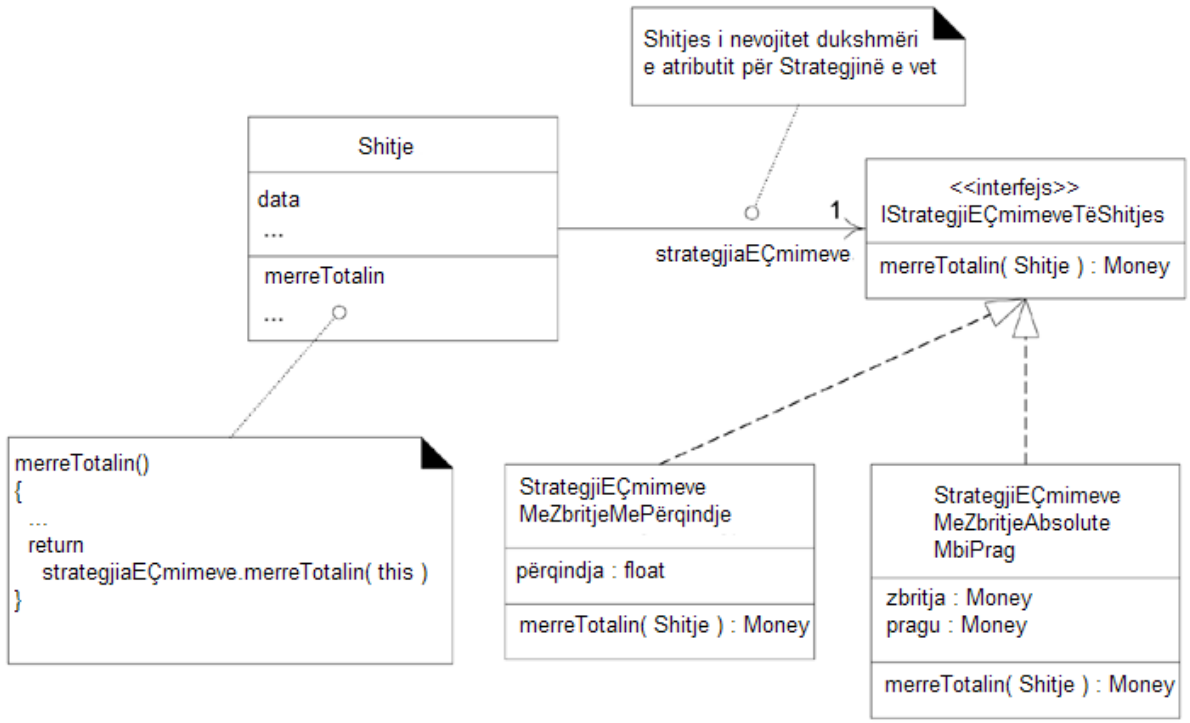
Një objekt strategji i bashkangjitet **objektit të kontekstit** - objektit në të cilin e aplikon algoritmin. Në këtë shembull, objekti i kontekstit është një *Shitje*. Kur një mesazh *merreTotalin* i dërgohet një Shitjeje, ai ia delegon një pjesë të punës objektit të vet strategji, siç është ilustruar në Figurën 26.10. Nuk është e domosdoshme që mesazhi në objektin e kontekstit dhe objekti i strategjisë me pasë emrin e njëjtë, si në këtë shembull (për shembull, merreTotalin dhe merreTotalin), por është e zakonshme. Megjithatë, është e zakonshme - në fakt, zakonisht kërkohet - që objekti i kontekstit me bartë një referencë të vetes (this) në objektin strategji, ashtu që strategjia ka dukshmëri parametrike në objektin e kontekstit, për bashkëpunim të mëtutjeshëm.

Figura 26.10. Strategjia në bashkëpunim.



Vëreni se objekti i kontekstit (Shitja) kërkon dukshmëri të atributit për strategjinë e vet. Kjo reflektohet në DCD-në në Figurën 26.11.

Figura 26.11. Objektit të kontekstit i nevojitet dukshmëri e atributit për strategjinë e vet.



Krijimi i një Strategjie me Fabrikë

Ka algoritme apo strategji të ndryshme të çmimeve, dhe ato ndryshojnë gjatë kohës. Kush duhet me kriju strategjinë? Një qasje e drejtpërdrejtë është me apliku paternin Fabrikë përsëri: Një FabrikëStrategjiEÇmimeve mundet me qenë përgjegjëse për m'i kriju të gjitha strategjitë (të gjitha algoritmet apo politikat e integrueshme) që i nevojiten aplikacionit. Si edhe me FabrikënEServiseve, ajo mundet me lexu emrin e klasës së implementimit të strategjisë së çmimit nga një veti e sistemit (apo ndonjë burim i jashtëm i shënimeve), dhe pastaj me kriju një instancë të saj. Me këtë dizajn të pjesshëm të drejtuar nga shënimet (data-driven), apo dizajn reflektiv, dikush mundet me ndryshu dinamikisht në çfarëdo kohe - derisa aplikacioni NextGenPOS është duke u ekzekutu - politikën e çmimeve, duke e specifiku një klasë apo Strategji tjetër për me kriju.

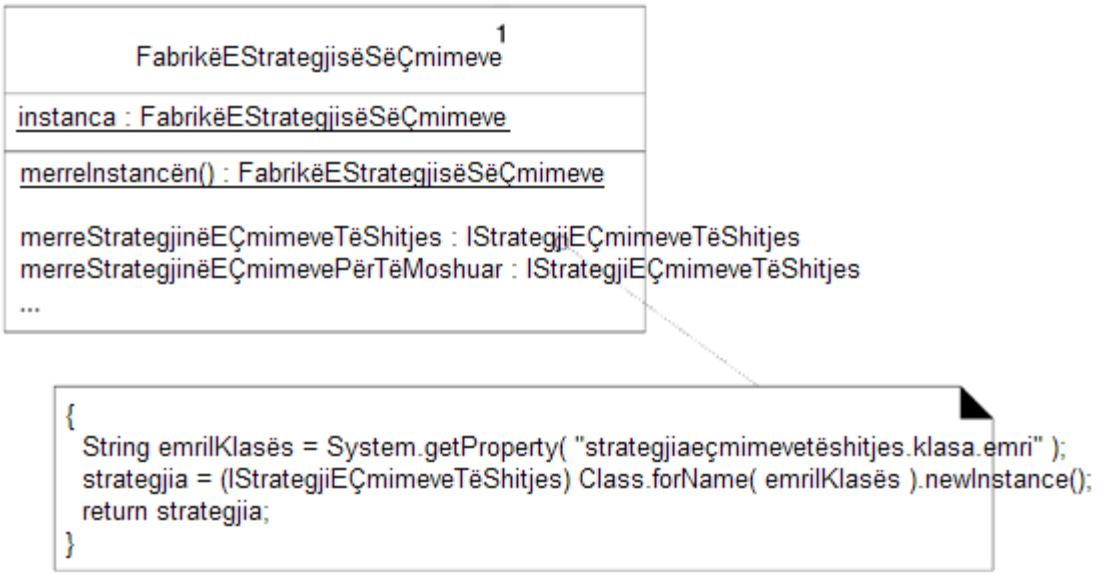
Vëreni se për strategjitë është përdorë një fabrikë e re: domethënë, e ndryshme nga FabrikaEServiseve. Kjo e përkrah qëllimin e Kohezionit të Lartë - secila fabrikë është e fokusuar kohezivisht në krijimin e një familjeje të ndërlidhur të objekteve.

UML Vëreni se në Figurën 26.11 referenca përmes një asociacioni të drejtuar është në interfejsin IStrategjiEÇmimeveTëShitjes, jo në ndonjë klasë konkrete. Kjo tregon se atributi i referencës në Shitje do të jetë i deklaruar në terma të interfejsit, e jo të një klase, ashtu që cilido implementim i interfejsit mundet m'u lidhë me atributin.

Vëreni se për shkak të ndryshimit të shpeshtë të politikës së çmimeve (mundet me qenë çdo orë), nuk është e dëshirueshme me futë në kesh instancën e krijuar të strategjisë në një fushë të FabrikësSëStrategjisëSëÇmimeve, por me rikriju çdo herë, duke e lexu vetinë e jashtme për emrin e saj të klasës, pastaj me instancinu strategjinë.

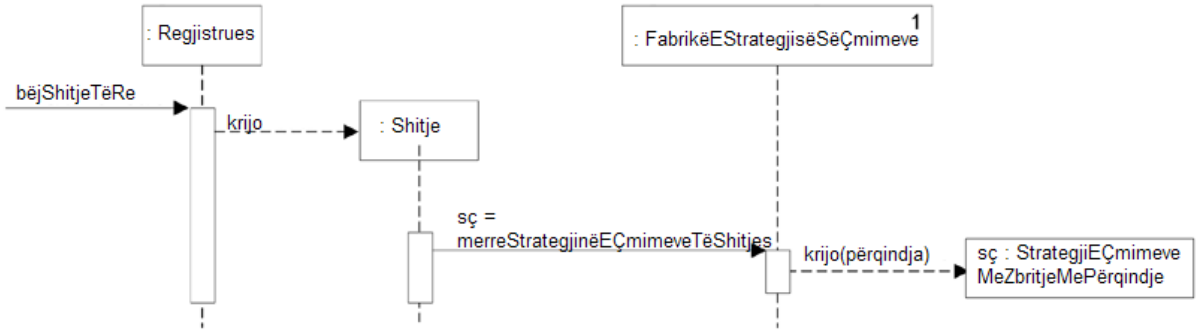
Si te shumica e fabrikave, FabrikaEStrategjiveTëÇmimeve do të jetë singleton (një instancë) dhe do të qaset përmes paternit Singleton (shih Figurën 26.12).

Figura 26.12. Fabrika për strategjitë.



Kur krijohet një instancë Shitje, ajo mundet me kërku nga fabrika strategjinë e vet të çmimeve, siç është treguar në Figurën 26.13.

Figura 26.13. Krijimi i një strategjie.



Leximi dhe Inicializimi i Vlerës së Përqindjes

Përfundimisht, një problem i dizajnit që është injoru deri tash është çështja se si m'i gjetë numrat e ndryshëm për zbritjet me përqindje apo absolute. Për shembull, të hënën, StrategjiaEÇmimeveMeZbritjeMePërqindje mundet me pasë vlerën e përqindjes 10%, por të martën 20%.

Vëreni po ashtu se një zbritje me përqindje mundet me qenë e lidhur me një tip të blerësit, siç është një qytetar i moshuar, e jo me një periudhë kohore.

Këta numra do të vendosen në ndonjë depo të jashtme të shënimeve, siç është një bazë relacionale, ashtu që ata munden m'u ndërru lehtë. Pra, cili objekt ka m'i lexu ata dhe me siguru që i jepen strategjisë? Një zgjedhje e arsyeshme është vetë FabrikaEStrategjive, meqë po e krijon strategjinë e çmimeve, dhe mundet me ditë se cilën përqindje me lexu nga depoja e shënimeve ("zbritja aktuale e shitores", "zbritja për të moshuar", e kështu me radhë).

Dizajnet për m'i lexu këta numra nga depot e jashtme të shënimeve ndryshojnë nga të thjeshtat deri te komplekset, siç është një thirrje e thjeshtë JDBC SQL (në teknologjitë Java, për shembull) apo bashkëpunimi me objekte që shtojnë nivele të indireksionit në mënyrë që me fshehë lokacionin e caktuar, gjuhën e kërkimit të shënimeve, apo tipin e ruajtjes së shënimeve. Analizimi i pikave të variacionit dhe të evolucionit për depon e shënimeve do të zbulojë se a ka nevojë për variacion të mbrojtur. Për shembull, mundemi me pytë, "A jemi të gjithë rehat me zotimin afatgjatë në përdorimin e një baze relacionale që kupton SQL?". Nëse po, një thirrje e thjeshtë JDBC nga brenda FabrikësStrategji mundet me mjaftu.

Përmbledhje

Variacionet e Mbrojtura për politikat e çmimeve që ndryshojnë dinamikisht janë arritë me paternat Strategji dhe Fabrikë. Strategjia ndërtohet mbi Polimorfizëm dhe interfejsa për me leju algoritme të integrueshme në një dizajn të objekteve.

Paternat e ndërlidhur

Strategjia është e bazuar në Polimorfizëm, dhe ofron Variacione të Mbrojtura për ndryshimin e algoritmeve. Strategjitë shpesh krijohen nga një Fabrikë.

26.8. Kompoziti (GoF) dhe principet tjera të dizajnit

Për me ngritë edhe një problem interesant të kërkesave dhe të dizajnit: Si e trajtojmë rastin e shumë politikave konfliktuoze të çmimeve? Për shembull, supozojmë se një shitore i ka politikat vijuese efektive sot (e hënë):

- politika 20% zbritje për të moshuarit
- zbritja e klientit të preferuar prej 15% për shitjet mbi 400 Euro
- të hënën, ka 50 Euro zbritje për blerjet mbi 500 Euro
- bleje një paketë të çajit Darjeeling, merr 15% zbritje të çdo gjëje tjetër

Supozojmë se një i moshuar që është edhe klient i preferuar e blen një paketë të çajit Darjeeling, dhe 600 Euro veggieburgera (qartas, një vegjetarian entuziast që i pëlqen çaji). Çfarë politike e çmimeve duhet m'u apliku?

Për me qartësu: Tash ka strategji të çmimeve që lidhen me shitjen sipas tre faktorëve:

1. periudha kohore (e hënë)
2. tipi i klientit (i moshuar)
3. një produkt i veçantë (çaji Darjeeling)

Një pikë tjetër e qartësimit: Tri nga katër politikat shembuj janë në fakt vetëm strategji të "zbritjes së përqindjes", gjë që e thjeshton pamjen tonë të problemit.

Një pjesë e përgjigjjes ndaj këtij problemi kërkon që shitorja me definu **strategjinë për zgjidhje të konfliktit**. Zakonisht, një shitore e zbaton strategjinë e zgjidhjes së konfliktit "më e mira për klientin" (çmimi më i ulët), por kjo nuk është e domosdoshme, dhe mundet me ndryshu. Për shembull, gjatë një periudhe të vështirë financiare, shitorja mundet m'u detyru me përdorë strategjinë "çmimi më i lartë" për zgjidhje të konfliktit.

Poenta e parë është me vërejtë se munden me ekzistu shumë strategji bashkë-ekzistuese, domethënë, një shitje mundet me pasë disa strategji të çmimit. Një poentë tjetër për me vërejtë është se një strategji e çmimeve mundet m'u ndërlidhë me tipin e klientit (për shembull, i moshuar). Kjo ka implikime të dizajnit të krijimit: tipi i klientit duhet m'u ditë nga FabrikaEStrategjive në kohën e krijimit të një strategjie të çmimeve për klientin.

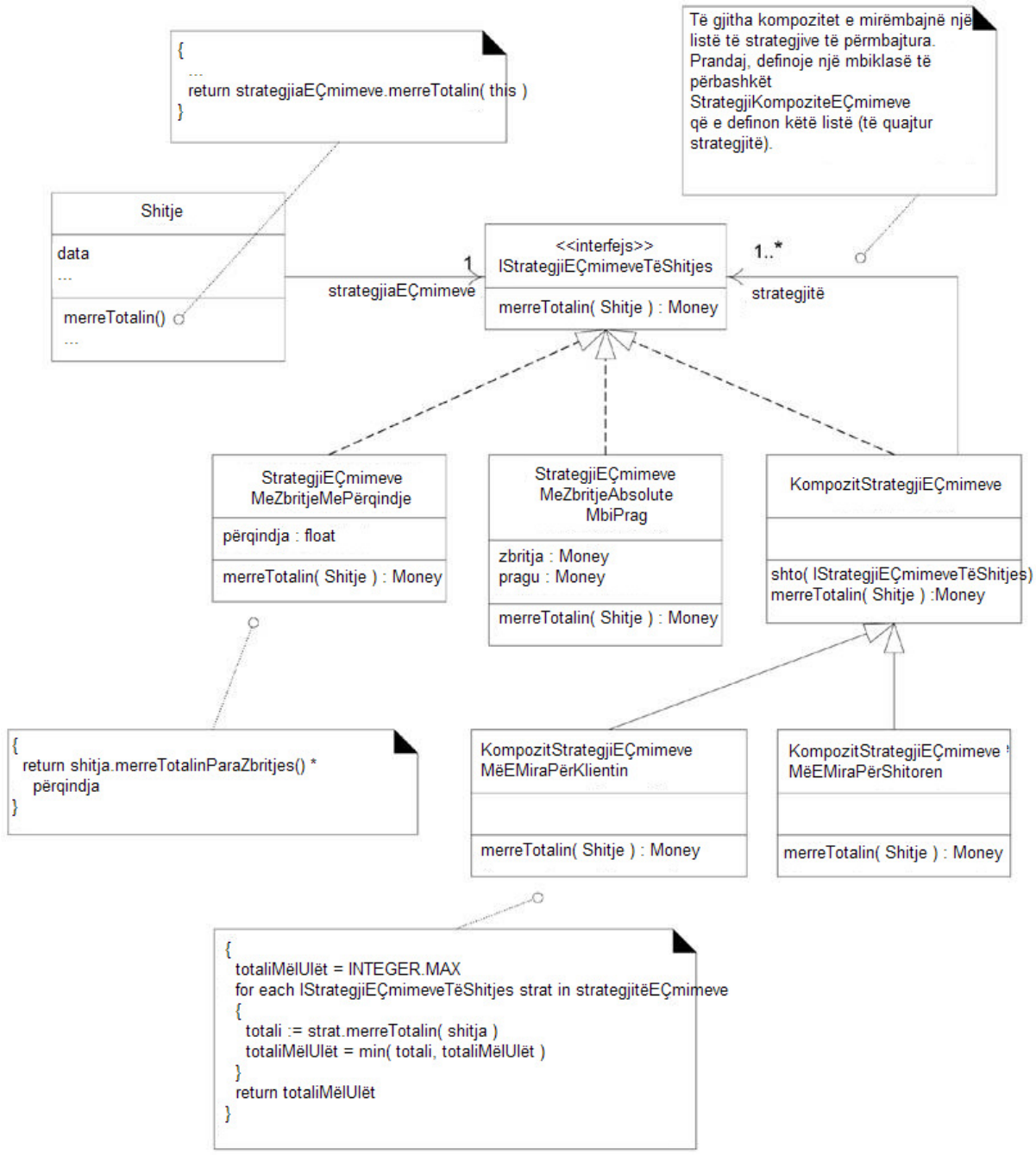
Ngjashëm, një strategji e çmimeve mundet m'u ndërlidhë me tipin e produktit që po blehet (për shembull, çaji Darjeeling). Kjo po ashtu ka implikime të dizajnit të krijimit: PërshkrimiIProduktit duhet m'u ditë nga FabrikaEStrategjive në kohën e krijimit të një strategjie të çmimeve që ndikohet nga produkti.

A ka mënyrë për me ndryshu dizajnin ashtu që objekti Shitje nuk e din se a ka punë me një apo shumë strategji të çmimeve, dhe po ashtu me ofru dizajn për zgjidhjen e konfliktit? Po, me paternin Kompoziti.

Emri:	**Kompoziti**
Problemi:	Si me trajtu një strukturë grup apo kompozicion të objekteve në mënyrë të njëjtë (polimorfikisht) si objekt jo-kompozit (atomik)?
Zgjidhja (këshilla):	Definoji klasat për objektet kompozite dhe atomike ashtu që ato e implementojnë interfejsin e njëjtë.

Për shembull, një klasë e re e quajtur KompozitStrategjiEÇmimeveMëEMiraPërKlientin (epo, të paktën është përshkruese) mundet me implementu interfejsin IStrategjiEÇmimeveTëShitjes dhe vetë me përmbajtë objekte tjera IStrategjiEÇmimeveTëShitjes. Figura 26.14 e shpjegon idenë e dizajnit detalisht.

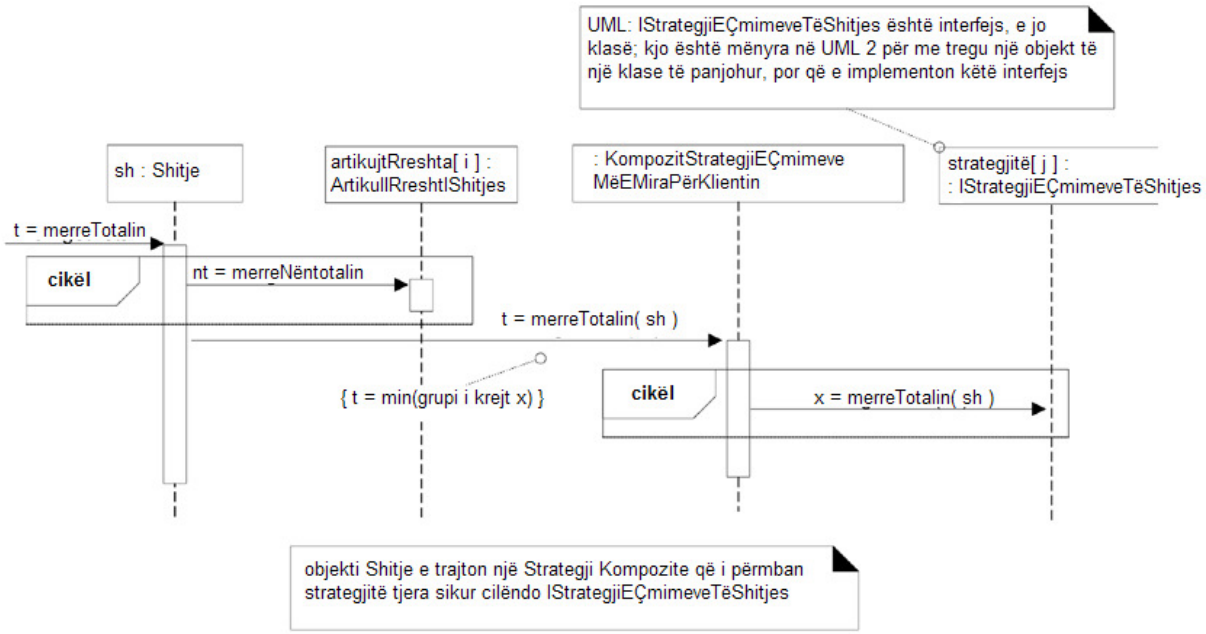
Figura 26.14. Paterni Kompozit.



Vëreni se në këtë dizajn, klasat kompozite siç është KompozitStrategjiEÇmimeveMëEMiraPërKlientin e trashëgojnë një atribut strategjitëEÇmimeve që e përmban një listë të më shumë objekteve IStrategjiEÇmimeveTëShitjes. Kjo është veçori e nënshkrimit (signature) të një objekti kompozit: Objekti i jashtëm kompozit e përmban një listë të objekteve të brendshme, dhe edhe objektet e jashtme edhe objektet e brendshme e implementojnë interfejsin e njëjtë. Domethënë, vetë klasa kompozite e implementon interfejsin IStrategjiEÇmimeveTëShitjes.

Kështu, objektit Shitje mundemi me ia bashkangjitë ose një objekt kompozit *KompozitStrategjiEÇmimeveMëEMiraPërKlientin* (që përmban strategji tjera brenda vetes) ose një objekt atomik *StrategjiZbritjeEÇmimeveMePërqindje*. Është thjesht vetëm edhe një objekt tjetër që e implementon interfejsin *IStrategjiEÇmimeveTëShitjes* dhe e kupton mesazhin *merreTotalin* (Figura 26.15).

Figura 26.15. Bashkëpunimi me një Kompozit.

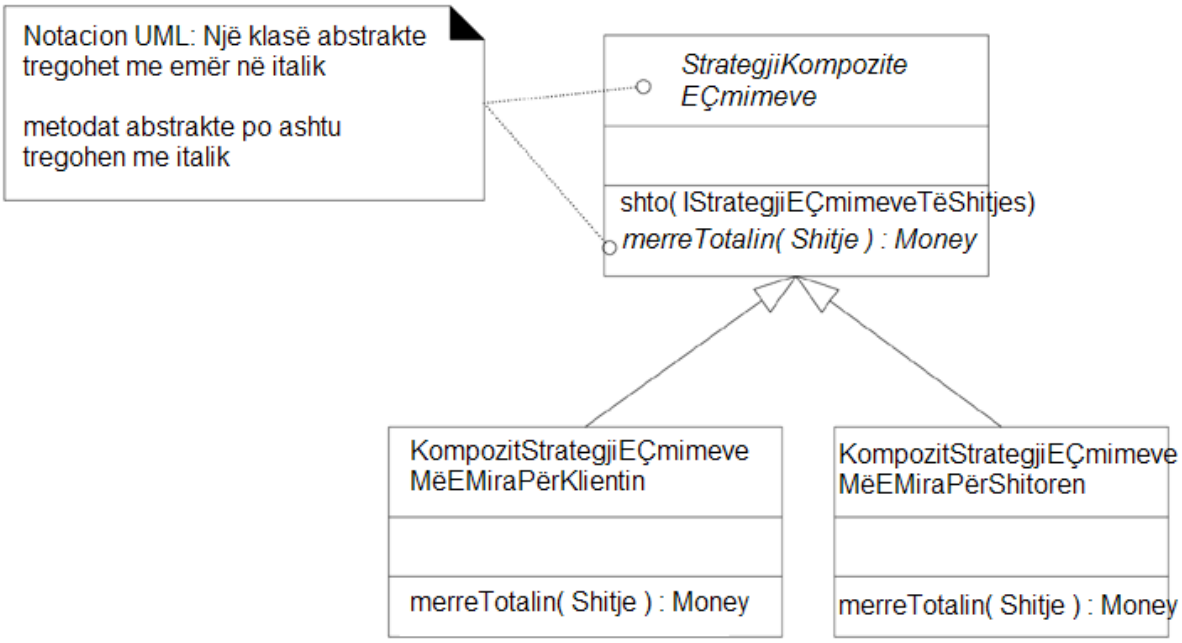


UML Në Figurën 26.15, ju lutem vërejeni një mënyrë për m'i tregu objektet që e implementojnë një interfejs, kur nuk na intereson me specifiku klasën e saktë të implementimit.

Për me qartësu me ca kod shembull në Java, *KompozitStrategjiEÇmimeve* dhe njëra nga nënklasat e saj janë definuar si vijon:

```
// mbiklasë ashtu që të gjitha nënklasat munden me trashëgu
// një List (listë) të strategjive

public abstract class KompozitStrategjiEÇmimeve
	implements IStrategjiEÇmimeveTëShitjes
{
	protected List strategjitë = new ArrayList();

	public shto( IStrategjiEÇmimeveTëShitjes s )
	{
		strategjitë.add( s );
	}

	public abstract Money merreTotalin( Shitje shitja );

} // fundi i klasës


// një Strategji Kompozite që e kthen totalin më të ulët
// të StrategjiveTëÇmimeveTëShitjes të veta të brendshme
public class KompozitStrategjiEÇmimeveMëEMiraPërKlientin
	extends KompozitStrategjiEÇmimeve
{
	public Money merreTotalin( Shitje shitja )
	{
		Money totaliMëIUlët = new Money( Integer.MAX_VALUE );

		// itero nëpër të gjitha strategjitë e brendshme
		for ( Iterator i = strategjitë.iterator(); i.hasNext() )
		{
			IStrategjiEÇmimeveTëShitjes strategjia =
				(IStrategjiEÇmimeveTëShitjes)i.next();
			Money totali = strategjia.merreTotalin( shitja );
			totaliMëIUlët = totali.min( totaliMëIUlët );
		}

		return totaliMëIUlët;
	}

} // fundi i klasës
```

Figura 26.16. Mbiklasat abstrakte, metodat abstrakte, dhe trashëgimia në UML.



Krijimi i shumë StrategjiveTëÇmimeveTëShitjes

Me paternin Kompozit, kemi bërë që një grup i strategjive të shumta (dhe konfliktuoze) të çmimeve t'i duken objektit Shitje si një strategji e vetme e çmimeve. Edhe objekti kompozit që e përmban grupin e implementon interfejsin IStrategjiEÇmimeveTëShitjes. Pjesa më sfiduese (dhe më interesante) e këtij problemi të dizajnit është: Kur i krijojmë këto strategji?

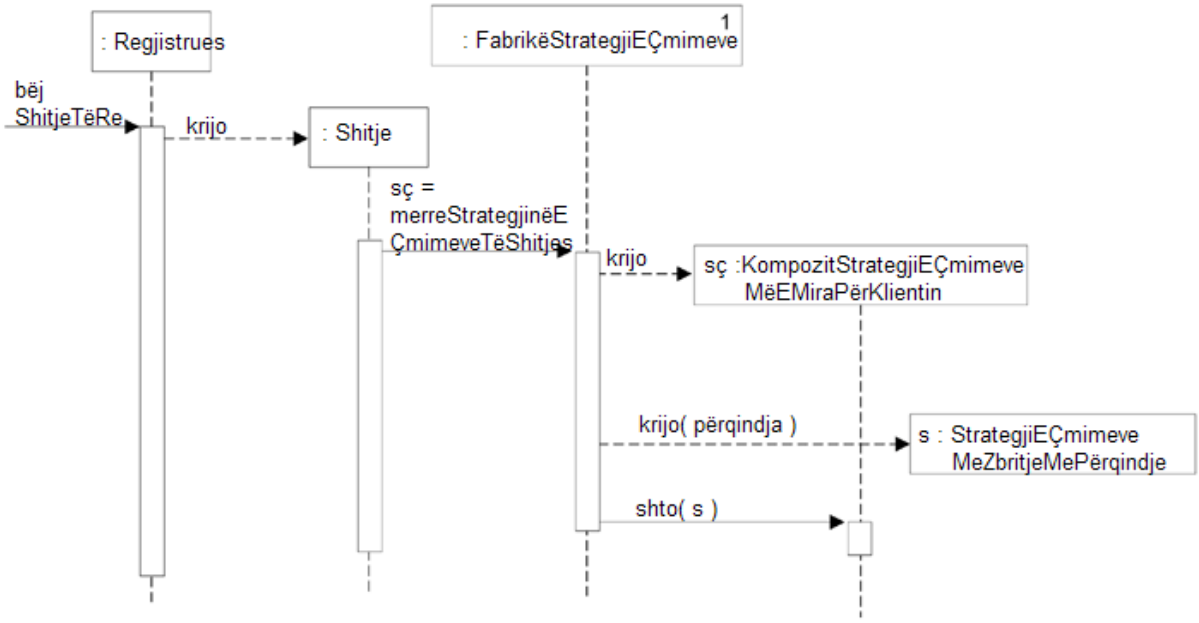
Një dizajn i dëshirueshëm do të fillojë duke e kriju një Kompozit që e përmban politikën e momentit aktual të zbritjes të shitores (që mundet me u caktu në 0% nëse asnjë nuk është aktive), siç është ndonjë StrategjiEÇmimeveMeZbritjeMePërqindje. Pastaj, nëse në ndonjë hap të mëvonshëm në skenar, zbulohet se duhet m'u apliku edhe ndonjë strategji tjetër (siç është zbritja për të moshuar), do të jetë lehtë me ia shtu kompozitit, duke e përdorë metodën e trashëguar *KompozitStrategjiEÇmimeve.shto*.

Janë tri pika në skenar ku munden me iu shtu kompozitit strategjitë e çmimeve:

1. Zbritja aktuale e definuar në shitore, që shtohet kur krijohet shitja
2. Zbritja e tipit të klientit, që shtohet kur tipi i klientit i komunikohet POSit.
3. Zbritja e tipit të produktit (nëse është blerë çaj Darjeeling, 15% zbritje për tërë shitjen), që shtohet kur produkti futet në shitje.

Dizajni i rastit të parë është treguar në figurën 26.17. Si edhe në dizajnin origjinal të diskutuar më herët, emri i klasës së strategjisë për me instancinu mundet m'u lexu si veti e sistemit, dhe një vlerë e përqindjes mundet m'u lexu nga një depo e jashtme e shënimeve.

Figura 26.17. Krijimi i një strategjie kompozite.



Për rastin e dytë të një zbritje të tipit të klientit, së pari rikujtoni zgjerimin e rasit të përdorimit që më herët e ka njoftë këtë kërkesë:

Rasti i përdorimit RP1: PërpunoShitjen

...

Zgjerimet (apo Rrjedhat Alternative):

5b. Klienti thotë se kanë të drejtë në zbritje (p.sh. nëpunës, klient i preferuar):

1. Kasieri e sinjalizon kërkesën për zbritje.
2. Kasieri e fut identifikimin e Klientit
3. Sistemi e paraqet totalin e zbritjes, bazuar në rregullat e zbritjes.

Kjo e tregon një operacion të ri të sistemit në sistemin POS, pos operacioneve *bëjShitjeTëRe*, *futeArtikullin*, *kryejShitjen*, dhe *bëjePagesën*. Do ta quajmë *futeKlientinPërZbritje* këtë operacion të pestë të sistemit; mundet me ndodhë opsionalisht pas operacionit *kryejShitjen*. Kjo nënkupton se një formë e identifikimit të klientit do të duhet të vijë përmes ndërfaqes së përdoruesit, klientiID. Ndoshta mundet m'u marrë nga një lexues i kartelave, apo përmes tastierës.

Dizajni i rastit të dytë është i treguar në Figurën 26.18 dhe Figurën 26.19. Nuk është çudi që objekti fabrikë është përgjegjës për krijimin e strategjisë shtesë të çmimeve. Ajo mundet me bë edhe një

StrategjiTëÇmimeveMeZbritjeMePërqindje që e përfaqëson, për shembull, një zbritje për të moshuar. Por, sikur me dizajnin origjinal të krijimit, zgjedhja e klasës do të lexohet si veti e sistemit, si edhe përqindja specifike për tipin e klientit, për me ofru Variacione të Mbrojtura ndaj ndërrimit të klasës apo vlerave. Vëreni se sipas virtytit të paternit Kompozit, Shitja mundet me pasë dy apo tri strategji konfliktuoze të lidhura për të, por ajo vazhdon të duket si një strategji e vetme për objektin Shitje.

Figura 26.18. Krijimi i strategjisë së çmimeve për një zbritje të klientit, pjesa 1.

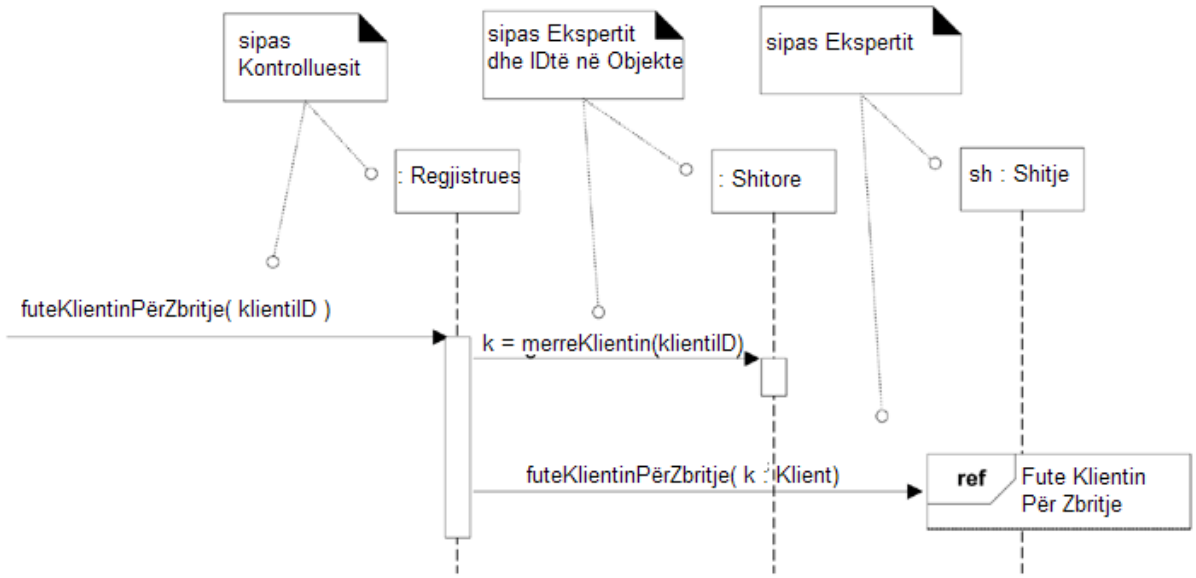
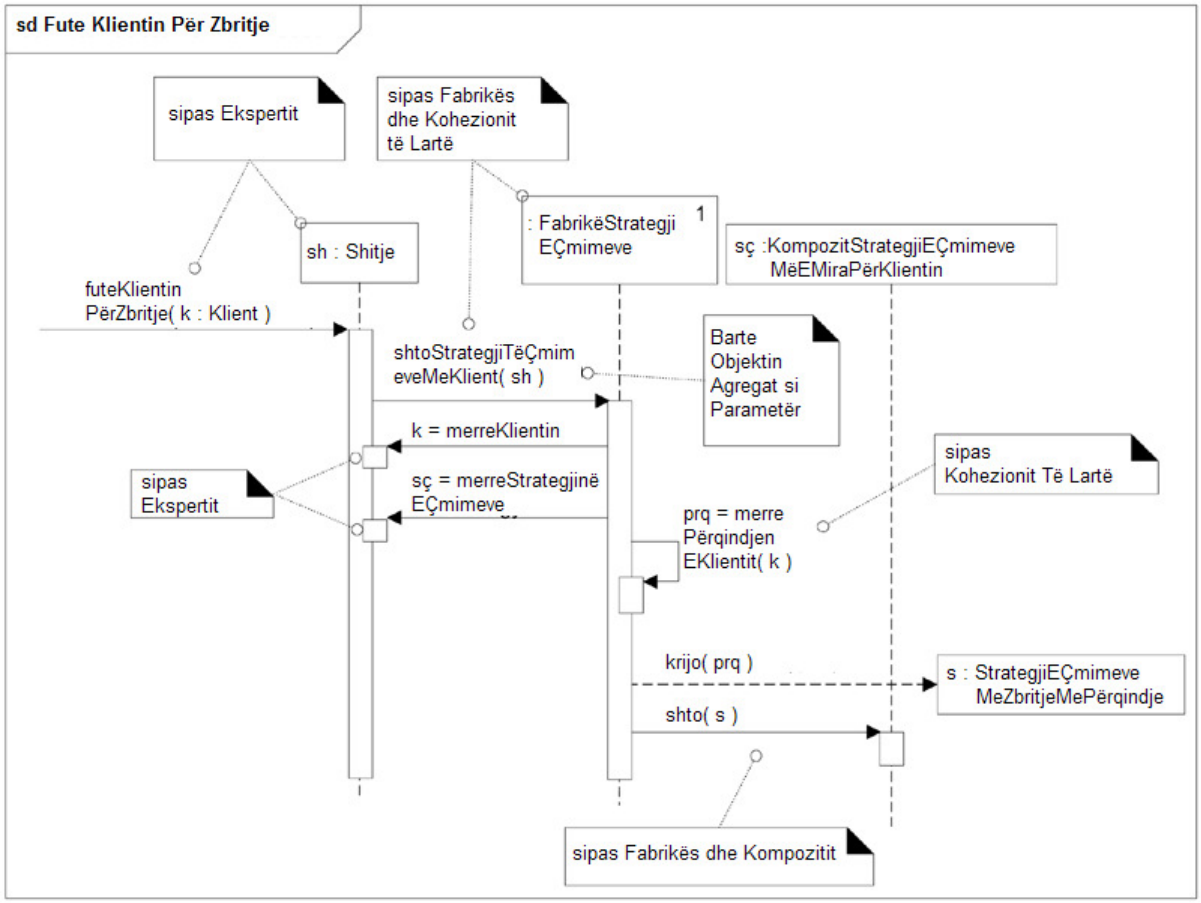


Figura 26.19. Krijimi i strategjisë së çmimeve për një zbritje të klientit, pjesa 2.



UML Figura 26.18 dhe Figura 26.19 e tregojnë një ide të rëndësishme UML 2 në diagrame të interaksionit: Përdorimi i kornizave ref dhe sd[3] për m'i ndërlidhë diagramet.

Konsiderimi i principeve GRASP dhe principeve tjera në dizajn

Për me përsëritë të menduarit në terma të disa paternave themelorë GRASP: Për rastin e dytë, pse mos me ia dërgu Regjistruesi një mesazh FabrikësSëStrategjisëSëÇmimeve, për me kriju këtë strategji të re të çmimeve dhe pastaj me ia bartë Shitjes? Një arsye është me përkrahë Çiftimin e Ulët. Shitja tashmë është e çiftuar me fabrikën; duke e bë edhe Regjistruesin me bashkëpunu me të, çiftimi në dizajn do të rritej. Për më tepër, Shitja është Ekspert i Informatave që e di strategjinë e vet aktuale të çmimeve (që do të ndryshohet); kështu që sipas Ekspertit, po ashtu është e arsyetuar me ia delegu Shitjes.

Vëreni në dizajn se klientiID transformohet në objekt Klient përmes Regjistruesit duke e kërkuar nga Shitorja një Klient, përmes një IDje të dhënë. Së pari, është e justifikueshme me ia dhënë Shitores përgjegjësinë merreKlientin; sipas Ekspertit të Informatave dhe qëllimit të zbraztësisë së ulët të reprezentimit, Shitorja mundet m'i njoftë të gjithë Klientët. Dhe Regjistruesi e pyet Shitoren, sepse

[3]ref = reference (referencë), sd = sequence diagram (diagram sekuencial) (vërejtje e përkthyesit)

Regjistruesi tashmë ka dukshmëri të atributit për Shitoren (nga puna e mëhershme e dizajnit); nëse Shitja do të duhej me pytë Shitoren, Shitjes do t'i nevojitej një referencë në Shitore, duke e rritë çiftimin përtej niveleve të veta aktuale, duke mos e përkrahë kështu Çiftimin e Ulët.

IDtë në Objekte

E dyta, pse me transformu klientinID (një "ID", mbase një numër) në objekt Klient? Kjo është praktikë e shpeshtë në dizajnin e objekteve - m'i transformu çelësat dhe IDtë për gjërat - në objekte të vërteta. Ky transformim shpesh ndodh shpejt pas momentit kur një ID apo çelës futet në shtresën e domenit të Modelit të Dizajnit nga shtresa UI. Nuk ka emër të paternit, por mundet me qenë kandidat për patern meqë është idiomë kaq e shpeshtë ndërmjet dizajnerëve të rryer të objekteve - ndoshta IDtë në Objekte. Pse m'u lodhë? Me pasë një objekt të vërtetë Klient që e enkapsulon një grup të informatave për klientin dhe që mundet me pasë sjellje (të ndërlidhur me Ekspertin e Informatave, për shembull), shpesh bëhet gjë përfituese dhe fleksibile derisa rritet dizajni, edhe nëse dizajneri nuk e percepton fillimisht nevojën për objektin e vërtetë dhe ka mendu se në vend të kësaj një numër i thjeshtë apo ID do të mjaftonte. Vëreni se në dizajnin e mëhershëm, transformimi i artikullitID në objektin PërshkrimIProduktit është edhe një shembull tjetër i këtij paterni IDtë në Objekte.

Barte Objektin Agregat si Parametër

Përfundimisht, vëreni se në mesazhin *shtoStrategjiTëÇmimeveMeKlient(sh : Shitje)* fabrikës ia bartim Shitjen, dhe pastaj fabrika kthehet dhe e kërkon Klientin dhe StrategjinëEÇmimeve nga Shitja.

Pse në vend të kësaj mos m'i ekstraktu thjesht këto dy objekte nga Shitja, dhe me ia bartë fabrikës Klientin dhe StrategjinëEÇmimeve? Përgjigjja është edhe një idiomë tjetër e shpeshtë e dizajnit: Shmangiu ekstraktimit të objekteve fëmijë nga prindi apo objektet agregate, dhe pastaj m'i shëtitë objektet fëmijë. Në vend të kësaj, shëtite objektin agregat që i përmban objektet fëmijë.

Përcjellja e këtij principi e rrit fleksibilitetin, sepse pastaj fabrika mundet me bashkëpunu me krejt SHitjen në mënyra të cilat mund të mos i kemi parashiku si të nevojshme (që është shumë e shpeshtë), dhe si konkluzion, e redukton nevojën me parashiku se çka i nevojitet objektit fabrikë; dizajneri thjesht e bart si parametër komplet Shitjen, pa e ditë se çfarë objekte më specifike i nevojiten fabrikës. Edhe pse kjo idiomë nuk ka emër, është e ndërlidhur me Çiftimin e Ulët dhe Variacionet e Mbrojtura. Ndoshta kish mujtë m'u qujtë paterni Barte Objektin Agregat si Parametër.

Përmbledhje

Ky problem i dizajnit është shtrydhë për shumë udhëzime në dizajnin e objekteve. Një dizajner i shkathtë i objekteve i ka shumë nga këto paterna të ngulitura në memorje përmes studimit të shpjegimeve të tyre të publikuara, dhe i ka internalizu principet thelbësore, siç janë ato të përshkruara në familjen GRASP.

Ju lutem vëreni se edhe pse aplikimi i Kompozitit ishte në një familje Strategji, paterni Kompozit mundet m'u apliku edhe në lloje tjera të objekteve, jo vetëm strategji. Për shembull, është e shpeshtë me kriju komanda "makro komanda" që përmbajnë komanda tjera - përmes përdorimit të Kompozitit. Paterni Komandë është përshkruar në një kapitull të mëvonshëm.

Paternat e ndërlidhur

Kompoziti shpesh përdoret me paternat Strategji dhe Komandë. Kompoziti është i bazuar në Polimorfizëm dhe ofron Variacione të Mbrojtura për një klient ashtu që ai nuk ndikohet nga ajo se a janë objektet e tij atomike apo kompozite.

26.9. Fasada (GoF)

Një tjetër kërkesë e zgjedhur për këtë iteracion janë rregullat e integrueshme (pluggable) të biznesit. Domethënë, në pika të parashikueshme në skenarë, siç është kur ndodh *bëjShitjeTëRe* në rastin e përdorimit *Përpuno Shitjen*, apo kur kasieri fillon me bo pagesën, klientët e ndryshëm që dojnë me ble NextGen POSin kishin dashtë me specifiku (kustomizu) pak sjelljen e tij.

Për me qenë më preciz, supozojmë se dëshirohen rregulla që munden me blloku një aksion. Për shembull:

- Supozojmë se kur krijohet një shitje e re, është e mundur me identifiku se ka m'u pagu nga një vërtetim i dhuratës (kjo është e mundur dhe e shpeshtë). Pastaj, një shitore mundet me pasë rregull me leju m'u ble vetëm një artikull nëse përdoret vërtetim i dhuratës. Si pasojë, operacionet pasuese futeArtikullin, pas të parit, duhet m'u blloku.
- Nëse shitja paguhet nga një vërtetim i dhuratës, blloko të gjitha tipet e kusurit për klientin pos për ndonjë vërtetim tjetër të dhuratës. Për shembull, nëse kasieri kish me kërku kusur në formë të keshit, apo si kredi në llogarinë e klientit në shitore, bllokoji këto kërkesa.
- Supozojmë se është krijuar një shitje e re, është e mundur me identifiku se është për një donacion të bëmirësisë (nga shitorja në shoqatë). Një shitore mundet me pasë edhe rregull me leju futje të artikujve ku secili është më lirë se 250 Euro, dhe po ashtu m'i shtu artikuj shitjes vetëm nëse "kasieri" momental i kyçur është menaxher.

Në terma të analizave të kërkesave, duhet m'u identifiku pikat specifike të skenarit nëpër të gjitha rastet e përdorimit (*futeArtikullin*, *zgjedheKusurinKesh*, ...). Për këtë eksplorim, do të konsiderohet vetëm pika *futeArtikullin*, mirëpo zgjidhja e njëjtë vlen baraz për të gjitha pikat.

Supozojmë se artikekti i zgjidhjes e dëshiron një dizajn që ka ndikim të ulët në komponentat ekzistuese të softuerit. Domethënë, ajo apo ai dëshiron me dizajnu për ndarje të shqetësimeve, dhe me faktorizu këtë trajtim të rregullave në një shqetësim të veçantë. Për më tepër, supozojmë se arkitekti është i pasigurtë për implementimin më të mirë për këtë trajtim të rregullave të integrueshme, dhe mundet me dëshiru me eksperimentu me zgjidhje të ndryshme për paraqitjen, përdorimin, dhe vlerësimin e rregullave. Për shembull, rregullat munden m'u implementu me paternin Strategji, apo me interpretues falas me kod të hapur që e lexojnë dhe e interpretojnë një grup të rregullave IF-THEN, apo me interpretues komercialë të blerë të rregullave, ndërmjet zgjidhjeve tjera.

Për me zgjidhë këtë problem, mundet m'u përdorë paterni Fasadë.

Emri:	**Fasada**
Problemi:	Kërkohet një interfejs i përbashkët, i unifikuar për një grup të pangjashëm të implementimeve apo interfejsave, siç është brenda një nënsistemi. Mundet me pasë çiftim të padëshirueshëm me shumë gjëra në nënsistem, apo implementimi i nënsistemit mundet me ndryshu. Çka të bëjmë?
Zgjidhja (këshilla):	Definoje një pikë të vetme të kontaktit të nënsistemit - një objekt fasadë që e mbështjell nënsistemin. Ky objekt fasadë e paraqet një interfejs të vetëm të unifikuar dhe është përgjegjës për bashkëpunimin me komponentat e nënsistemit.

Një Fasadë është objekt i "përparmë" ("front-end") që është pikë e vetme e hyrjes për serviset e një nënsistemi[4]; implementimi dhe komponentat tjera të nënsistemit janë private dhe nuk munden m'u pa nga komponentat e jashtme. Fasada ofron Variacione të Mbrojtura nga ndryshimet në implementimin e një nënsistemi.

Për shembull, do ta definojmë një nënsistem "motori i rregullave" (rule engine), implementimi specifik i të cilit nuk dihet ende[5]. Ai do të jetë përgjegjës për me vlerësu një grup të rregullave ndaj një operacioni (nga ndonjë implementim i fshehur), dhe pastaj me tregu se a e ka shfuqizu operacionin ndonjëra nga rregullat.

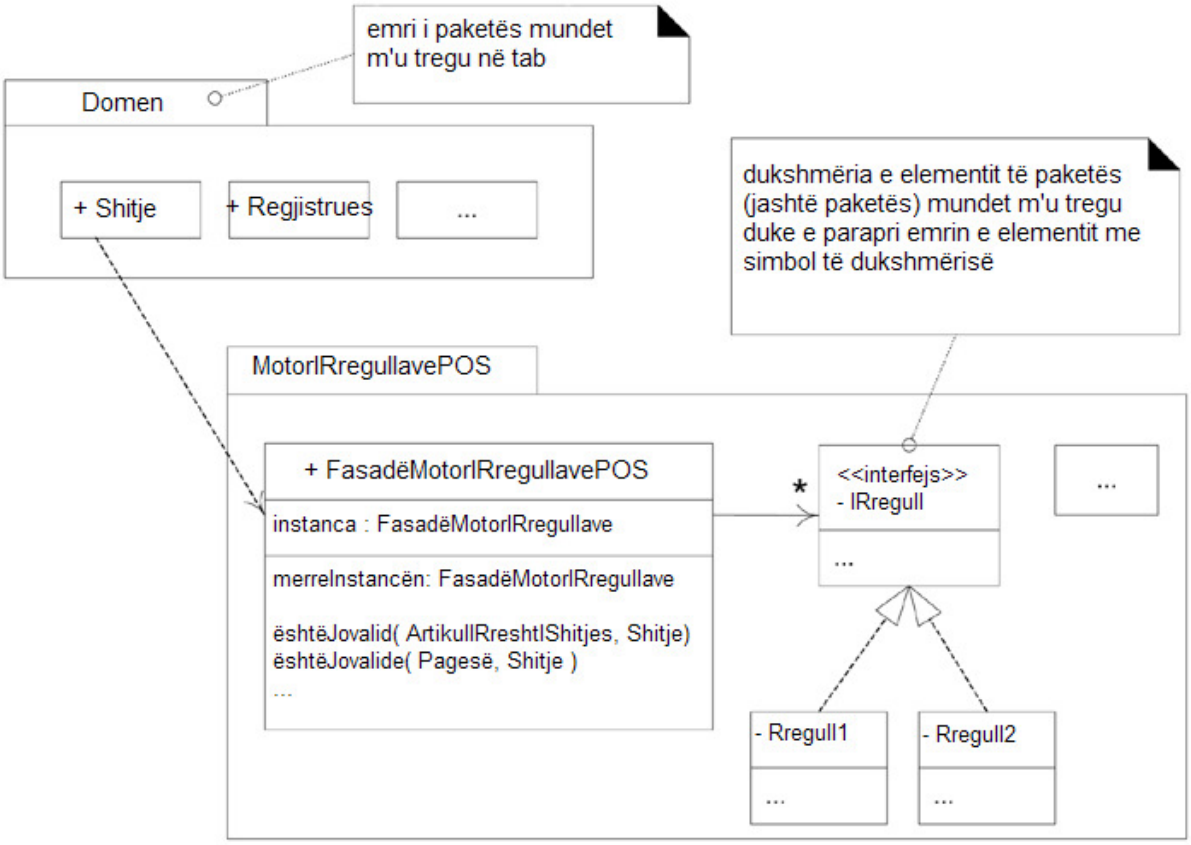
Objekti fasadë në këtë nënsistem do të quhet FasadëMotorIRregullavePOS. Shih Figurën 26.20. Dizajneri vendos m'i vendosë thirrjet në këtë fasadë afër fillimit të metodave që janë definu si pika për rregulla të integrueshme, si në këtë shembull.

```
public class Shitje
{
      public void bëjArtikullTëRreshtit(
            PërshkrimIProduktit përsh, int sasia )
      {
            ArtikullRreshtIShitjes ars = new ArtikullRreshtIShitjes(
                  përsh, sasia );

                  // thirrja në Fasadë
       if ( FasadëMotorIRregullavePOS.merreInstancën().është Jovalid( ars, this ))
                  return;

            artikujtRreshta.shto( ars );
      }
      // ...
} // fundi i klasës
```

[4] "Nënsistemi" këtu është përdorë në sens joformal për me tregu një grupim të veçantë të komponentave të ndërlidhura, e jo si është definuar saktësisht në UML.

[5] Ka disa motorë të rregullave falas me kod të hapur dhe komercialë. Për shembull, Jess, një motor i rregullave falas-për-përdorim-akademik i disponueshëm në http://herzberg.ca.sandia.gov/jess/.

Figura 26.20. Diagrami i paketave UML me një Fasadë.



Vëreje përdorimin e paternit Singleton. Fasadave shpesh iu qasemi përmes Singletonit.

Për këtë dizajn, kompleksiteti dhe implementimi se si do të paraqiten dhe do të vlerësohen rregullat janë fshehur në nënsistemin "motori i rregullave", të qasura përmes fasadës FasadëMotorIRregullavePOS. Vëreni se nënsistemi i fshehur nga objekti fasadë mundet me përmbajtë dhjetëra apo qindra klasë të objekteve, apo edhe zgjidhje jo-të-orientuar-kah-objektet, e prapë ne si klient i nënsistemit, e shohim vetëm pikën e tij të vetme të qasjes.

Dhe deri dikund është arritë një ndarje e shqetësimeve - të gjitha shqetësimet e trajtimit të rregullave i janë delegu një nënsistemi tjetër.

Përmbledhje

Paterni Fasadë është i thjeshtë dhe përdoret gjerësisht. Ai e fsheh një nënsistem prapa një objekti.

Paternat e ndërlidhura

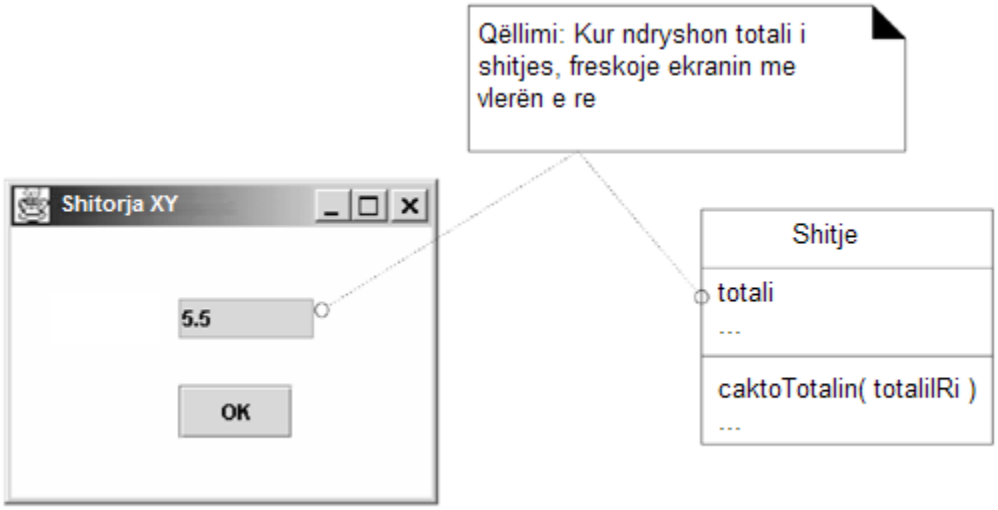
Fasadave zakonisht i qasemi përmes paternit Singleton. Ato ofrojnë Variacione të Mbrojtura nga implementimi i një nënsistemi, duke e shtu një objekt Indireksion për me ndihmu përkrahjen e Çiftimit të Ulët. Objektet e jashtme çiftohen me një pikë në nënsistem: objektin fasadë.

Siç është përshkru në paternin Adapter, një objekt adapter mundet m'u përdorë për me mbështjellë qasjen në sisteme të jashtme me interfejsa të ndryshueshëm. Kjo është një lloj fasade, por theksi është me ofru adaptim për interfejsa të ndryshueshëm, dhe prandaj është quajtë në mënyrë më specifike adapter.

26.10. Vëzhguesi/Publiko-Abonohu/Modeli i Delegimit të Ngjarjeve

Një kërkesë tjetër për interacionin është me shtu aftësinë e një dritare GUI me rifresku pamjen e vet të totalit të shitjes kur ndryshon totali (shih Figurën 26.21). Ideja është me zgjidhë problemin për këtë rast të vetëm, dhe pastaj në iteracionet e mëvonshme, me zgjeru zgjidhjen në rifreskimin e ekranit GUI edhe për shënime tjera ndryshuese.

Figura 26.21. Freskimi i ndërfaqes kur ndryshon totali i shitjes.



Pse mos me bë zgjidhjen si vijon? Kur Shitja e ndryshon totalin e vet, objekti Shitje ia dërgon një mesazh dritares, duke i thënë me rifresku pamjen e vet.

Për me përsëritë, principi i Ndarjes Model-Pamje i kundërshton këto zgjidhje. Ai thotë se objektet "model" (objektet jo-UI siç është Shitja) nuk duhet me ditë për objektet pamje apo prezentuese siç është një dritare. Ai e promovon Çiftimin e Ulët nga shtresat tjera në shtresën e prezentimit (UI) të objekteve.

Një pasojë e përkrahjes së këtij çiftimi të ulët është se ai e lejon ndërrimin e pamjes apo shtresës së prezentimit me një të re, apo të disa dritareve me dritare të reja, pa ndiku në objektet jo-UI. Nëse objektet model nuk dinë për objektet Java Swing (për shembull), atëherë është e mundur me shkyçë një ndërfaqe Swing, apo me shkyçë një dritare të caktuar, dhe me kyçë diçka tjetër.

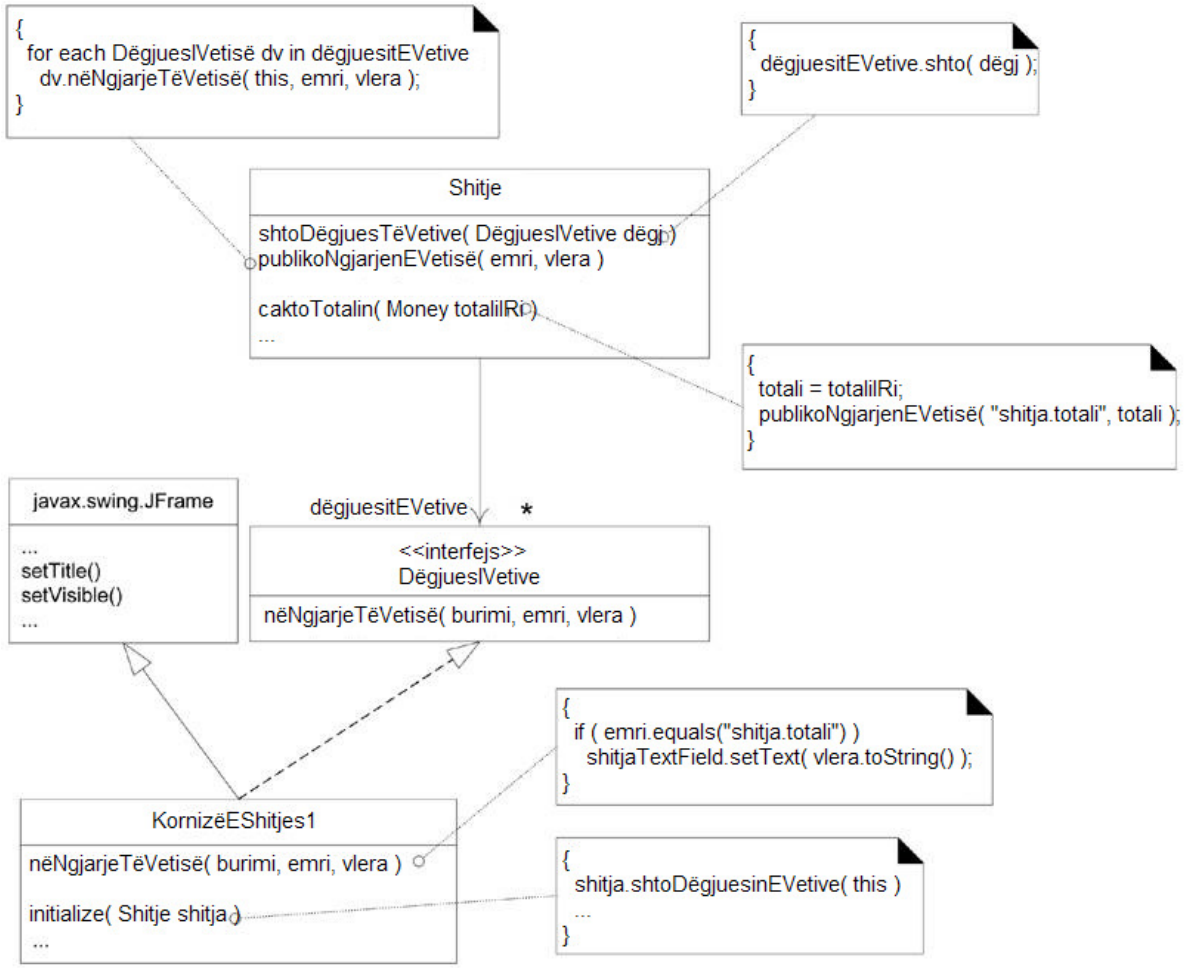
Kështu, Ndarja Model-Pamje i përkrah Variacionet e Mbrojtura ndaj ndryshimit të ndërfaqes së përdoruesit.

Për me zgjidhë këtë problem të dizajnit, mundet m'u përdorë paterni Vëzhgues.

Emri:	**Vëzhguesi (Publiko-Abonohu)**
Problemi:	Lloje të ndryshme të objekteve abonuese janë të interesuara në ndryshimet e gjendjes apo ngjarjet e një objekti publikues, dhe duan me reagu në mënyrën e tyre unike kur publikuesi e prodhon një ngjarje. Për më tepër, publikuesi don me mbajtë çiftim të ulët me abonentët. Çka të bëjmë?
Zgjidhja (kështilla):	Definoje një interfejs "abonues" apo "dëgjues". Abonentët e implementojnë këtë interfejs. Publikuesi mundet m'i regjistru dinamikisht abonentët që janë të interesuar në një ngjarje dhe m'i njoftu ata kur ndodh një ngjarje.

Një zgjidhje shembull është e përshkruar detalisht në Figurën 26.22.

Figura 26.22. Paterni Vëzhguesi (Observer).



Idetë dhe hapat kryesorë në këtë shembull:

1. Definohet një interfejs; në këtë rast, *DëgjuesIVetive* me operacionin *nëNgjarjeTëVetisë*.

2. Definoje dritaren për me implementu interfejsin.
   - *KornizëEShitjes1* ka me implementu metodën *nëNgjarjeTëVetisë*.

3. Kur dritarja *KornizëEShitjes1* inicializohet, bartja instancën *Shitje* nga e cila po e shfaq totalin.

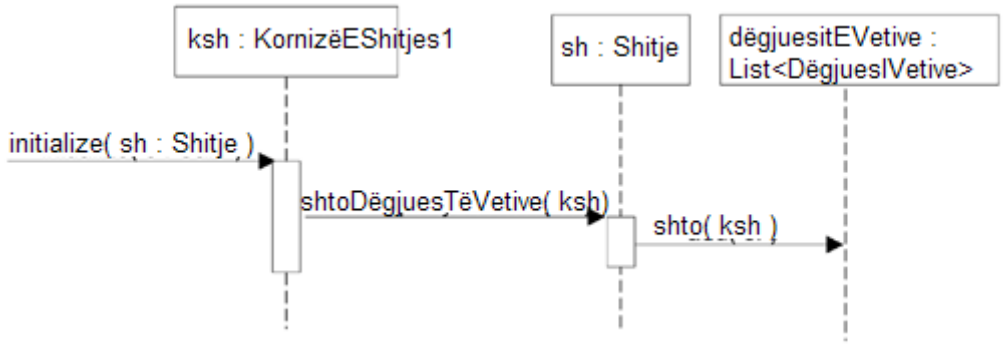
4. Dritarja *KornizëEShitjes1* regjistrohet apo abonohet në instancën *Shitje* për njoftimin e "ngjarjes së vetisë", përmes mesazhit *shtoDëgjuesTëVetisë*. Domethënë, kur një veti (siç është totali) ndryshon, dritarja don m'u njoftu.

5. Vëreni se *Shitja* nuk din për objektet *KornizëEShitjes1*; në vend të kësaj, ajo din vetëm për objektet që e implementojnë interfejsin *DëgjuesIVetive*. Kjo e zvogëlon çiftimin e *Shitjes* me dritaren - çiftimi është vetëm me një interfejs, jo me një klasë GUI.

6. Pra instanca *Shitje* është publikues i "ngjarjeve të vetive". Kur ndryshon totali, ajo iteron nëpër tërë *DëgjuesitEVetive* të abonuar, duke e njoftu secilin.

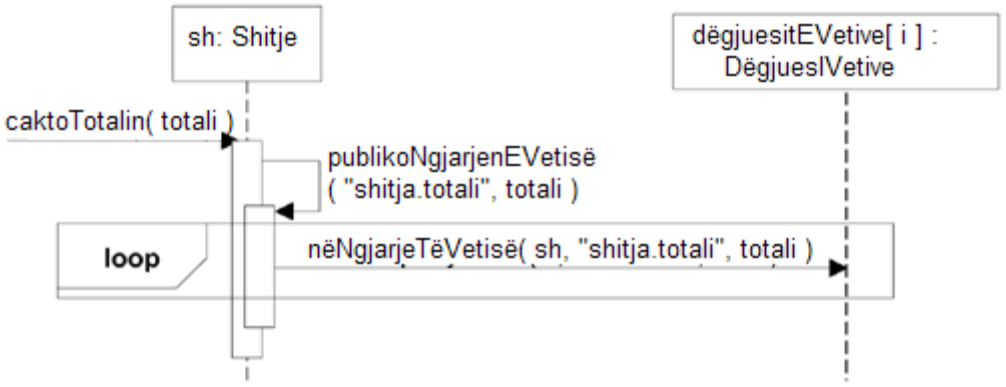
Objekti *KornizëEShitjes1* është vëzhguesi/abonenti/dëgjuesi. Në Figurën 26.23, ai abonohet me interesimin në ngjarjet e vetive të Shitjes, që është publikues i vetive të shitjes. *Shitja* e shton objektin në listën e vet të abonentëve. Vëreni se *Shitja* nuk din për *KornizënEShitjes1* si objekt *KornizëEShitjes1*, por vetëm si objekt *DëgjuesIVetive*; kjo e zvogëlon çiftimin nga modeli telart në shtresën e pamjes.

Figura 26.23 Vëzhguesi *KornizëEShitjes1* abonohet në publikuesin *Shitje*.



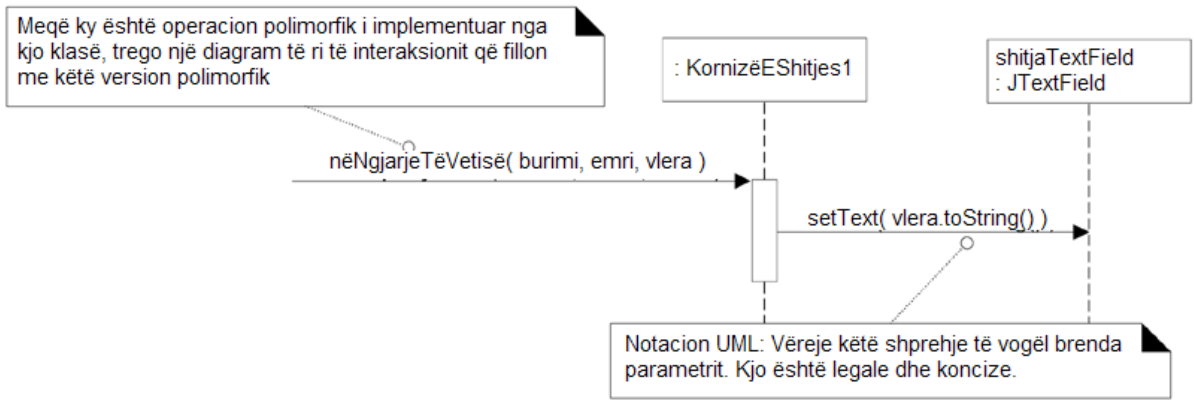
Siç është ilustruar në Figurën 26.24, kur ndryshon totali i Shitjes, ajo iteron nëpër tërë abonentët e vet të regjistruar, dhe "e publikon një ngjarje" duke ia dërgu secilit mesazhin nëNgjarjeTëVetisë.

Figura 26.24. Shitja ua publikon të gjithë abonentëve të vet një ngjarje të vetisë.



Aplikim i UML: Vëreje qasjen për trajtimin e mesazheve polimorfike në një diagram të interaksionit, në Figurën 26.24. Mesazhi *nëNgjarjeTëVetisë* është polimorfik; rastet specifike të implementimit polimorfik do të tregohen në diagrame tjera, si në Figurën 26.25.

Figura 26.25. Abonenti KornizëEShitjes1 e pranon njoftimin e një ngjarjeje të publikuar.



KornizaEShitjes1, që e implementon interfejsin DëgjuesIVetive, pra e implementon edhe një metodë *nëNgjarjeTëVetisë*. Kur *KornizaEShitjes1* e pranon mesazhin, ajo ia dërgon një mesazh objektit të vet GUI JTextField për m'u rifresku me totalin e ri të shitjes. Shih Figurën 26.25.

Në këtë patern, ende ka çiftim nga objekti model (Shitje) me objektin pamje (KornizëEShitjes1). Por është çiftim i ulët me një interfejs të pavarur nga shtresa e prezentimit - interfejsi DëgjuesIVetive. Dhe dizajni nuk kërkon kurrfarë objekti abonues me qenë i regjistruar te publikuesi (s'ka nevojë me dëgju asnjë objekt). Domethënë, lista e DëgjuesveTëVetive të regjistruar në Shitje mundet me qenë e zbrazët. Si përmbledhje, çiftimi me një interfejs të përgjithshëm të objekteve që nuk duhet me qenë patjetër të pranishëm, dhe që munden m'u shtu (apo m'u heqë) dinamikisht, e përkrah çiftimin e ulët. Prandaj, Variacionet e Mbrojtura ndaj ndryshimit të ndërfaqes së përdoruesit janë arritë përmes përdorimit të një interfejsi dhe të polimorfizmit.

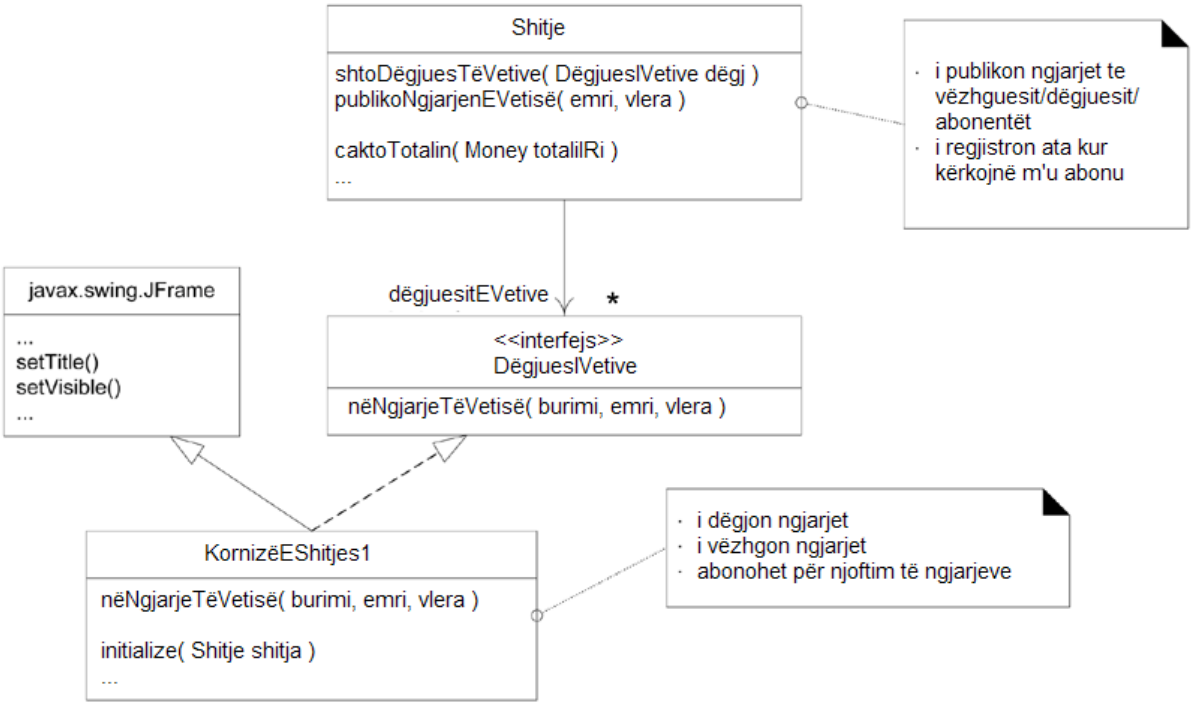
Pse quhet Vëzhgues, Publiko-Abonohu, apo Model i Delegimit të Ngjarjeve (Observer/Publish-Subscribe/Delegation Event Model)?

Fillimisht, kjo idiomë është quajtë publiko-abonohu, dhe ende është e njohur gjerësisht me atë emër. Një objekt "publikon ngjarje", siç është Shitja që e publikon "ngjarjen e vetisë" kur ndryshon totali. Asnjë objekt mundet mos me qenë i interesuar në këtë ngjarje, me ç'rast, Shitja nuk ka abonentë të regjistruar. Por objektet që janë të interesuara, "abonohen" apo regjistrohen me interesimin në një ngjarje duke kërku nga publikuesi m'i njoftu ata. Kjo është bërë me mesazhin *Shitje.shtoDëgjuesTëVetisë*. Kur ngjarja ndodh, abonentët e regjistruar njoftohen me një mesazh.

Është quajtë Vëzhgues sepse dëgjuesi apo abonenti po e vëzhgon ngjarjen; ai term ishte i popullarizuar në Smalltalk në 1980tat e hershme.

Është quajtë Model i Delegimit të Ngjarjeve (në Java) sepse publikuesi ua delegon trajtimin e ngjarjeve "dëgjuesve" (abonentëve; shih Figurën 26.26).

Figura 26.26. Kush është vëzhguesi, dëgjuesi, abonenti, dhe publikuesi?



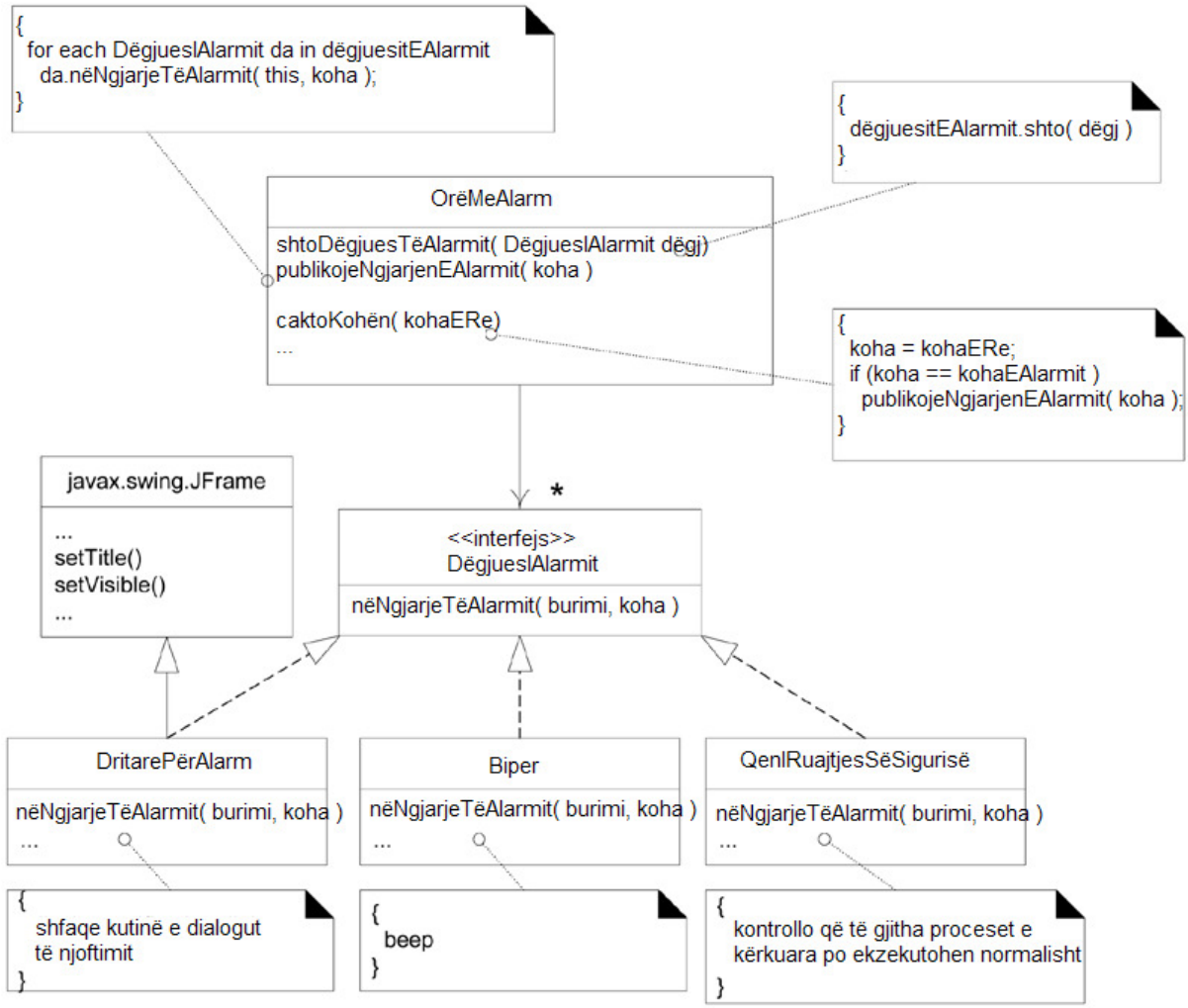
Vëzhguesi nuk është vetëm për m'i lidhë UI-të dhe objektet model

Shembulli paraprak e ilustroi lidhjen e një objekti jo-UI me një objekt UI me paternin Vëzhgues. Mirëpo, edhe përdorimet tjera janë të shpeshta.

Përdorimi më i përhapur i këtij paterni është për trajtimin e ngjarjeve të copave (kontrollave) GUI, edhe në teknologjitë Java (AWT dhe Swing) edhe në .NET-in e Microsoftit. Secila copë është publikues i ngjarjeve të ndërlidhura me GUI, dhe objektet tjera munden m'u abonu për m'u interesu për to. Për shembull, një JButton Swing e publikon një "ngjarje aksion" kur truset. Një objekt tjetër ka m'u regjistru te butoni ashtu që kur ai të truset, objektit i dërgohet një mesazh dhe mundet me ndërmarrë ndonjë aksion.

Si shembull tjetër, Figura 26.27 e ilustron një OrëMeAlarm, që është publikues i ngjarjeve alarmuese, dhe abonentë të ndryshëm. Ky shembull është ilustrativ në atë që e thekson se shumë klasa munden me implementu interfejsin *DëgjuesIAlarmit*, shumë objekte munden me qenë njëkohësisht dëgjues të regjistruar, dhe të gjithë munden me reagu në "ngjarjen e alarmit" në mënyrën e vet unike.

Figura 26.27. Vëzhguesi i aplikuar në ngjarje të alarmit, me abonentë të ndryshëm.



Një publikues mundet me pasë shumë abonentë për një ngjarje

Siç është sugjeruar në FIgurën 26.27, një instancë e publikuesit mundet me pasë prej zero deri në shumë abonentë të regjistruar. Për shembull, një instancë e një OreMeAlarm mundet m'i pasë tri DritarePërAlarm të regjistruara, katër Bipera, dhe një QenTëRuajtjesSëSigurisë. Kur ndodh një ngjarje alarm, të tetë DëgjuesitEAlarmit njoftohen përmes një mesazhi nëNgjarjeTëAlarmit.

Implementimi

Ngjarjet

Në të dy implementimet Java dhe C#.NET të Vëzhguesit, një "ngjarje" komunikohet përmes një mesazhi të rregullt, siç është nëNgjarjeTëVetisë (onPropertyEvent). Për më tepër, në të dy rastet, ngjarja më formalisht definohet si klasë, dhe mbushet me shënimet e duhura të ngjarjes. Pastaj ngjarja bartet si parametër në mesazhin e ngjarjes.

Për shembull:

```
class NgjarjeEVetisë extends Event
{
	private Object burimiINgjarjes;
	private String emriIVetisë;
	private Object vleraEVjetër;
	private Object vleraERe;
	//...
}

//...

class Shitje
{
	private void publikoNgjarjeTëVetisë(
		String emri, Objekt i_vjetri, Object i_riu )
	{
		NgjarjeEVetisë ngj =
			new NgjarjeEVetisë( this, "shitja.totali",
				i_vjetri, i_riu);

		for each DëgjuesIAlarmit da in dëgjuesitEAlarmit
			da.nëNgjarjeTëVetisë( ngj );
	}
}
```

Java

Kur është publiku JDK 1.0 në Janar 1996, ai përmbante implementim të dobët publiko-abonohu të bazuar në një klasë dhe interfejs të quajtur Observable (i vëzhgueshëm) dhe Observer (vëzhgues) përkatësisht. Kjo është kopjuar në thelb pa përmirësime nga një qasje e hershme e 1980tave ndaj publiko-abonent të implementuar në Smalltalk.

Prandaj, në 1996 e vonshëm, si pjesë e përpjekjes JDK 1.1, dizajni Observable-Observer u pat ndërru në mënyrë efektive nga versioni më i fuqishëm Java Delegation Event Model (DEM) i publiko-abonohu, edhe pse dizajni origjinal është mbajtë për kompatibilitetin-me-prapa (por në përgjithësi me iu shmangë).

Dizajnet që janë përshkru në këtë kapitull janë konsistente me DEM, por pak të thjeshtuara për m'i theksu idetë thelbësore.

Përmbledhje

Vëzhguesi e ofron një mënyrë për m'i çiftu ulët objektet në terma të komunikimit. Publikuesit dinë për abonentët vetëm përmes një interfejsi, dhe abonentët munden m'u regjistru (apo m'u çregjistru) dinamikisht te publikuesi.

Paternat e ndërlidhur

Vëzhguesi bazohet në Polimorfizëm, dhe ofron Variacione të Mbrojtura në terma të mbrojtjes së publikuesit nga njohuria për klasën specifike të objektit, dhe numrin e objekteve, me të cilët komunikon kur publikuesi e prodhon një ngjarje.

26.11. Përfundim

Mësimi kryesor për me nxjerrë nga ky shpjegim është se objektet munden m'u dizajnu dhe përgjegjësitë munden m'u nda me ndihmën e paternave. Ato ofrojnë një grup të shpjegueshëm të idiomave me të cilat mund të ndërtohen sisteme të orientuara kah objektet që janë të dizajnuara mirë.

26.12. Resurset e rekomanduara

Design Patterns nga Gamma, Helm, Johnson, dhe Vlissides është teksti rrënjësor i paternave, dhe lexim esencial për të gjithë dizajnuesit e objekteve.

Secilin vit është një konferencë "Pattern Languages of Programs" (PLOP), nga e cila publikohet një përmbledhje vjetore e paternave, në seritë *Pattern Languages of Program Design*, volumet 1, 2 e kështu me radhë. Rekomandohet tërë seria.

Pattern-Oriented Software Architecture, volumet 1 dhe 2, e kanë vazhdu diskutimin e paternave në shqetësimet arkitekturale të shkallës më të madhe. Vëllimi 1 e ka prezentu një taksonomi të paternave.

Ka qindra paterna të publikuara. Kalendari i Paternave (*The Pattern Almanac*) nga Rising e përmbledh një përqindje të respektuar të tyre.

Paterna tjerë GoF

Në vijim janë edhe disa paterna GoF që përmenden në kapitujt tjerë (vërejtje e përkthyesit).

35.4. Kalimi në rast të dështimit (failover) në serviset lokale me Përfaqësues (Proxy) (GoF)

Kalimi në servis lokal për informatën e produktit është arritë duke e futë servisin lokal para servisit të jashtëm; servisi lokal gjithmonë provohet i pari. Mirëpo, ky dizajn nuk është i përshtatshëm për të gjitha serviset; nganjëherë servisi i jashtëm duhet m'u provu i pari, dhe një version lokal i dyti. Për shembull, konsideroje postimin e shitjeve në servisin e kontabilitetit. Biznesi don m'i pasë të postume ato sa më herët që është e mundur, për vëzhgim në kohë reale të shitores dhe të aktivitetit të regjistruesit.

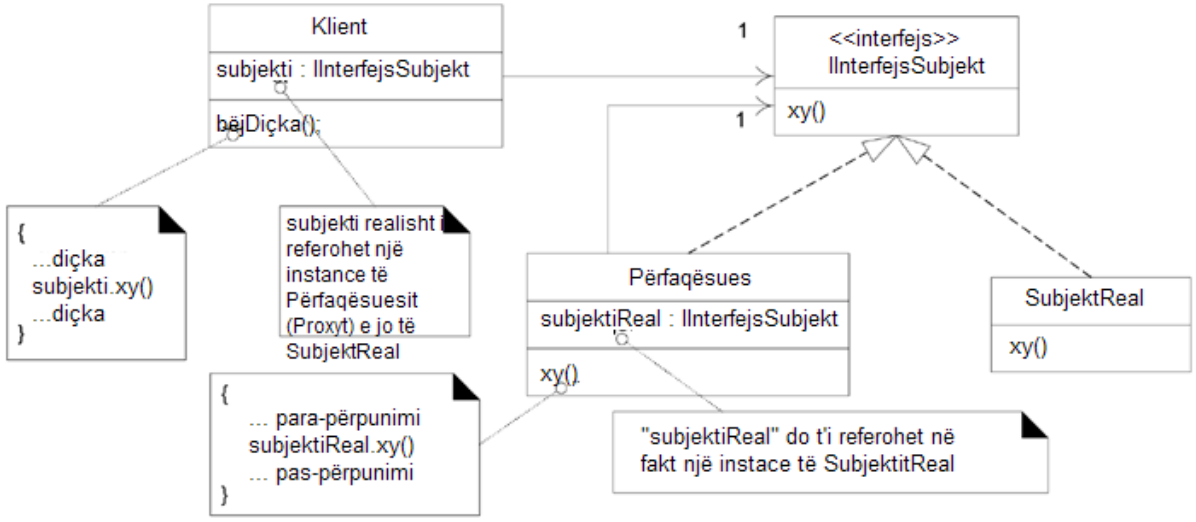
Në këtë rast, një patern tjetër GoF mundet me zgjidhë problemin: Përfaqësuesi. Përfaqësuesi është patern i thjeshtë, dhe përdoret gjerësisht në variantin e tij **Përfaqësuesi në Distancë** (Remote Proxy). Për shembull, në RMI të Javës dhe në CORBA, një objekt lokal kah ana e klientit (i quajtur "bisht" ("stub")) thirret për me iu qasë serviseve të një objekti në distancë. Bishti kah ana e klientit është përfaqësues lokal, apo një përfaqësues për objektin në distancë.

Ky shembull i përdorimit të Përfaqësuesit te NextGen nuk është varianti Përfaqësuesi në Distancë, por varianti **Përfaqësuesi i Ridrejtimit** (Redirection Proxy) (i njohur edhe si **Përfaqësuesi i Kalimdështimit** (Failover Proxy)).

Pavarësisht nga varianta, struktura e Përfaqësuesit është gjithmonë e njëjtë; variacionet janë të ndërlidhura me atë se çka bën përfaqësuesi pasi të thirret.

Një përfaqësues është thjesht një objekt që e implementon interfejsin e njëjtë si objekti subjekt, e mban një referencë në subjektin real, dhe përdoret për me kontrollu qasjen në të. Për strukturën e përgjithshme, shih Figurën 35.12.

Figura 35.12. Struktura e përgjithshme e paternit Përfaqësuesi (Proxy).



Përfaqësuesi

Konteksti/Problemi

Qasja e drejtpërdrejtë në një subjekt të vërtetë nuk është e dëshirueshme apo e mundur. Çka të bëjmë?

Zgjidhja

Shtoje një nivel të indireksionit me një përfaqësues zëvendësues që e implementon interfejsin e njëjtë si objekti subjekt, dhe është përgjegjës për kontrollimin apo përmirësimin e qasjes në të.

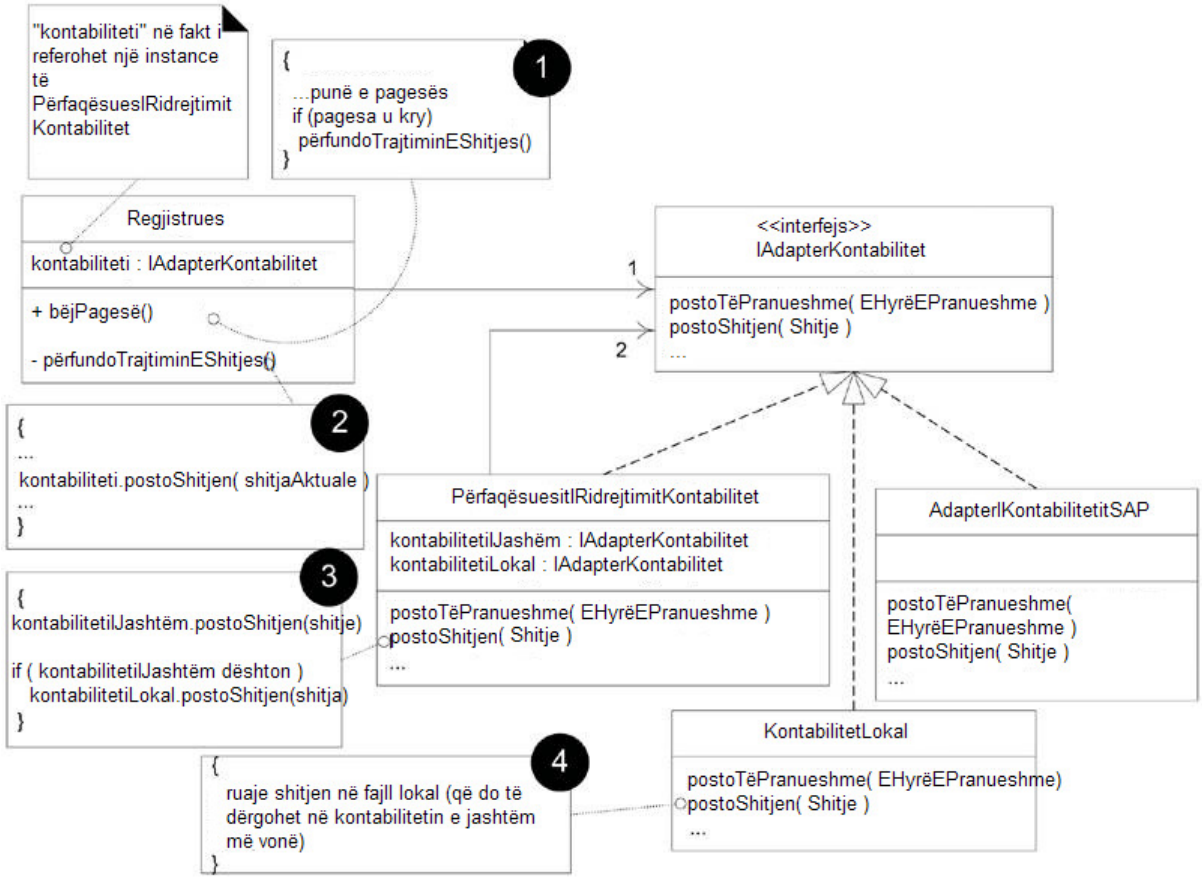
I aplikuar në studimin e rastit NextGen për qasje të jashtme të servisit të kontabilitetit, një përfaqësues i ridrejtimit përdoret si vijon:

1. Dërgoje një mesazh *postoShitjen* në përfaqësuesin e ridrejtimit, duke e trajtu atë sikur të ishte shërbimi i vërtetë i jashtëm i kontabilitetit.

2. Nëse përfaqësuesi i ridrejtimit dështon me kriju kontakt me servisin e jashtëm (përmes adapterit të vet), atëherë e ridrejton mesazhin postoShitjen në një servis lokal, që i ruan lokalisht shitjet për bartje të tyre në servisin e kontabilitetit, atëherë kur është aktiv.

Figura 35.13 e ilustron një diagram të klasave të elementeve interesante.

Figura 35.13. Përdorimi i një përfaqësuesi të ridrejtimit në NextGen.



Aplikim i UML:

- Për me iu shmangë krijimit të një diagrami të interaksionit për me tregu sjelljen dinamike, vëreni se si ky diagram statik e përdor numërimin për me tregu sekuencën e interaksionit. Zakonisht preferohet një diagram i interaksionit, por ky stil është prezentu për me ilustru një stil alternativ.
- Vëreni shënuesit publikë dhe privatë (+, -) të dukshmërisë afër metodave të Regjistruesit. Nëse mungojnë, ata janë të paspecifikuar, e jo të paracaktuar në publik apo privat. Mirëpo, sipas marrëveshjes më të zakonshme, dukshmëria e paspecifikuar interpretohet nga shumica e lexuesve (dhe veglave të gjenerimit CASE) se domethënë atribute private dhe metoda publike. Sidoqoftë, në këtë diagram, dua që në mënyrë të veçantë me tregu faktin se *bëjePagesën* është publike, dhe për kontrast, *kryejeTrajtiminEShitjes* është private. Zhurma pamore dhe mbingarkesa e informatave janë gjithmonë shqetësime në komunikim, prandaj është e dëshirueshme me përdorë interpretim me marrëveshje për m'i mbajtë diagramet të thjeshta.

Për me përmbledhë, një përfaqësues është objekt i jashtëm që e mbështjell një objekt të brendshëm, dhe të dy e implementojnë interfejsin e njëjtë. Një objekt klient (siç është Regjistruesi) nuk e di se i referohet një përfaqësuesi - ai është i dizajnuar sikur me qenë duke bashkëpunu me subjektin e vërtetë (për shembull *AdapterIKontabilitetitSAP*). Përfaqësuesi ndërhyn në thirrje në mënyrë që me përmirësu qasjen në subjektin e vërtetë, në këtë rast duke e ridrejtu operacionin në një servis lokal (KontabilitetLokal) nëse servisi i jashtëm nuk është i qasshëm.

35.7. Fabrika Abstrakte (GoF) për familjet e objekteve të ndërlidhura

Implementimet JavaPOS do të blehen nga prodhuesit. Për shembull[6]:

```
// drajverat e IBM
com.ibm.pos.jpos.CashDrawer (e implementon jpos.CashDrawer, për me nxjerrë keshin)
com.ibm.pos.jpos.CoinDispenser (e implementon jpos.CoinDispenser, për monedha)
...

// drajverat e NCR
com.ncr.pos.jpos.CashDrawer (e implementon jpos.CashDrawer, për me nxjerrë keshin)
com.ncr.pos.jpos.CoinDispenser (e implementon jpos.CoinDispenser, për monedha)
```

Tash, si me dizajnu aplikacionin NextGen POS për m'i përdorë drajverat IBM Java, nëse përdoret hardueri IBM, drajverat NCR nëse duhet, e kështu me radhë?

Vëreni se ka familje të klasave (CashDrawer + CoinDispenser + ...) që duhet m'u kriju, dhe secila familje i implementon interfejsat e njëjtë.

Për këtë situatë, ekziston një patern GoF që përdoret shpesh: Fabrika Abstrakte.

Fabrika Abstrakte

Konteksti/Problemi

Si me kriju familje të klasave të ndërlidhura që e implementojnë një interfejs të përbashkët?

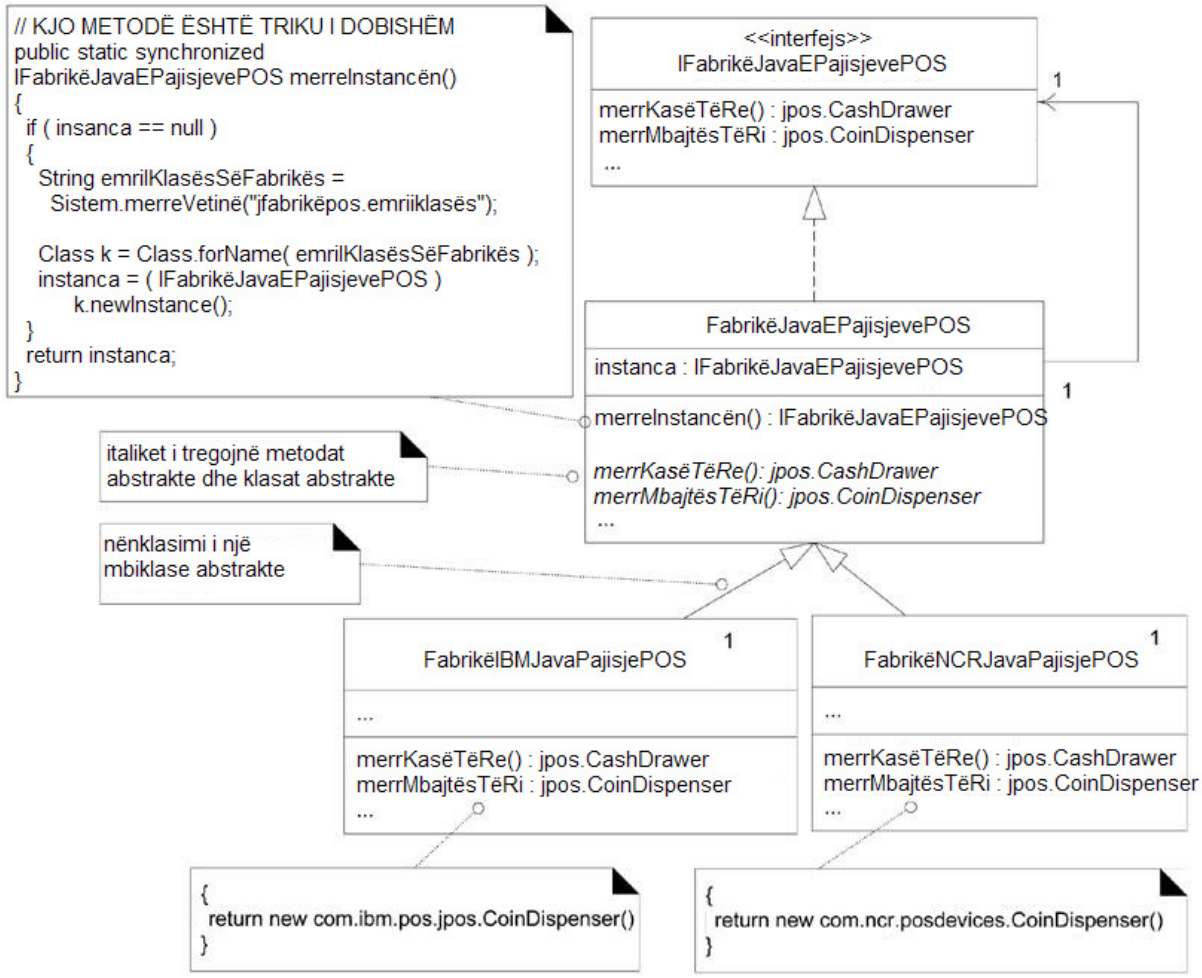
Zgjidhja

Definoje një interfejs fabrikë (fabrika abstrakte): Definoje një klasë fabrikë konkrete për secilën familje të gjërave që do të krijohen. Opsionalisht, definoje një klasë të vërtetë abstrakte që e implementon interfejsin e fabrikës dhe i ofron serviset e përbashkëta për fabrikat konkrete të cilat e zgjerojnë atë.

Figura 35.15 e ilustron idenë themelore; ajo është përmirësuar në seksionin tjetër.

[6] Këta janë emra të imagjinuar të paketave

Figura 35.15. Një fabrikë themelore abstrakte.



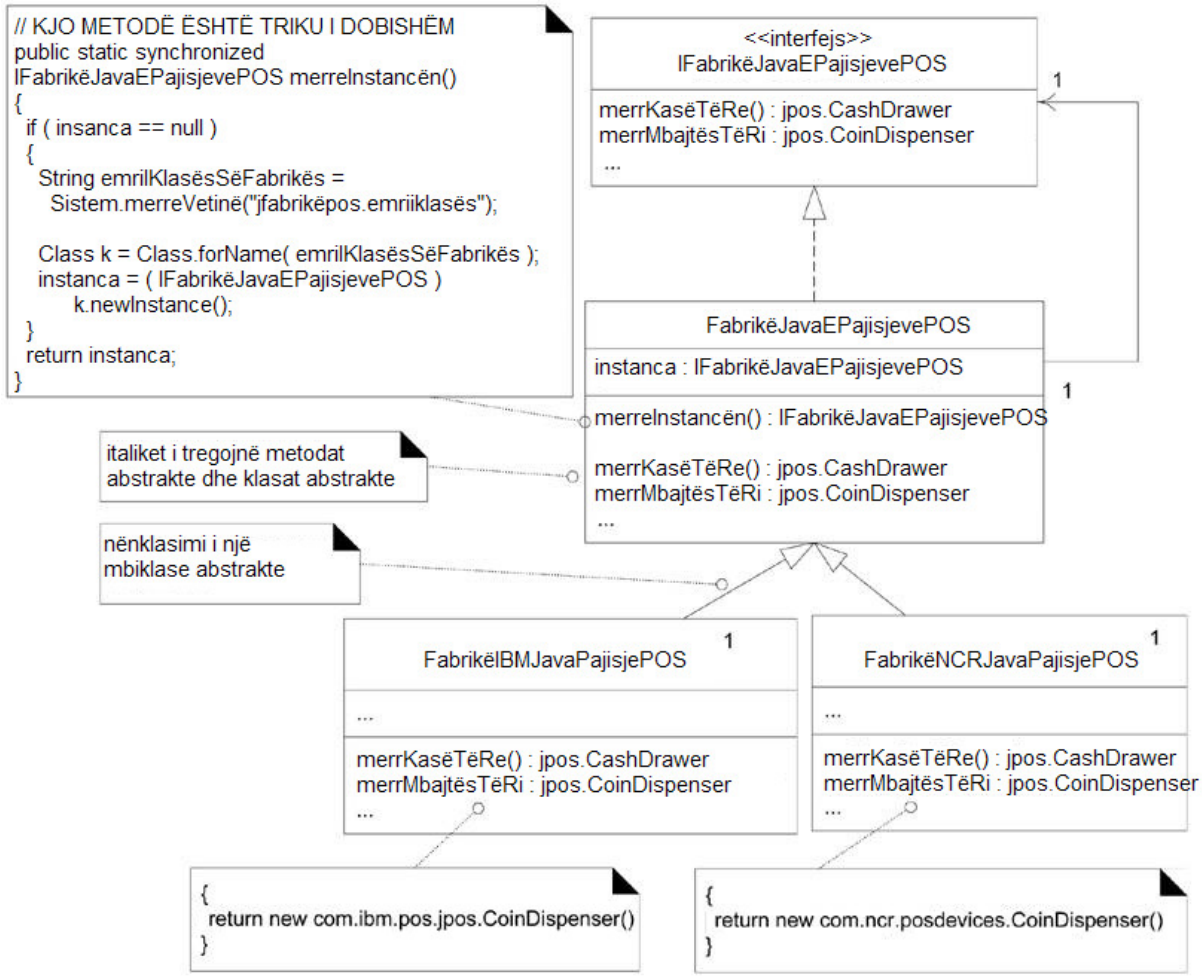
Fabrikë Abstrakte që është Klasë Abstrakte

Një variacion i shpeshtë i Fabrikës Abstrakte është me kriju një fabrikë klasë abstrakte të cilës i qasemi duke e përdorë paternin Singleton, lexon nga një veti e sistemit për me vendosë se cilën nga fabrikat e veta nënklasë me kriju, dhe pastaj e kthen instancën e duhur të nënklasës. Kjo përdoret, për shembull, në libraritë Java me klasën java.awt.Toolkit, që është një fabrikë abstrakte klasë abstrakte për krijimin e familjeve të kontrollave GUI për sisteme të ndryshme operative dhe nënsisteme GUI.

Përparësia e kësaj qasjeje është se e zgjidh këtë problem: Si e din aplikacioni se cilën fabrikë abstrakte me përdorë? FabrikëIBMJavaPajisjePOS? FabrikëNCRJavaPajisjePOS?

Përmirësimi vijues e zgjidh këtë problem. Figura 35.16 e ilustron zgjidhjen.

Figura 35.16. Një fabrikë abstrakte klasë abstrakte.



Me këtë fabrikë klasë abstrakte dhe metodën merreInstancën të paternit Singleton, objektet munden me bashkëpunu me mbiklasën abstrakte, dhe me siguru referencë në një nga instancat e nënklasave të saj. Për shembull, konsiderojeni urdhërin:

```
kasa = FabrikëPajisjetJavaPOS.merreInstancën().merrKasëTëRe();
```

Shprehja FabrikëPajisjetJavaPOS.merreInstancën() ka me kthy një instancë të FabrikëIBMPajisjetJavaPOS ose FabrikëNCRPajisjetJavaPOS, varësisht nga vetia e sistemit që është lexuar. Vëreni se duke e ndryshu vetinë e jashtme të sistemit "fabrikëjpos.emriiklasës" (që është emri i klasës si String) në një fajll të vetive, sistemi NextGen ka me përdorë familje tjetër të drajverëve JavaPOS. Variacionet e Mbrojtura ndaj një fabrike që ndryshon janë arritë me një dizajn të udhëhequr nga shënimet (data-driven) (duke e lexu një fajll të vetive) dhe dizajn të programimit reflektiv, duke e përdorë shprehjen *k.newInstance()*.

Interaksioni me fabrikën do të ndodhë në një Regjistrues. Sipas qëllimit të zbraztësisë së ulët të reprezentimit, është e arsyeshme për Regjistruesin softuerik (emri i të cilit ta kujton tërë terminalin POS) me mbajtë një referencë për pajisjet siç është Kasa. Për shembull:

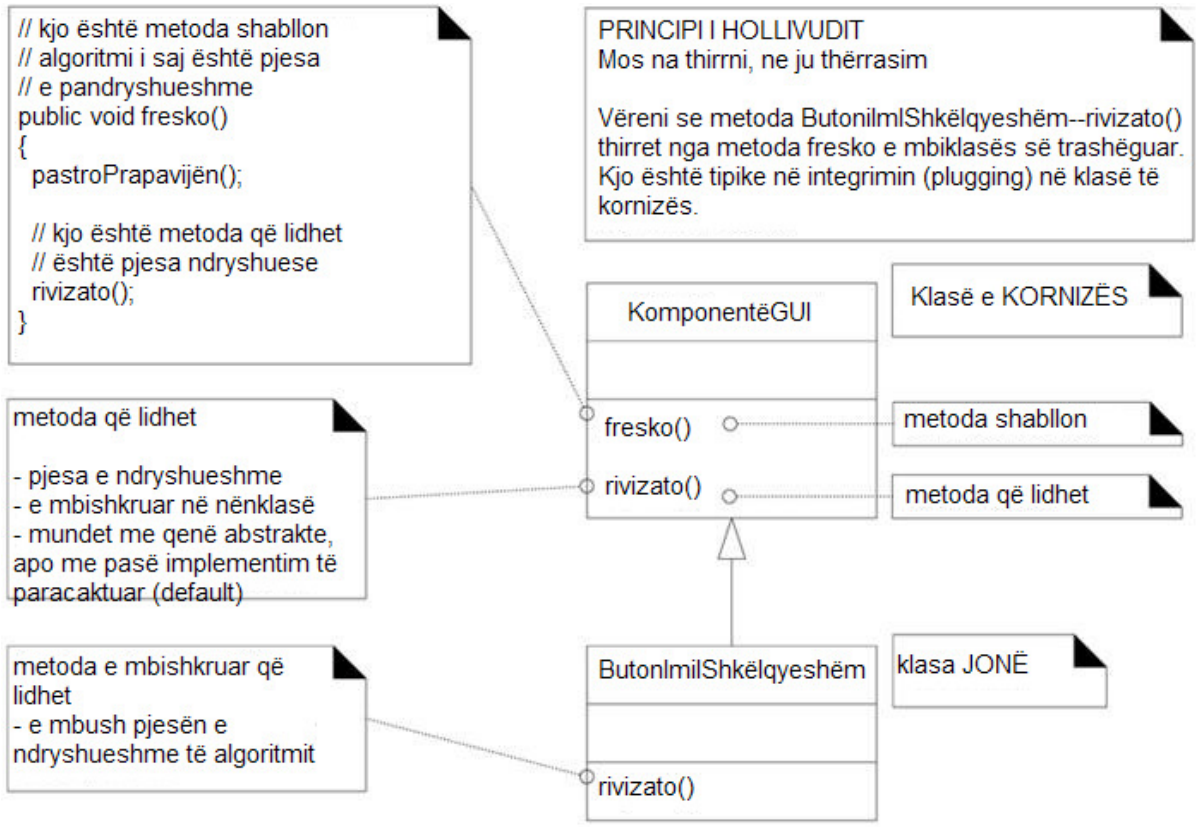
```
class Regjistrues
{
	private jpos.CashDrawer kasa;
	private jpos.CoinDispenser mbajtesi_i_monedhave;

	public Regjistrues
	{
		kasa =
			FabrikëPajisjetJavaPOS.merreInstancën.merrKasëTëRe();

		//...
	}

	//...
}
```

38.11. Dizajni i kornizës së punës me paternin "Metoda Shabllon"

Seksioni i ardhshëm i përshkruan disa nga veçoritë esenciale të Mapuesve të Bazës, që janë pjesë qendrore e PFW (Persistence Framework - Korniza Persistente). Këto veçori të dizajnit janë të bazuara në paternin GoF të dizajnit **Metoda Shabllon**[7]. Ky patern është në zemër të dizajnit të kornizës së punës[8], dhe është i njohur për shumicën e programerëve OO në praktikë nëse jo sipas emrit.

Ideja është me definu një metodë (Metoda Shabllon) në një mbiklasë që e definon skeletin e një algoritmi, me pjesët e veta të ndryshueshme dhe të pandryshueshme. Metoda Shabllon i thirr metodat tjera, disa nga të cilat munden me qenë të mbishkruara në një nënklasë. Kështu, nënklasat munden m'i mbishkru metodat e ndryshme ashtu që e shtojnë sjelljen e vet unike në pikat e ndryshimit (shih Figurën 38.6).

Figura 38.6. Paterni Metoda Shabllon në një kornizë GUI.



[7] Ky patern nuk ka lidhje me shabllonat C++. Ai e përshkruan shabllonin e një algoritmi.
[8] Më saktësisht, për kornizat kuti e bardhë. Këto janë zakonisht korniza të orientuara kah hierarkia e klasave dhe e nënklasimit që kërkojnë që përdoruesi me ditë diçka për dizajnin dhe strukturën e tyre; prandaj, kuti e bardhë.

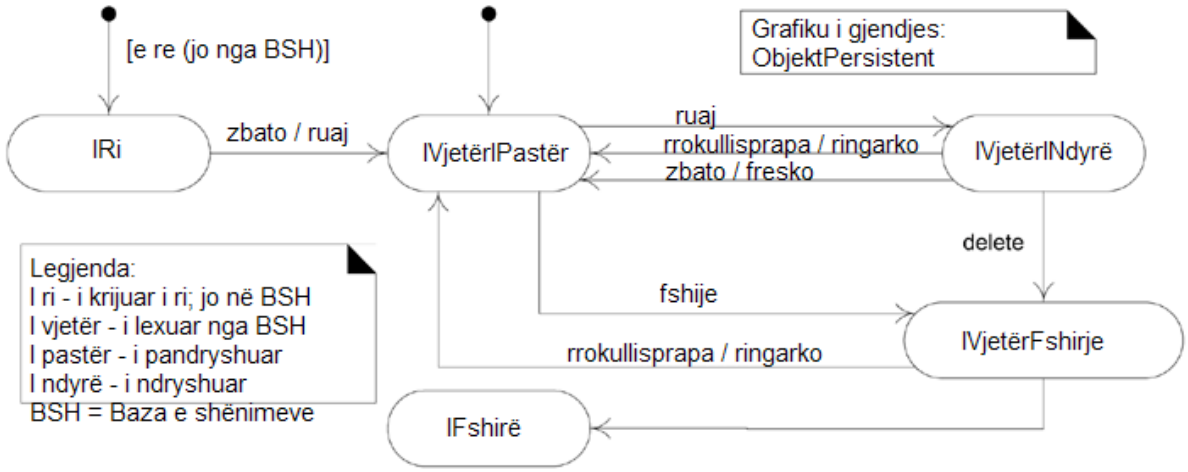
38.16. Gjendjet transaksionale dhe paterni Gjendje (State)

Çështjet e përkrahjes së transaksioneve munden m'u bë komplekse, por për m'i mbajtë gjërat të thjeshta tash për tash - për m'u përqendru në paternin GoF Gjendje - supozojmë kështu:

- Objektet persistente munden m'u futë, m'u fshi, apo m'u ndryshu
- Operimi në një objekt persistent (për shembull, ndryshimi i tij) nuk shkakton ndryshim të menjëhershëm në bazë; në vend të kësaj, duhet të kryhet një operacion eksplicit i zbatimit (commit).

Pos kësaj, përgjigjja në një operacion varet nga gjendja transaksionale e objektit. Si shembull, përgjigjjet munden m'u tregu në grafikun në Figurën 38.12.

Figura 38.12. Grafiku i gjendjes për ObjektPersistent.

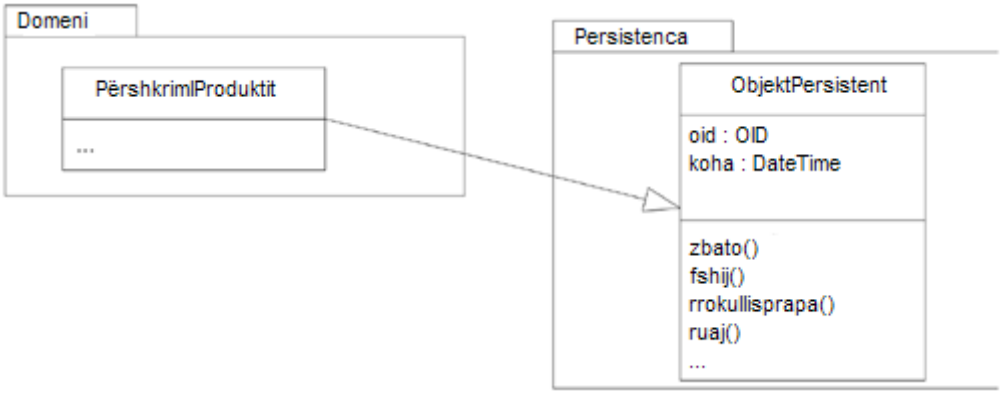


Për shembull, një objekt "i vjetër i ndyrë" është një që pranohet nga baza pastaj ndryshohet. Në një operacion zbato, ai duhet të ndryshohet në bazë - për dallim nga një që është në gjendje "i vjetër i pastër", që nuk duhet me bë asgjë (meqë nuk ka ndryshu). Brenda PFW të orientuar kah objektet (PFW = Persistence Framework - Korniza Persistente), kur bëhet një operacion fshije ose ruaje, ai nuk shkakton menjëherë fshirje apo ruajtje në bazë; në vend të kësaj, objekti persistent kalon në gjendjen përkatëse, duke e pritë një zbato apo rrokullis-prapa për me bë diçka vërtet.

Si koment UML, ky është shembull i mirë ku një graf i gjendjes është i dobishëm në komunikimin e shkurtë të informatave që përndryshe është siklet m'i shprehë.

Në këtë dizajn, supozojmë se do t'i bëjmë të gjitha klasat e objekteve persistente ta zgjerojnë një klasë *ObjektPersistent*, që i ofron serviset e zakonshme teknike për persistencë. Për shembull, shih Figurën 38.13.

Figura 38.13. Objektet Persistente.



Tani - dhe kjo është çështja që do të zgjidhet me paternin Gjendja - vëreni se metodat zbato dhe rrokullise-prapa kërkojnë struktura të ngjashme të logjikës case, bazuar në kodin e gjendjes të transaksionit. Zbato dhe rrokullis-prapa kryejnë aksione të ndryshme në rastet e veta, por ata kanë struktura të ngjashme logjike.

```
public void zbato() // commit
{
    switch ( gjendja )
    {
        case E_VJETËR_E_NDYRË:
            //...
            break;
        case E_VJETËR_E_PASTËR:
            //...
            break;
        ...
    }
}
```

```
public void rrokullisprapa() //rollback
{
    switch ( gjendja )
    {
        case E_VJETËR_E_NDYRË:
            //...
            break;
        case E_VJETËR_E_PASTËR:
            //...
            break;
        ...
    }
}
```

Një alternativë për këtë strukturë përsëritëse të logjikës case është paterni GoF Gjendja.

Gjendja

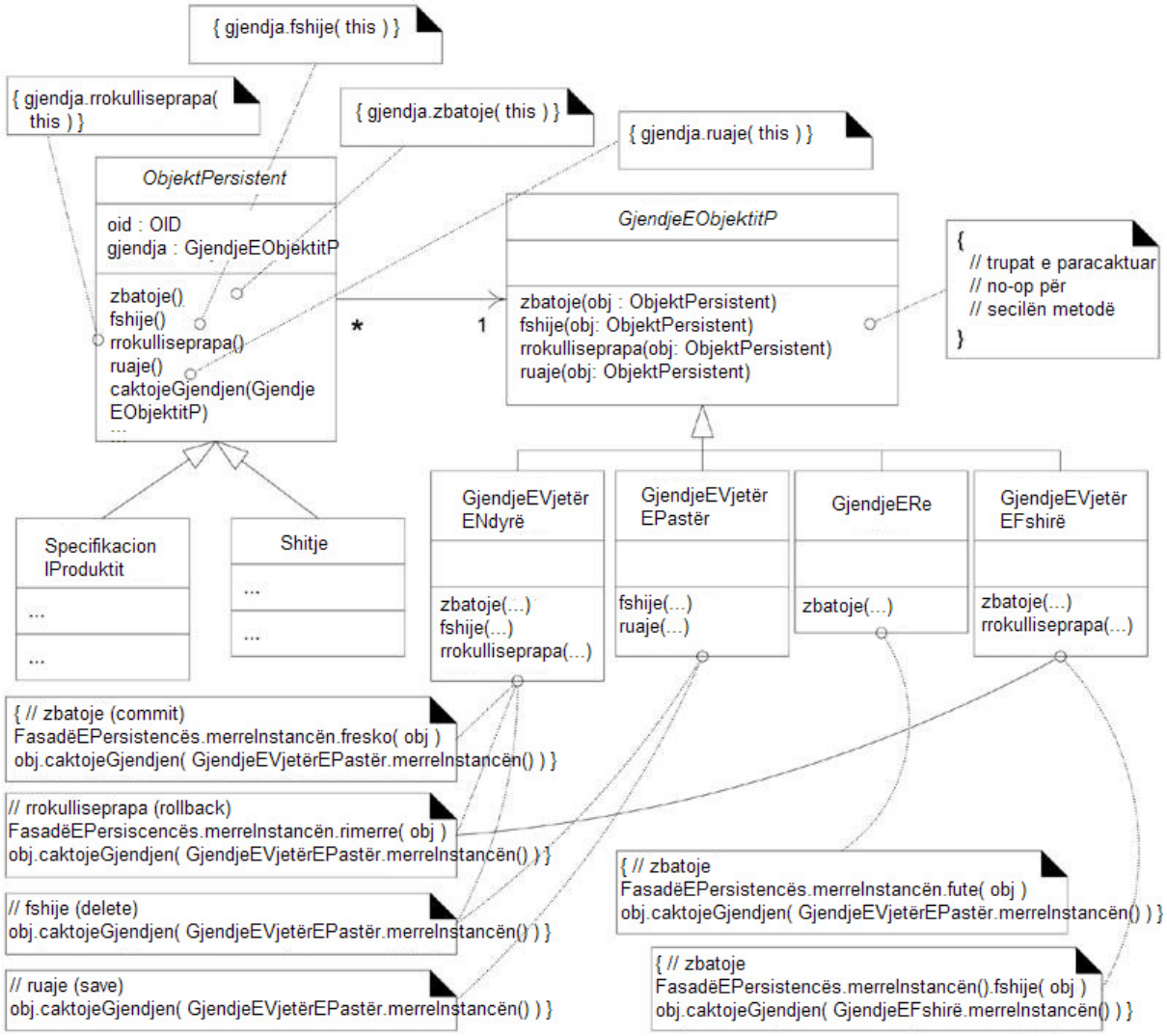
Konteksti/Problemi

Një sjellje e një objekti varet nga gjendja e tij, dhe metodat e tij përmbajnë logjikë case që i reflekton aksionet e kushtëzuara të varura nga gjendja. A ka alternativë ndaj logjikës kushtëzuese?

Zgjidhja

Krijo klasë të gjendjes për secilën gjendje, duke e implementu një interfejs të përbashkët. Delegoji operacionet e varura nga gjendja nga objekti i kontekstit në objektin i gjendjes së vet aktuale. Sigurohu që objekti i kontekstit gjithmonë është i lidhur me një objekt të gjendjes që e reflekton gjendjen e tij.

Figura 38.14. Aplikimi i paternit Gjendja[9].



Metodat e varura nga gjendja në *ObjektPersistent* ia delegojnë ekzekutimin e vet një objekti të asociuar të gjendjes. Nëse objekti kontekstual është duke iu referu objektit *GjendjeEVjetërENdryrë*, atëherë 1) metoda *zbatoje* do ta shkaktojë një freskim në bazë dhe 2) objekti kontekstual do të ricaktohet me iu referu objektit *GjendjeEVjetërEPastër*. Në anën tjetër, nëse objekti kontekstual është duke iu referu objektit *GjendjeEVjetërEPastër*, metoda e trashëguar e zbatimit mos-bëj-asgjë ekzekutohet dhe nuk bën asgjë (siç edhe pritet, meqë objekti është i pastër).

Vëreni në Figurën 38.14 se klasat e gjendjes dhe sjellja e tyre i korrespondojnë grafit të gjendjes në Figurën 38.12. Paterni Gjendja është një mekanizëm për me implementu një model të ndryshimit të gjendjes në softuer[10]. Ai shkakton që një objekt me ndryshu në gjendje të ndryshme si përgjigjje ndaj ngjarjeve.

Si koment i performansës, këto objekte të gjendjes janë - për ironi - pa gjendje (nuk kanë atribute). Prandaj, nuk ka nevojë me pasë shumë instanca të një klase - secila është singleton. Mijëra objekte persistente munden me iu qasë instancës së njëjtë *GjendjeEVjetërENdyrë*, për shembull.

[9] Klasa EFshirë është heqë për shkak të kufizimit të hapësirës në diagram

[10] Ka mekanizma tjerë, duke e përfshi logjikën e kushtëzuar të shkruar në kod, interpretues të makinës së gjendjes, dhe gjenerues të kodit të udhëhequr nga tabela të gjendjes.

38.17. Dizajnimi i një transaksioni me paternin Komanda

Seksioni i fundit pati pamje të thjeshtuar të transaksioneve. Ky seksion e zgjeron diskutimin, por nuk i mbulon të gjitha çështjet e dizajnit të transaksionit. Joformalisht, një transaksion është një njësi e detyrave - një grup i detyrave - detyrat e të cilit duhet të kryhen të gjitha me sukses, apo të mos kryhet asnjëra. Domethënë, kryerja është atomike.

Në terma të servisit të persistencës, detyrat e transaksionit përfshijnë objektet e futjes, ndryshimit, dhe të fshirjes. Një transaksion, për shembull, mundet m'i përmbajtë dy futje, një ndryshim, dhe tri fshirje. Për me prezentu këtë, është shtu një klasë Transaksion. Siç është potencu në librin "Draft patterns on object-relational persistence services", radha e detyrave brenda transaksionit mundet me ndiku në suksesin (dhe performansën) e tij.

Për shembull:

1. Supozojmë se baza ka një kufizim të integritetit referencial ashtu që kur ndryshohet një rresht në TabelënA që e përmban një çelës të jashtëm të një rreshti në TabelënB, baza kërkon që rreshti në TabelënB tashmë ekziston.

2. Supozojmë se një transaksion e përmban një detyrë INSERT (futje) për me shtu rreshtin në TabelënB, dhe një detyrë UPDATE (ndryshim) për me ndryshu rreshtin në TabelënA. Nëse UPDATE ekzekutohet para INSERT, ngritet një gabim i integritetit referencial.

Renditja e punëve të bazës mundet me ndihmu. Disa çështje të renditjes janë specifike sipas skemës, por një strategji e përgjithshme është m'i kry së pari futjet, pastaj ndryshimet, pastaj fshirjet.

Vëreni se renditja në të cilën shtohen detyrat në transaksion nga një aplikacion mund të mos e reflektojnë radhitjen e tyre më të mirë të ekzekutimit. Detyrat duhet të renditen pikërisht para ekzekutimit të tyre.

Kjo çon edhe në një patern tjetër GoF: Komanda

Komanda

Konteksti/Problemi

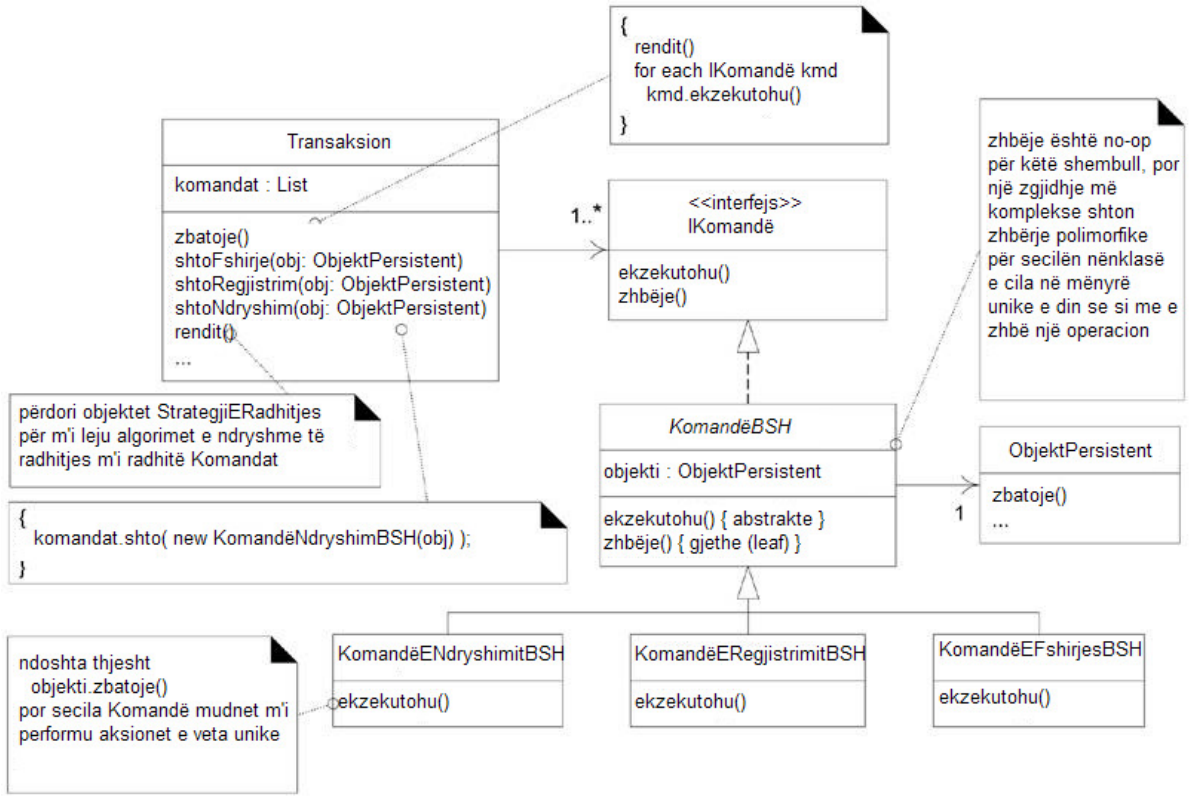
Si m'i trajtu kërkesat apo detyrat që kërkojnë funksione siç është renditja (prioritizimi), futja në radhë, vonimi, futja në ditar, apo zhbërja?

Zgjidhja

Bëje secilën detyrë një klasë që e implementon një interfejs të përbashkët.

Ky është patern i thjeshtë me shumë aplikime të dobishme; aksionet bëhen objekte, dhe kështu munden m'u renditë, m'u futë në ditar, m'u futë në radhë, e kështu me radhë. Për shembull, në PFW, Figura 38.15 i tregon klasat Komanda (apo të detyrave) për operacionet e bazës së shënimeve.

Figura 38.15. Komandat për operacionet e bazës së shënimeve.



Ka shumë më shumë për me kompletu një zgjidhje për transaksione, por ideja kryesore e këtij seksioni është me përfaqësu secilën detyrë apo aksion në transaksion si një objekt me një metodë polimorfike ekzekuto; kjo e hap një botë të fleksibilitetit duke e trajtu kërkesën si objekt.

Shembulli kuintesencial i Komandës është për aksionet GUI, siç janë preje dhe ngjite (cut and paste). Për shembull metoda ekzekuto e *KomandëPreje* e bën një prerje, dhe metoda e saj *zhbëje* (undo) e kthen prerjen mbrapa. Objekti *KomandëPreje* do t'i ruajë edhe shënimet e nevojshme për me kry zhbërjen. Të gjitha komandat GUI munden m'u mbajtë në një stek historik, ashtu që munden m'u nxjerrë me radhë, dhe secila m'u zhbë.

Një tjetër përdorim i shpeshtë i Komandës është për trajtimin e kërkesave në anën e serverit. Kur një objekt i serverit e pranon një mesazh (në distancë), ai e krijon një objekt Komandë për atë kërkesë, dhe ia dorëzon një objekti *PërpunuesIKomandave*, i cili mundet m'i futë në radhë, në ditar, m'i prioritizu, dhe m'i ekzekutu komandat.

Shtojcë: artifakte dhe shembuj

Në vijim është përshkrimi i dy studimeve të rasteve që janë përdorë si shembuj gjatë tërë librit, si dhe disa artifakte më të rëndësishme nëpër kapitujt tjerë. Kjo shtojcë është edhe në pjesën e parë të përkthyer: “GRASP: Dizajnimi i Objekteve me Përgjegjësi”.

Studimet e rasteve (case studies)

(kapitulli 3)

Rasti Një: Sistemi NextGEN POS

Studimi i rastit të parë është sistemi pikë-e-shitjes NextGen (point-of-sale, POS). Në këtë domen të problemit që në dukje të parë është i drejtpërdrejtë, do të shohim se ka probleme interesante të kërkesave dhe të dizajnit për m’i zgjidhë. Pastaj, është problem real – grupet vërtet zhvillojnë sisteme POS me teknologji të objekteve.

Një sistem POS është aplikacion i kompjuterizuar që përdoret (pjesërisht) për m’i regjistru shitjet dhe m’i trajtu pagesat; zakonisht përdoret në shitore me pakicë. Ai i përfshin komponentat e harduerit siç janë një kompjuter dhe një skener i bar-kodeve, dhe softuerin për me ekzekutu sistemin. Ai ndërvepron me interfejsa me aplikacione të ndryshme të serviseve, siç është llogaritësi i jashtëm i taksave dhe kontrolli i inventarit. Këto sisteme duhet të jenë relativisht tolerante ndaj dështimit; domethënë, edhe nëse serviset në distancë janë përkohësisht të paqasshme (siç është sistemi i inventarit), ato prapë duhet me qenë të afta m'i rujtë shitjet dhe m'i trajtu të paktën pagesat kesh (ashtu që biznesi të mos dëmtohet).

Një sistem POS në mënyrë graduale duhet t'i përkrahë shumë terminalë dhe interfejsa të ndryshëm. Këto përfshijnë një terminal me shfletues Web, një kompjuter i rregullt personal me diçka si ndërfaqja grafike e përdoruesit Java Swing, hyrje me ekran prekës, pajisje pa tela, e kështu me radhë.

Gjithashtu, jemi duke e kriju një sistem komercial POS që do t'ua shesim klientëve të ndryshëm me nevoja të pangjashme në terma të përpunimit të rregullave të biznesit. Secili klient do të dëshirojë grup unik të logjikës për m'u ekzekutu në disa pika të parashikueshme në skenarët e përdorimit të sistemit, siç është kur niset një shitje e re apo kur shtohet një artikull i ri në shitje. Prandaj, do të na duhet një mekanizëm për me ofru këtë fleksibilitet dhe mundësi të specifikimit.

Duke përdorë strategji iterative të zhvillimit, do të vazhdojmë nëpër kërkesa, analizë të orientuar kah objektet, dizajn dhe implementim.

Rasti Dy: Sistemi i Lojës Monopoly

Për me tregu se praktikat e njëjta të OOA/D munden m'u apliku në probleme shumë të ndryshme, e kam zgjedhë një version të softuerit të lojës Monopoly si një studim tjetër të rastit. Edhe pse domeni dhe kërkesat nuk janë fare si një sistem i biznesit siç është NextGen POS, do të shohim se modelimi i domenit, dizajni i objekteve me paterna, dhe aplikimi i UML janë ende relevante dhe të dobishme. Si me një POS, versionet softuerike të Monopoly't janë vërtet të zhvilluara dhe të shitura, edhe me UI klient-i-pasur (rich-client) edhe me UI Web.

Nuk do t'i përsëris rregullat për Monopoly; si duket pothuajse secili person, në secilin vent, e ka luajtë këtë lojë si fëmijë apo adoleshent. Nëse keni pyetje, rregullat janë online në shumë web-faqe.

Versioni softuerik i lojës do të ekzekutohet si simulim. Një person do ta fillojë lojën dhe do ta tregojë numrin e lojtarëve të simuluar, dhe pastaj do të shohë derisa loja ekzekutohet deri në përfundim duke e paraqitë një gjurmë të aktivitetit gjatë radhëve të lojtarëve të simular.

Rasti i përdorimit RP1: Përpunoje Shitjen

(kapitulli 6)

Niveli: qëllim i përdoruesit

Aktori primar, kryesor: Kasieri

Pjesëmarrësit (aksionarët) dhe Interesat:

- Kasieri: Dëshiron futje të saktë dhe të shpejtë, dhe pa gabime të pagesës, meqë mungesat e keshit tërhiqen nga rroga e tij.
- Shitësi: I dëshiron komisionet (shtesat) e shitjes të freskuara
- Klienti: Dëshiron blerje dhe shërbim të shpejtë me angazhim minimal. Dëshiron shfaqje të lehtë e të dukshme të artikujve dhe çmimeve të futura. Dëshiron dëshmi të blerjes për t'i përkrahë kthimet.
- Kompania: Dëshiron t'i regjistrojë saktësisht transaksionet dhe t'i kënaqë interesat e klientit. Dëshiron të sigurojë që regjistrohen të pranueshmet e pagesës për Shërbimin e Autorizimit të Pagesës. Dëshiron ca tolerancë ndaj gabimit për me leju regjistrim të shitjeve edhe nëse komponentat e serverit (p.sh. validimi në distancë i kredisë) janë të padisponueshme. Dëshiron freskim automatik dhe të shpejtë të kontabilitetit dhe të inventarit.
- Menaxheri: Dëshiron të jetë i aftë që t'i kryejë shpejt operacionet e mbishkrimit, dhe t'i rregullojë lehtë problemet e Kasierit.
- Agjencitë Qeveritare të Tatimeve: Duan të mbledhin taksë nga secila shitje. Mund të jenë shumë agjenci, si, kombëtare, shtetorëre dhe qarkore.
- Shërbimi i Autorizimit të Pagesës: Dëshiron të pranojë kërkesa me autorizim digjital në formatin dhe protokolin korrekt. Dëshiron që të llogarisë saktë të pagueshmet e tyre për shitoren.

Parakushtet: Kasieri është identifikua dhe autentiku.

Garantimi i Suksesit (ose Paskushtet): Shitja është ruajtur. Tatimi është llogaritur në mënyrë korrekte. Kontabiliteti dhe Inventari janë freskuar. Komisionet janë regjistruar. Fatura gjenerohet. Aprovimet e autorizimit të pagesës janë regjistruar.

Skenari Kryesor i Suksesit (apo Rrjedha Themelore):

1. Klienti arrin te kasa me të mirat dhe/ose shërbimet që don t'i blejë
2. Kasieri e nis një shitje të re.
3. Kasieri e fut identifikuesin e artikullit.
4. Sistemi e ruan rreshtin e shitjes dhe e paraqet përshkrimin e artikullit, çmimin dhe totalin aktual. Çmimi llogaritet nga një grup i rregullave të çmimeve.

   Kasieri i përsërit hapat 3-4 derisa ta indikojë "mjaft".

5. Sistemi e paraqet totalin me taksat e llogaritura.
6. Kasieri ia tregon totalin klientit dhe e kërkon pagesën.

7. Klienti paguan dhe sistemi e trajton pagesën.
8. Sistemi e fut në ditar shitjen e kryer dhe i dërgon informatat e shitjes dhe të pagesës në sistemin e jashtëm të Kontabilitetit (për kontabilitet dhe komisione) dhe sistemin e Inventarit (për me freskua Inventarin).
9. Sistemi e paraqet faturën.
10. Klienti shkon me faturën dhe të mirat (nëse ka).

Zgjerimet (ose Rrjedhat Alternative):

*a. Në çfarëdo kohe, Menaxheri kërkon veprim të mbishkrimit:

1. Sistemi hyn në mënyrën Menaxheri i autorizuar
2. Menaxheri apo Kasieri e kryejnë një veprim të Mënyrës së menaxherit, p.sh., ndërrimin e balansit të keshit, rifillo një shitje të suspenduar në një regjistër tjetër, anuloje një shitje, etj.
3. Sistemi kthehet në mënyrën Kasieri i autorizuar.

*b. Në çfarëdo kohe, Sistemi bjen (dështon):

Për me përkrahë rikëndelljen dhe për me rregullu kontabilitetin, sigurohu që të gjitha gjendjet dhe ngjarjet e ndijshme të transaksionit mund të rikëndellen nga cilido hap i skenarit.

1. Kasieri e ristarton Sistemin, kyçet, dhe kërkon rikthim të gjendjes së mëparshme.
2. Sistemi e rikonstrukton gjendjen e mëparshme.

2a. Sistemi detekton anomali që e parandalojnë rikëndelljen:

1. Sistemi e sinjalizon për gabimin Kasierin, e regjistron gabimin, dhe hyn në gjendje të pastër
2. Kasieri fillon shitje të re

1a. Klienti apo Menaxheri indikojnë ta rikthejnë një shitje të anuluar.

1. Klienti e kryen operacionin e rikthimit, dhe e fut IDnë për me lexu shitjen.
2. Sistemi e shfaq gjendjen e shitjes së rikthyer, me nëntotalin.
   2a. Shitja nuk gjendet.
      1. Sistemi e sinjalizon Kasierin për gabimin.
      2. Kasieri me gjasë fillon një shitje të re dhe i ri-fut të gjithë artikujt.
3. Kasieri vazhdon me shitjen (me siguri duke futur më shumë artikuj apo duke e trajtu pagesën).

2-4a. Klienti i tregon Kasierit se ai ka status të çlirimit nga tatimi (p.sh. seniorët, autoktonët)

1. Kasieri e verifikon, dhe pastaj e fut kodin e lirimit nga taksa
2. Sistemi e regjistron statusin (që do ta përdorë gjatë llogaritjes së tatimeve)

3a. ID jovalide e artikullit (nuk gjendet në sistem)

1. Sistemi e sinjalizon gabimin dhe e refuzon futjen
2. Kasieri reagon ndaj gabimit:

    2a. Ka ID që lexohet nga njeriu (p.sh. UPC numerike)

    1. Kasieri e fut manualisht IDnë e artikullit
    2. Sistemi e shfaq përshkrimin dhe çmimin

        2a. ID jovalide e artikullit: Sistemi e sinjalizon gabimin. Kasieri provon metodë alternative.

    2b. Nuk ka ID të artikullit, por ka çmim në etiketë:

    1. Kasieri i thotë Menaxherit me kry një operasion të mbishkrimit.
    2. Menaxheri e kryen mbishkrimin.
    3. Kasieri e indikon futjen manuale të çmimit, e fut çmimin, dhe kërkon taksim standard për këtë sasi (meqë nuk ka informatë për produktin, makina e taksës nuk mund ta nxjerrë ndryshe se si me e taksu)

    2c. Kasieri e kryen Ndihmën Gjeje Produktin për me marrë IDnë e vërtetë dhe çmimin

    2d. Përndryshe, Kasieri e lut një nëpunës për IDnë ose çmimin e vërtetë, dhe e bën ose IDnë manuale ose çmimin manual (shih më lart).

3b. Ka shumë nga kategoria e njëjtë e artikullit dhe përcjellja e identitetit unik të artikullit nuk është e rëndësishme (p.sh., 5 paqeta të burgerëve vegjetarianë):

1. Kasieri mund ta fusë identifikuesin e kategorisë së artikujve dhe sasinë.

3c. Artikulli kërkon futje manuale të kategorisë dhe të çmimit (si lulet apo kartolinat me çmime në to):

1. Kasieri e fut kodin e kategorisë speciale manuale, plus çmimin.

3-6a: Klienti i thotë Kasierit me e heqë një artikull nga blerja:
Kjo është legale vetëm nëse vlera e artikullit është më e vogël se sa limiti i heqjes për Kasierë, përndryshe nevojitet mbishkrim i Menaxherit.

1. Kasieri e fut identifikuesin e artikullit për me heqë nga shitja.

2. Sistemi e heq artikullin dhe e shfaq totalin e freskuar aktual.

2a. Çmimi i artikullit e tejkalon kufirin e heqjes për Kasierët:

1. Sistemi sinjalizon gabim, dhe sugjeron mbishkrim të Menaxherit

2. Kasieri e kërkon mbishkrimin nga Menaxheri, e merr, dhe e përsërit operacionin.

3-6b: Klienti i thotë Kasierit me anulu shitjen:

1. kasieri e anulon shitjen në Sistem.

3-6c: Kasieri e suspendon shitjen:

1. Sistemi e regjistron shitjen ashtu që të jetë e disponueshme për marrje në cilindo regjistër POS.

2. Sistemi e paraqet një "faturë të suspendimit" që i përfshin artikujt e rreshtit dhe një ID të shitjes që përdoret për me çelë dhe rivendosë shitjen.

4a. Çmimi i ofruar nga sistemi është i padëshiruar (p.sh. Klienti është anku për diçka dhe i ofrohet çmim më i ulët):

1. Kasieri e kërkon aprovimin nga Menaxheri.

2. Menaxheri e kryen operacionin e mbishkrimit.

3. Kasieri e fut çmimin manual të mbishkrimit.

4. Sistemi e paraqet çmimin e ri.

5a. Sistemi detekton dështim në komunikimin me shërbimin e jashtëm të llogaritjes së taksimit:

1. Sistemi e ristarton shërbimin në nyjen e POS dhe vazhdon.

1a. Sistemi detekton se shërbimi nuk ristartohet

1. Sistemi sinjalizon për gabimin.

2. Kasieri mund ta llogarisë manualisht dhe ta fusë taksën, ose ta anulojë shitjen.

5b. Klienti thotë se kanë të drejtë në zbritje (p.sh. nëpunës, klient i preferuar):

1. Kasieri e sinjalizon kërkesën për zbritje.

2. Kasieri e fut identifikimin e Klientit

3. Sistemi e paraqet totalin e zbritjes, duke u bazu në rregullat e zbritjes.

5c. Klienti thotë se ka kredi në llogarinë e vet, për me apliku në shitje:

1. Kasieri e sinjalizon kërkesën për kredi

2. Kasieri e fut identifikimin e Klientit

3. Sistemi e aplikon kredinë deri në çmimin = 0, dhe e redukton kredinë e mbetur.

6a. Klienti thotë se ka dashtë me pagu në kesh por nuk ka mjaft kesh:

1. Kasieri kërkon mënyrë alternative të pagesës

1a. Klienti i tregon Kasierit me anulu shitjen. Kasieri e anulon shitjen.

7a. Pagesa me kesh:

1. Kasieri e fut sasinë e parave të ofruara.

2. Sistemi e paraqet balansin, dhe e liron fjokën e kasës.

3. Kasieri e depoziton keshin e futur dhe ia kthen balansin në kesh klientit.

4. Sistemi e regjistron pagesën kesh.

7b. Pagesa me kredi:

1. Klienti i fut informatat e llogarisë së kreditit.

2. Sistemi e shfaq pagesën e vet për verifikim.

3. Kasieri konfirmon.

3a. Kasieri e anulon hapin e pagesës:

1. Sistemi kthehet në mënyrën "futja e artikujve"

4. Sistemi e dërgon kërkesën për autorizim të pagesës te një Sistem i Shërbimit të Autorizimit të Pagesës, dhe kërkon aprovim të pagesës.

4a. Sistemi detekton gabim për me bashkëpunu me sistemin e jashtëm:

1. Sistemi e sinjalizon gabimin te Kasieri.

2. Kasieri i thotë Klientit për pagesë alternative.

5. Sistemi e pranon aprovimin e pagesës, e sinjalizon aprovimin te Kasieri, dhe e liron fjokën e kasës (për me futë faturën e nënshkruar të pagesës me kredi).

5a. Sistemi pranon refuzim të pagesës:

1. Sistemi e sinjalizon refuzimin te klienti

2. Kasieri i thotë Klientit për pagesë alternative.

5b. Kalon koha për pritjen e përgjigjjes

1. Sistemi e sinjalizon kalimin e kohës

2. Kasieri mund të provojë prapë, ose t'i thotë klientit për pagesë alternative

6. Sistemi e regjistron pagesën e kredisë, që e përfshin aprovimin e pagesës.

7. Sistemi e prezenton mekanizmin e futjes së nënshkrimit të kredisë.

8. Kasieri e lut klientin për nënshkrim të pagesës me kredi. Klienti e jep nënshkrimin.

9. Nëse nënshkrimi në letër, Kasieri e vendos faturën në fjokë dhe e mbyll.

7c. Pagesa me çek...

7d. Pagesa me debi...

7e. Kasieri e anulon hapin e pagesës.

1. Sistemi kthehet në mënyrën "futja e artikujve".

7f. Klienti i paraqet kuponat:

1. Para trajtimit të pagesës, Kasieri e regjistron secilin kupon dhe Sistemi e redukton çmimin si duhet. Sistemi i regjistron kuponat e përdorur për arsye të kontabilitetit.

1a. Kuponi i futur nuk është për asnjë artikull të blerë:

1. Sistemi sinjalizon gabim te Kasieri.

9a. Ka rabat (zbritje) të produkteve:

1. Sistemi i paraqet format e zbritjes dhe faturat e zbritjes për secilin artikull me rabat.

9b. Klienti kërkon faturë të dhuratës (pa çmime):

1. Kasieri kërkon faturë të dhuratës dhe sistemi e paraqet atë.

9c. Printeri pa letër.

1. Nëse sistemi mund ta detektojë gabimin, do ta sinjalizojë problemin.

2. Kasieri e vendos letrën.

3. Kasieri kërkon edhe një faturë.

Kërkesat speciale:

- Ndërfaqe e përdoruesit për Ekran Prekës në një monitor të madh të rrafshtë të panelit. Teksti duhet të jetë i dukshëm nga 1 metër.
- Përgjigjje e autorizimit të kredisë brenda 30 sekondave në 90% të kohës.
- Disi, duam rikëndellje të shpejtë kur qasja në shërbimet në distancë si ç'është sistemi i inventarit, dështon.
- Ndërkombëtarizimi i gjuhës në tekstin e shfaqur.
- Rregulla të futshme të biznesit të jenë të futshme në hapat 3 dhe 7.

Lista e Teknologjisë dhe e Variacioneve të Shënimeve:

*a. Mbishkrimi i menaxherit futet duke e kalu një kartelë të mbishkrimit përmes një lexuesi të kartelave, ose duke e futë një kod të autorizimit përmes tastierës.

3a. Identifikuesi i artikujve i futur nga barkod laser skeneri (nëse bar kodi është prezent) ose tastiera.

3b. Identifikues i sistemit mund të jetë cilado skemë e kodimit UPC, EAN; JAN, apo SKU.

7a. Informatat e llogarisë së kredisë të futura nga lexuesi i kartelës apo tastiera.

7b. Nënshkrimi i pagesës së kredisë i kapur në faturë letër. Por brenda dy viteve, parashikojmë se shumë klientë do të duan kapje të nënshkrimit digjital.

Frekuenca e Ngjarjes: mund të jetë pothuajse e vazhdueshme.

Çështjet e hapura:

- Cilat janë variacionet e ligjeve të taksave?

- Hulumtoje çështjen e rikëndelljes së shërbimit në distancë.

- Çfarë specifikimi nevojitet për biznese të ndryshme?

- A duhet një kasier me e marrë fjokën e kasës kur shkyçen?

- A mundet ta përdorë direkt klienti lexuesin e kartelës apo duhet me bë kasieri?

Ky rast i përdorimit është ilustrues e jo i detalizuar (edhe pse është i bazuar në kërkesat e një sistemi real POS – i zhvilluar me dizajn OO në Jaba). Megjithatë, këtu ka mjaft detale dhe kompleksitet për me ofru një sens realistik se një rast i përdorimit “plotësisht i veshur” (fully dressed) mundet me regjistru shumë detale të kërkesave. Ky shembull do të shërbejë edhe si model për shumë probleme të rasteve të përdorimit.

Rasti i përdorimit RP1: Luaje lojën Monopoly

(kapitulli 6)

Shtrirja: Aplikacioni Monopoly

Niveli: qëllimi i përdoruesit

Aktori kryesor: Vështruesi

Aksionarët dhe Interesat:

- Vështruesi: Dëshiron që ta vështrojë lehtë daljen e simulimit të lojës

Skenari kryesor i suksesit:

1. Vështruesi e kërkon një inicializim të ri të lojës, i jep numrat e lojtarëve
2. Vështruesi fillon play
3. Sistemi e shfaq gjurmën e lojës për lëvizjen e lojtarit tjetër (shih rregullat e domenit, dhe "gjurmën e lojës" në fjalor për detale të gjurmës).

Përsërite hapin 3 derisa të ketë fitues apo deri sa Vështruesi e anulon.

Zgjerimet:

*a. Në çdo kohë, Sistemi dështon:

(Për me përkrahë rikëndelljen, Sistemi shkruan në ditar pas çdo lëvizjeje të kryer)

1. Vështruesi e ristarton sistemin
2. Sistemi e detekton dështimin e kaluar, e rikonstrukton gjendjen, dhe kërkon për të vazhduar
3. Vështruesi zgjedh me vazhdu (nga rendi i lojtarit të fundit që e ka kry).

Kërkesat speciale:

- Ofro të dy mënyrat: edhe gjurmë grafike edhe tekstuale.

Specifikim plotësues (suplementar) – Shembulli NextGen

(kapitulli 7)

Historia e Rishikimeve

Versioni	Data	Përshkrimi	Autori
Drafti i zanafillës	10 janar 2031	Drafti i parë. M'u gdhendë kryesisht gjatë shtjellimit (elaborimit).	Craig Larman

Hyrje

Ky dokument është depo e të gjitha kërkesave të NextGen që nuk janë përfshirë në rastet e përdorimit.

Funksionaliteti

(Funksionaliteti i përbashkët nëpër shumë raste të përdorimit)

Shkruarja në Ditar dhe Trajtimi i Gabimeve

Shkruaji në ditar (log) të gjitha gabimet në një depo persistente (të qëndrueshme)

Rregullat e kyçshme (të integrueshme, pluggable)

Në pika të ndryshme të skenarit të disa rasteve të përdorimit (që do të definohen) përkrahe aftësinë me e specifiku (customize) funksionalitetin e sistemit me një grup të rregullave arbitrare që ekzekutohen në atë pikë ose ngjarje.

Siguria

Të gjitha përdorimet kërkojnë autentikim të përdoruesit

Përdorshmëria

Faktorët njerëzorë

Klienti do të jetë në gjendje ta shohë një shfaqje të POS në një monitor të madh. Prandaj:

- Teksti duhet të jetë i dukshëm nga 1 metër
- Shmangiu ngjyrave që shoqërohen me format më të shpeshta të verbërisë së ngjyrave.

Përpunimi i shpejtë, i lehtë, dhe pa gabime janë supreme në përpunimin e shitjeve, meqë blerësi don të shkojë shpejt, ose e perceptojnë përjetimin e blerjes (dhe shitësin) si më pak pozitive.

Kasieri shpesh e shikon klientin apo artikujt, jo ekranin e kompjuterit. Prandaj, sinjalet dhe alarmet duhet të njoftohen me zë e jo vetëm përmes grafikës.

Besueshmëria

Rikëndellshmëria

Nëse ka dështim në përdorimin e shërbimeve të jashtme (autorizuesi i pagesës, sistemi i kontabilitetit, ...) mundohu me i zgjidhë me një zgjidhje lokale (p.sh. vendos dhe dërgo) në mënyrë që ende të mund të kryhet shitja. Këtu nevojitet shumë më shumë analizë...

Performansa

Si u përmend te faktorët njerëzorë, blerësit duan ta bëjnë përpunimin e shitjeve shumë shpejt. Një qafëshishe është autorizimi i jashtëm i pagesës. Qëllimi ynë: autorizimi për më pak se 1 minutë, 90% të kohës.

Përkrahshmëria

Adaptabiliteti

Klientët e ndryshëm të POSit NextGen kanë nevoja unike të rregullave të biznesit dhe të përpunimit gjatë përpunimit të një shitjeje. Prandaj, në disa pika të definuara të skenarit (për shembull, kur niset një shitje, kur shtohet një artikull i ri) rregulla e futshme do të aktivizohet.

Konfigurueshmëria

Klientët e ndryshëm dëshirojnë konfigurime të ndryshme të rrjetës për sistemet e veta POS, siç kanë klientët e trashë versus të hollë, shtresat fizike dy-shtresore versus N-shtresore, etj. Pastaj, ata e duan mundësinë e ndryshimit të këtyre konfiguracioneve, për m'i reflektu nevojat e tyre të ndryshueshme të biznesit dhe performansës. Prandaj, sistemi do të jetë disi i konfigurueshëm për

m'i reflektu këto nevoja. Në këtë fushë nevojitet shumë më shumë analizë për m'i zbulu zonat dhe shkallën e fleksibilitetit, dhe përpjekjet për me e arritë atë.

Kufizimet e implementimit

Udhëheqja e NextGen insiston në zgjidhje me teknologji të Java's, duke parashiku se kjo do ta përmirësojë bartjen dhe përkrahshmërinë afat-gjatë, si shtojcë të lehtësisë së zhvillimit.

Komponentat e blera

- Llogaritësi i taksave. Duhet të përkrahë llogaritës së integrueshëm për shtete të ndryshme.

Komponentat e Lira (free) me Kod të Hapur

Në përgjithësi, e rekomandojmë makzimizimin e komponentave të lira me kod të hapur të teknologjisë Java në këtë projekt.

Edhe pse është heret të dizajnohen dhe zgjedhen definitivisht komponentat, sugjerojmë si në vijim si kandidatë potencialë:

- JLog korniza për kyçje
- ...

Interfejsat

Hardueri dhe Interfejsat e Rëndësishëm

- Monitori me ekran prekës (ky perceptohet nga sistemet operative si monitor i rregullt, dhe gjestet e prekjes si ngjarje të mausit)
- Skener laserik i barkodit (këta normalisht lidhen me tastierë speciale, dhe hyrja e skenuar perceptohet në softuer si taste të trusura)
- Printeri për fatura
- Lexuesi i kartelave debi/kredi
- Lexuesi i nënshkrimit (por jo në lëshimin 1)

Interfejsat Softuerikë

Për shumicën e sistemeve të jashtme bashkëpunuese (llogaritësi i taksave, kontabiliteti, inventari,...) na duhet të jemi të aftë me integru sisteme të ndryshme, dhe prandaj interfejsa të ndryshëm.

Rregullat e Domenit (Biznesit) që janë Specifike për Aplikacionin

(Shih dokumentin e veçantë Rregullat e Biznesit për rregullat e përgjithshme)

ID	Rregulla	Ndërrueshmëria	Burimi
RREGULLA1	Rregullat e zbritjes së blerësit. Shembuj: Nëpunësi: 20% më lirë. Klient i preferuar: 10% më lirë. Senior: 15% më lirë.	E lartë. Secili shitës me pakicë përdor rregulla të ndryshme.	Politika e shitësit.
RREGULLA2	Rregullat e shitjes (në nivel të transaksionit). Zbatohet në totalin para-tatimit. Shembuj: 10% më lirë nëse totali mbi 100$ 5% më lirë çdo të Hënë. 10% më lirë secila shitje nga 10am në 3pm sot. Tofu 50% më lirë nga 9am-10am sot.	E lartë. Secili shitës përdor rregulla të ndryshme, dhe ato mund të ndryshojnë në baza ditore ose brenda orës.	Politika e shitësit.
RREGULLA3	Rregullat e zbritjes për produkt (artikull i rreshtit). Shembuj: 10% më lirë traktorët këtë javë. Bleji dy vegjeburgerë, merre 1 falas.	E lartë. Secili shitës përdor rregulla të ndryshme, dhe ato mund të ndryshojnë në baza ditore ose brenda orës.	Politika e shitësit.

Çështjet ligjore

I rekomandojmë disa komponenta me kod të hapur nëse mund të zgjidhen kufizimet e tyre të licensimit për me leju rishitjen e produkteve që përfshijnë softuer me kod të hapur.

Të gjitha rregullat duhet, sipas ligjit, të zbatohen gjatë shitjeve. Vëreni se këto mund të ndryshojnë shpesh.

Informata në Domenet e Interesit

Caktimi i çmimeve

Pos rregullave të caktimit të çmimit të përshkruara në seksionin e rregullave të domenit, vëreni se produktet kanë çmim origjinal, dhe opsionalisht një çmim të përhershëm të zbritjes. Çmimi i produktit (para zbritjeve të mëtutjeshme) është çmimi permanent i zbritjes, nëse ekziston. Organizatat e mirëmbajnë çmimin origjinal edhe nëse ka çmim të përhershëm të zbritjes, për arsye të kontabilitetit dhe të taksimit.

Trajtimi i pagesave debi dhe kredi

Kur një pagesë elektronike debi apo kredi aprovohet nga një shërbim i autorizimit të pagesës, ata janë përgjegjës për pagesën e shitësit, jo blerësi. Si pasojë, për secilën pagesë, shitësi duhet t'i regjistrojë paratë që i ka borxh në llogaritë e tyre të pranueshme, nga shërbimi i autorizimit. Zakonisht, çdo mbrëmje, shërbimi i autorizimit do të kryejë transfer elektronik të fondeve në llogarinë e shitësit për borxhin total ditor, minus një tarifë (e vogël) për transaksion që e kërkon si pagesë shërbimi.

Taksat e shitjes

Llogaritjet e taksave të shitjes mund të jenë shumë komplekse, dhe ndërrohen rregullisht në përgjigjje të legjislacionit në të gjitha nivelet e qeverisjes. Prandaj, delegimi i llogarive të taksimit një softueri llogaritës palë e tretë (prej të cilëve disa janë në dispozicion) është i këshillueshëm. Tatimin mund të jetë borxh për trupat e qytetit, regjionit, shtetit, dhe atij nacional. Disa artikuj mund të jenë të liruar nga taksa pa kualifikim, ose të liruar varësisht nga blerësi apo pranuesi cak (për shembull, një fermer apo një fëmijë).

Identifikuesit e artikujve: UPCtë, EANtë, SKUtë, Bar Kodet, dhe Lexuesit e Barkodeve.

NextGen POS nevojitet të përkrahë skema të ndryshme të identifikimit të artikuve. UPCtë (Universal Product Codes), EANs (European Article Numbering) dhe SKUtë (Stock Keeping Units) janë tre sisteme më të shpeshta të identifikimit për produktet që janë shitur. Japanese Article Numbers (JANët) janë një lloj i versionit EAN.

SKUtë janë identifikues plotësisht arbitrarë që definohen nga shitësi.

Mirëpo, UPCtë dhe EANët kanë një komponentë të standardeve dhe rregullatore. Shih www.adams1.com/pub/russadam/upccode.html për një përmbledhje të mirë. Po ashtu, shih www.uc-council.org dhe www.ean-int.org.

Dokument i vizionit - Shembulli NextGen: Vizioni (i Pjesshëm)

(kapitulli 7)

Historia e rishikimit

Versioni	Data	Përshkrimi	Autor
drafti i zanafillës	10 janar 2031	Drafti i parë. Do të gdhendet gjatë shtjellimit (elaboration)	Craig Larman

Hyrje

E vizionojmë një aplikacion pikë-e-shitjes (PESH) të gjeneratës së ardhshme, NextGen POS, me fleksibilitetin me përkrahë rregulla të ndryshme të biznesit të klientit, shumë mekanizma terminalë dhe të ndërfaqeve të përdoruesve, dhe integrimin me shumë sisteme përkrahëse si palë të treta.

> *Analiza në këtë shembull është ilustrative, por imagjinare.*

Pozicionimi

Oportuniteti i Biznesit

Produktet ekzistuese POS nuk janë të adaptueshme për biznesin e klientit, në termat e rregullave të ndryshueshme të biznesit dhe dizajnet e ndryshueshme të rrjetave (për shembull klient i trashë apo jo, arkitektura 2, 3, apo 4-shtresore). Pastaj, ato nuk shkallëzohen mirë derisa numri i terminaleve dhe biznesi rriten. Dhe, asnjë nuk mund të punojnë ose në mënyrën on-line ose off-line, duke u adaptu në mënyrë dinamike varësisht nga dështimet. Asnjëri nuk integrohet lehtë me shumë sisteme palë-të-treta. Asnjë nuk lejon teknologji të reja të terminalëve siç janë PDAtë mobile. Ka pakënaqësi të tregut me ktë gjendje jofleksibile të çështjeve, dhe kërkesë për një POS që e rregullon këtë.

Deklarata e Problemit

Sistemet tradicionale janë jofleksibile, jotolerante ndaj gabimeve, dhe vështirë të integrueshme me sistemet e palëve të treta. Kjo çon në probleme në përpunimin në kohë të shitjeve, duke kriju procese të avansuara që nuk përputhen me softuerin, dhe shënimet e sakta dhe në kohë të kontabilitetit dhe të inventarit për me përkrahë matjen dhe planifikimin, ndërmjet shqetësimeve tjera. Kjo ndikon në kasierët, menaxherët e shitjes, administratorët e sistemeve, dhe menaxhmentin e korporatës.

Deklarata e pozicionit të produktit

Përmbledhje e shkurtër se për kë është sistemi, veçoritë e tij të shquara, dhe çka e dallon nga konkurenca.

Alternativat dhe Gara...

Përshkrimet e aksionarëve

Kuptoni se kush janë lojtarët, dhe problemet e tyre.

Demografitë e tregut...

Përmbledhje e Aksionarëve (Jo-Përdorues)...

Përmbledhje e Përdoruesve...

Qëllimet kryesore të nivelit të lartë dhe problemet e aksionarëve

Një punëtori (workshop) një-ditore e kërkesave me ekspertët e temës dhe aksionarët tjerë, dhe analizat në disa tregje të shitjeve çuan në identifikimin e qëllimeve dhe problemeve vijuese kryesore:

Konsolido hyrjen nga Lista e Aktorëve dhe e Qëllimeve, dhe seksioni Interesat e Aksionarëve i rasteve të përdorimit.

Qëllimi i nivelit të lartë	**Prioriteti**	**Problemet dhe shqetësimet**	**Zgjidhjet aktuale**
Përpunim i shpejtë, i fuqishëm dhe i integruar i shitjeve	I lartë	Shpejtësia e zvogëluar derisa rritet ngarkesa Humbja e mundësisë së përpunimit të shitjeve nëse komponentat dështojnë. Mungesa e informatave aktuale dhe të sakta nga kontabiliteti dhe sistemet tjera si pasojë e jo-integrimit me sistemet ekzistuese të kontabilitetit, inventarit, dhe HR. Çon në vështirësi në matje dhe planifikim. Pamundësia me i specifiku rregullat e biznesit për kërkesat unike të biznesit. Vështirësia me shtu terminal të ri apo tipe të ndërfaqe së përdoruesit (për shembull PDA mobile).	Produktet ekzistuese POS ofrojnë përpunim themelor të shitjes, por nuk i adresohen këtyre problemeve.
. . .	. . .	. . .	. . .

Qëllimet e nivelit të përdoruesit

Përdoruesit (dhe sistemet e jashtme) kërkojnë një sistem që i plotëson këto qëllime:

> *Kjo mund të jetë Lista e Qëllimeve-Aktorëve e krijuar gjatë modelimit të rasteve të përdorimit, apo përmbledhje më e shkurtër.*

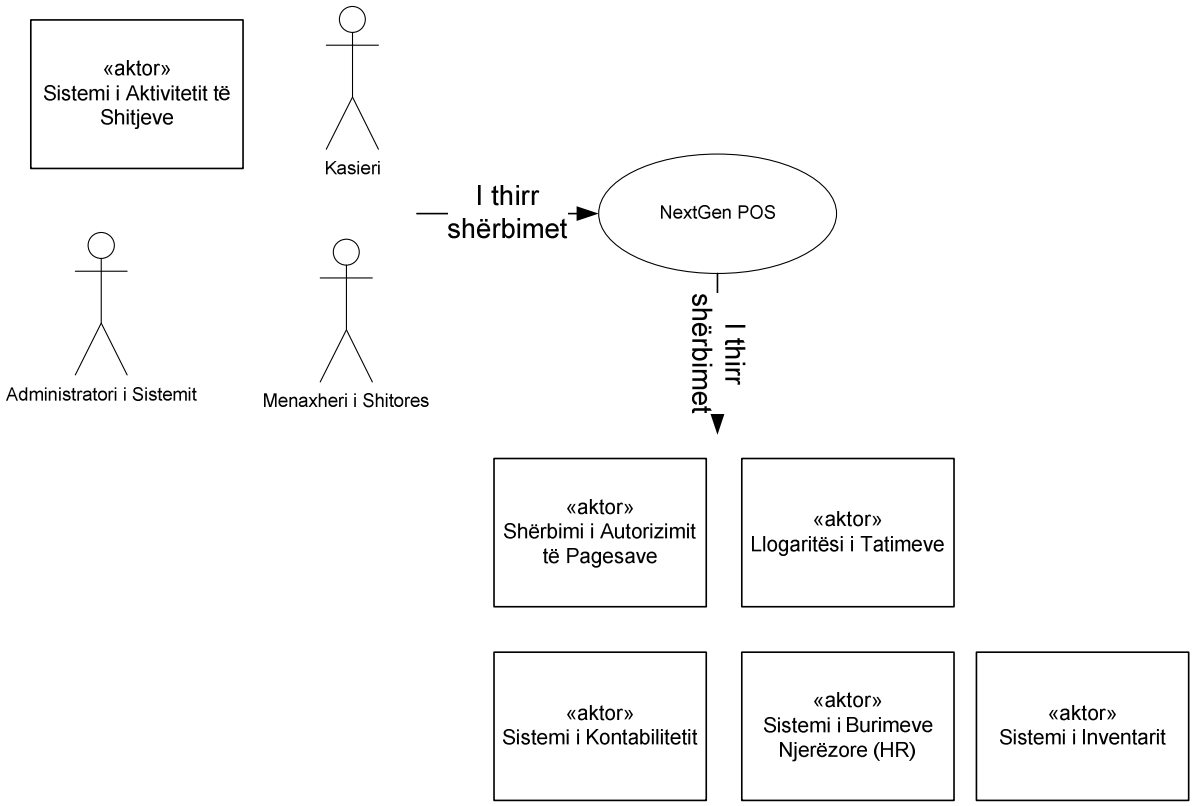
- Kasieri: përpuno shitjet, trajto kthimet, paret mrena, paret jashtë
- Administratori i sistemit: menaxho përdoruesit, sigurinë, dhe tabelat e sistemit
- Menaxheri: lshoje, nale
- Sistemi i aktivitetit të shitjeve: Analizo shënimet e shitjes

Ambienti i përdoruesit…

Përmbledhje e Produktit

Perspektiva e Produktit

POSi NextGen zakonisht do të rrijë në shitore; nëse përdoren terminalët mobilë, ata do të jenë në afërsi të afërt me rrjetën e shitorës, ose brenda ose afër jashtë. Do të ofrojë shërbime për përdoruesit, dhe bashkëpunojë me sistemet tjera, siç është e cekur në Figurën 1.



Përmbledhur nga Diagrami i Rasteve të Përdorimit. Diagramet e kontekstit vijnë në formate të ndryshme me detale ndryshuese, por të gjitha i tregojnë aktorët kryesorë që kanë të bëjnë me sistemin.

Përmbledhja e Benefiteve

Veçoria Përkrahëse	**Përfitimi i Aksionarit**
Funksionaliteti, sistemi do t'i ofrojë të gjitha shërbimet e zakonshme që i kërkon një organizatë e shitjes, duke i përfshirë regjistrimin e shitjes, autorizimin e pagesës, trajtimin e kthimit, etj.	Shërbime të automatizuara, të shpejta të Pikës-së-Shitjes.
Detektimi automatik i dështimeve, kalimi në përpunimin offline për shërbimet e padisponueshme	Përpunimi i vazhduar kur komponentat e jashtme dështojnë.
Rregullat e integrueshme të biznesit në skenarë të ndryshëm gjatë përpunimit të shitjeve.	Konfigurim fleksibilil i logjikës së biznesit.
Transaksione në kohë reale me sisteme të palëve të treta, duke i përdorë protokolet standarde industriale.	Shitje me kohë, të sakta, informata të kontabilitetit dhe të inventarit, për me përkrahë matjen dhe planifikimin
…	…

Ngjashëm me listën Qëllimet-Aktorët, kjo tabelë i lidh qëllimet, përfitimet dhe zgjidhjet, por në nivel më të lartë që nuk ka të bëjë vetëm me rastet e përdorimit. Ajo e përmbledh vlerën dhe cilësitë dalluese të produktit.

Supozimet dhe varësitë…

Kostoja dhe caktimi i çmimeve…

Licensimi dhe Instalimi…

Përmbledhja e Veçorive të Sistemit

Siç diskutohet më poshtë, veçoritë e sistemit janë një format i shkurtër për me përmbledhë funksionalitetin.

- Regjistrimi i shitjeve
- Autorizimi i pagesës (kredi, debi, çek)
- Administrimi sistemor për përdorues, siguri, kod dhe tabela të konstanave, etj.
- Procesimi automatik i shitjeve ofline kur komponentat e jashtme dështojnë
- Transaksionet në kohë reale, duke u bazu në standarde industriale, me sisteme të palëve të treta, duke e përfshi inventarin, kontabilitetin, resurset humane, llogaritësit e taksave, dhe shërbimet e autorizimit të pagesës
- Definicioni dhe ekzekutimi i rregullave të specifikueshme të futshme në pika të caktuara të përbashkëta në skenarë të përpunimit
- …

Kërkesat dhe kufizimet tjera

Duke i përfshirë kufizimet e dizajnit, përdorshmërinë, besueshmërinë, performansën, përkrahshmërinë, kufizimet e dizajnit, dokumentacionin, paketimin, etj: Shih Specifikacionin plotësues dhe rastet e përdorimit.

Fjalor i projektit

(kapitulli 7)

Shembulli NextGen: Fjalori (i pjesshëm)

Historia e rishikimit

Versioni	Data	Përshkrimi	Autori
Drafti i zanafillës	10 janar 2031	Drafti i parë. Do të gdhendet gjatë përpunimit.	Craig Larman

Definicionet

Termi	Definicioni dhe informata	Formati	Rregullat e validimit	Sinonimet/Aliaset
Artikull	Produkt ose shërbim për shitje			
Autorizim i pagesës	Validim nga një shërbim i jashtëm i autorizimit që do ta bëjnë ose do ta garantojnë pagesën ndaj shitësit			
Kërkesa e autorizimit të pagesës	Kompozit i elementeve që dërgohen elektronikisht në një shërbim të autorizimit, zakonisht një varg i karakterëve. Elementet përfshijnë: ID e shitores, numri i llogarisë së klientit, sasia dhe vula e kohës.			
UPC	Kodi numerik që e identifikon një produkt. Zakonisht simbolizohet me një barkod të vendosur në produkte. Shih www.uc-council.org për detale të formatit dhe validim.	Kod 12-shifror me disa nënpjesë	Shifra 12 është shifër e kontrollimit	Universal Product Code
…	…			

Rregullat e domenit - Shembulli NextGen

(kapitulli 7)

Rregullat e domenit

Historia e rishikimit

Versioni	Data	Përshkrimi	Autori
Drafti i zanafillës	10 Janar 2031	Drafti i parë. Do të gdhendet së pari gjatë elaborimit	Craig Larman

Lista e rregullave

(Shih po ashtu Rregullat specifike për Aplikacionin në Specifikimin Plotësues)

ID	Rregulla	Ndërrueshmëria	Burimi
RREGULLA1	Kërkohet nënshkrimi për pagesat kreditore	"Nënshkrimi" i blerësit do të vazhdojë të kërkohet, por brenda 2 viteve shumica e klientëve tonë duan regjistrim të nënshkrimit në një pajisje digjitale të regjistrimit, dhe brenda 5 viteve presim të ketë kërkesë për përkrahje të "nënshkrimit" me kod të ri digjital unik që përkrahet nga ligji.	Politika e thuajse të gjitha kompanive të autorizimit të kredive.
RREGULLA2	Rregullat e taksimit. Shitjet kërkojnë taksa të shtuara. Shih statuset e qeverisë për detalet aktuale.	E lartë. Ligjet e taksave ndërrojnë çdo vit, në të gjitha nivelet e qeverisë.	ligji
RREGULLA3	Ndryshimet e pagesës së kredisë mund të paguhen vetëm si kredi në llogarinë e kredisë të blerësit, jo si kesh.	E ulët	Politika e kompanisë së autorizimit të kredisë

Kërkesat e iteracionit 1

(kapitulli 8)

NextGen POS

Kërkesat për iteracionin e parë të aplikacionit NextGen POS janë si në vijim:

- Implemento një skenar themelor, kryesor të rastit të përdorimit Përpuno Shitjen: futja e artikujve dhe pranimi i pagesës kesh.
- Implemento një rast të përdorimit Fillo sipas nevojës për t'i përkrahë nevojat e inicializimit për iterimin.
- Asgjë ekstra ose komplekse nuk trajtohet, vetëm një skenar i thjeshtë i rrugës së lumtur, dhe dizajni dhe implementimi për me përkrahë atë.
- Nuk ka bashkëpunim me shërbimet e jashtme, siç është llogaritësi i tatimeve apo baza e produktit.
- Nuk aplikohen rregulla komplekse të çmimimit.
- Dizajni dhe implementimi i NP (UI) përkrahëse, bazës e kështu me radhë, po ashtu do të bëhej, por nuk është e mbuluar në ndonjë detal.

Monopoly

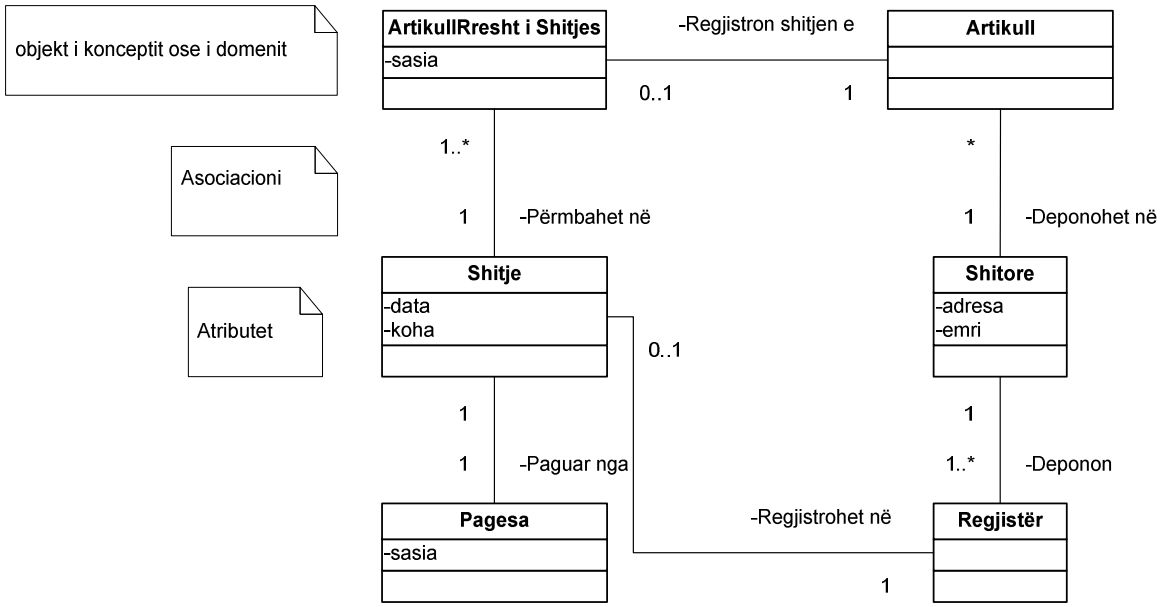
Kërkesat për iteracionin e parë të aplikacionit Monopoly janë si vijon:

- Implemento një skenar themelor, bazik të rastit të përdorimit Luaje Lojën Monopoly: lojtarët duke lëvizur nëpër katrorët e fushës.
- Implemento një rast të përdorimit Fillo sipas nevojës për me i përkrahë nevojat e inicializimit të iteracionit.
- Dy deri në tetë lojtarë mund të luajnë.
- Loja luhet si seri e rundeve. Gjatë një rundi, secili lojtar e merr radhën një herë. Në secilën radhë, një lojtar e shtyn figurën e tij në drejtim të akrepave përreth fushës në numër të katrorëve baraz me shumën e rrotulluar në dy zare gjashtë-faqëshe.
- Luaje lojën për vetëm 20 runde.
- Pasi të jenë rrokullisur zaret, emri i lojtarit dhe rrokullisja shfaqen. Kur lojtari lëviz dhe vendoset në një kuti, emri i lojtarit dhe emri i kutisë në të cilën është ndalur shfaqen.
- Në iteracionin-1 nuk ka para, fitues apo humbës, as prona për të blerë apo qira me pagu, dhe nuk ka kuti speciale të asnjë lloji.
- Secila kuti e ka një emër. Secili lojtar e fillon lojën me figurën e tij të vendosur në kutinë "Nisu." Emrat e kutive do të jenë Nisu, Kutia 1, Kutia 2, ... Kutia 39
- Nise lojën si simulim që nuk kërkon hyrje të përdoruesit, pos numrit të lojtarëve.

Iterimet pasuese do të ndërtohen në këto baza.

Model i Domenit

(kapitulli 9)



Klasat konceptuale

(kapitulli 9)

Regjistrues	Artikull	Shitore	Shitje
ArtikullRresht i Shitjes	Kasier	Klient	Libër i llogarive
Pagesë Kesh	Katalog i Produkteve	Përshkrim i Produktit	

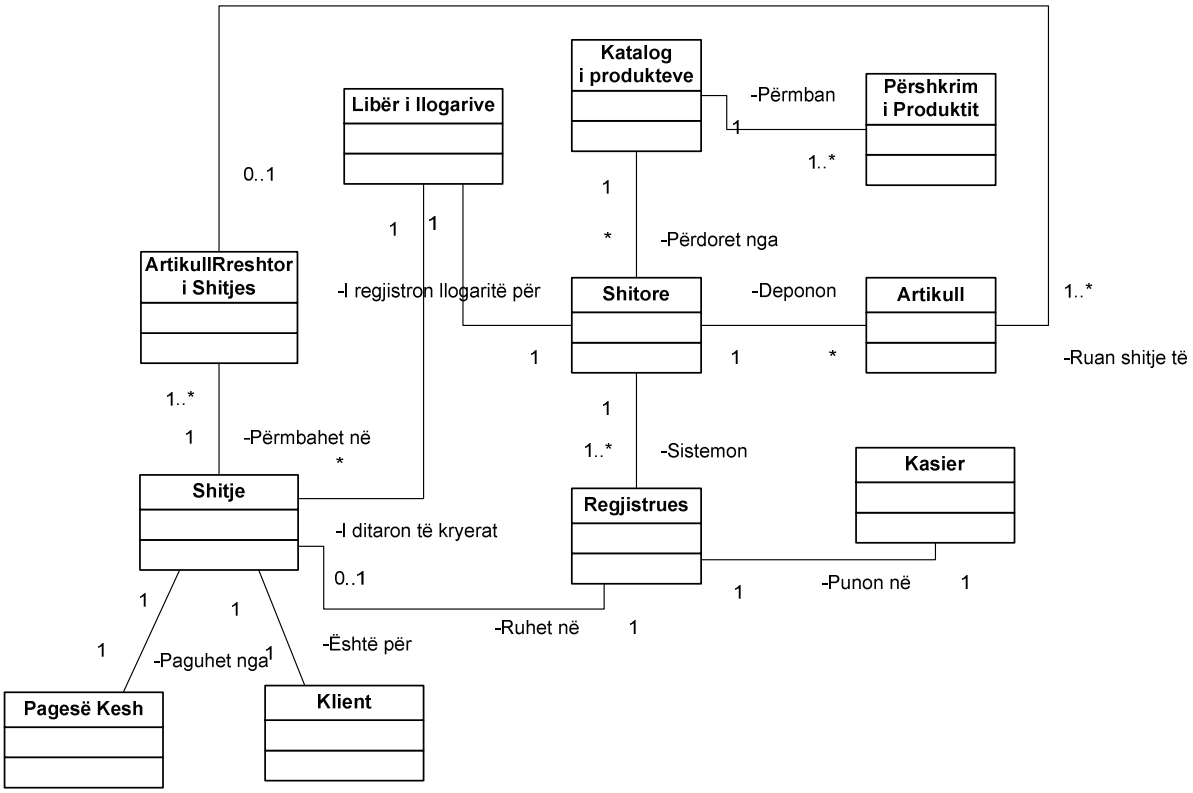
Lista e asociacioneve më të shpeshta

(kapitulli 9)

Kategoria	Shembuj
A është transaksion i lidhur me një transaksion tjetër B	PagesëKesh – Shitje Anulim – Rezervim
A është artikull rreshtor i një transaksioni B	ArtikullRreshtorIShitjes – Shitje
A është produkt apo shërbim për një tranaksion (apo artikull të rreshtit) B	Artikull – ArtikullRreshtorIShitjes Fluturim – Rezervim
A është rol i lidhur me një transaksion B	Klienti – Pagesa Udhëtari – Bileta
A është pjesë fizike apo logjike e B	Tërheqësi – Regjistri Fusha – Tabela Ulëse – Aeroplan
A përmbahet fizikisht apo logjikisht në B	Regjistër – Shitore, Artikull – Raft, Fushë – Tabelë, Udhëtar – Aeroplan
A është përshkrim për B	PërshkrimIProduktit – Artikull PërshkrimIFluturimit – Fluturim
A dihet/shkruhet në ditar/regjistrohet/raportohet/kapet në B	Shitje – Regjistër Figurë – Fushë Rezervim – ManifestIFluturimit
A është anëtar i B	Kasier – Shitore Lojtar – LojëMonopol Pilot – VijëAjrore
A është nënnjësi organizative e B	Departamenti – Shitorja Mirëmbajtje - VijëAjrore
A e përdor ose e menaxhon ose e posedon B	Kasier – Regjistër Lojtar – Figurë Pilot – Aeroplan
A është afër B	ArtikullRreshtorIShitjes – ArtikullRreshtorIShitjes Fushë – Fushë Qytet-Qytet

Modeli i pjesshëm i domenit për NextGen POS

(kapitulli 9)



Atributet - Modeli i pjesshëm i domenit për NextGen POS

(kapitulli 9)

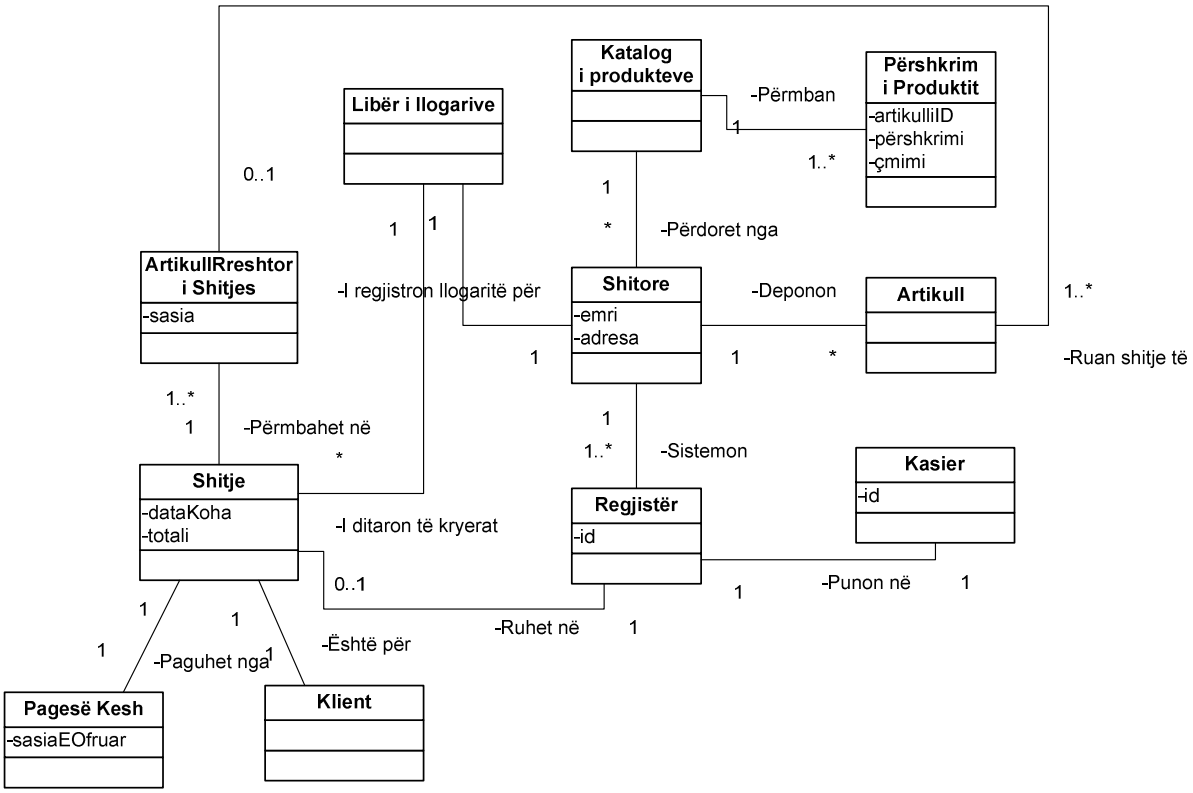
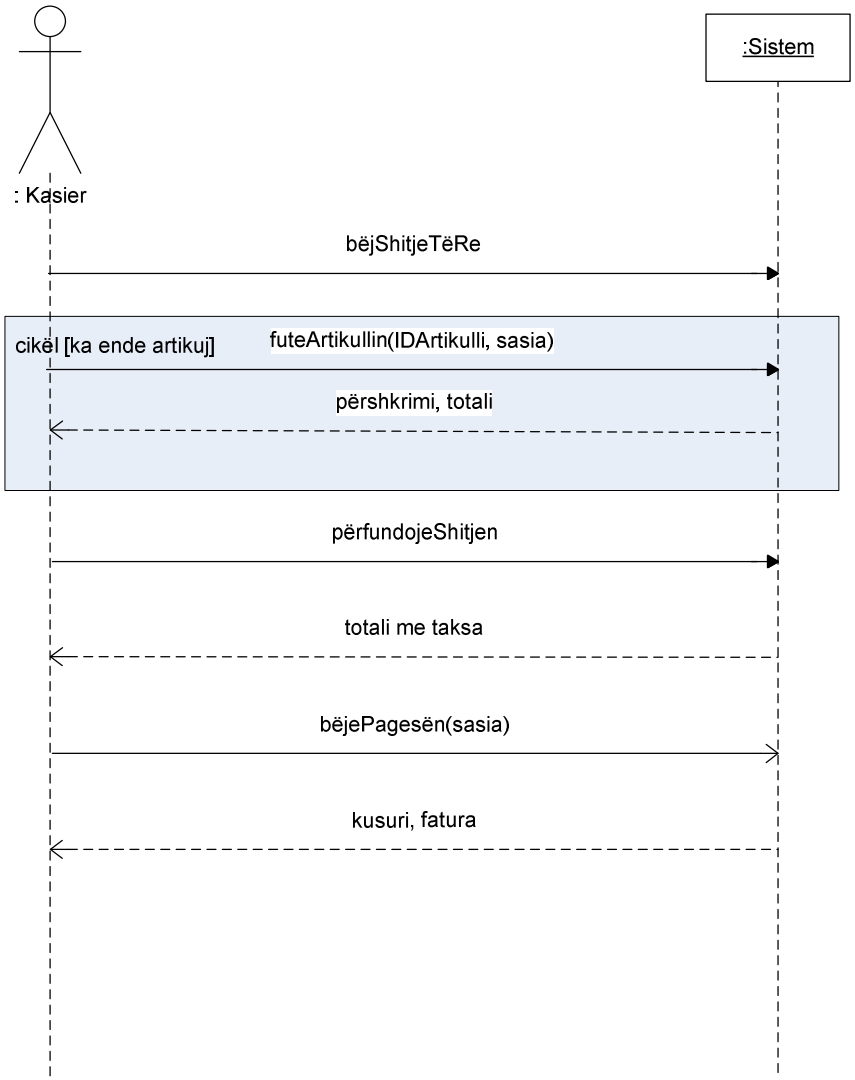


Diagram Sekuencial i Sistemit NextGen

(kapitulli 10)

Skenari i përpunimit të shitjes



Kontrata të operacioneve

(kapitulli 11)

Kontrata K01: bëjShitjeTëRe

Operacioni:	bëjShitjeTëRe()
Ndër – referencat:	Rastet e Përdorimit: Përpunoje Shitjen
Parakushtet:	Asnjë
Paskushtet:	- Është kriju një instancë Shitje sh - Sh është asociu me një Regjistër - Atributet e sh janë inicializu

Kontrata K02: regjistroArtikullin

Veprimi:	regjistroArtikullin(ArtikulliID: ArtikullID, sasia: integer)
Ndër – referencat:	Rastet e Përdorimit: Përpunoje Shitjen
Parakushtet:	Është një shitje në proces
Paskushtet:	- Është kriju një instancë ars e tipit ArtikullRreshtIShitjes (krijimi i instancës) - Ars është asociu me Shitjen aktuale (asociacioni u formua) - Ars.sasia është bë sasia (ndryshimi i atributit) - Ars është shoqëru me një PërshkrimIProduktit, duke u bazu në përputhjen e artikullitID (asociacioni i formuar)

Kontrata K03: mbaroShitjen

Operacioni:	mbaroShitjen()
Ndër – referencat:	Rastet e Përdorimit: Përpuno Shitjen
Parakushtet:	Është një shitje në proces
Paskushtet:	- Është kriju një instancë Pagesë p P.sasiaEOfruar është bë sasia - P është shoqëru me Shitjen aktuale - Shitja aktuale është shoqëru me Shitoren

Arkitektura logjike – shtresat

(kapitulli 13)

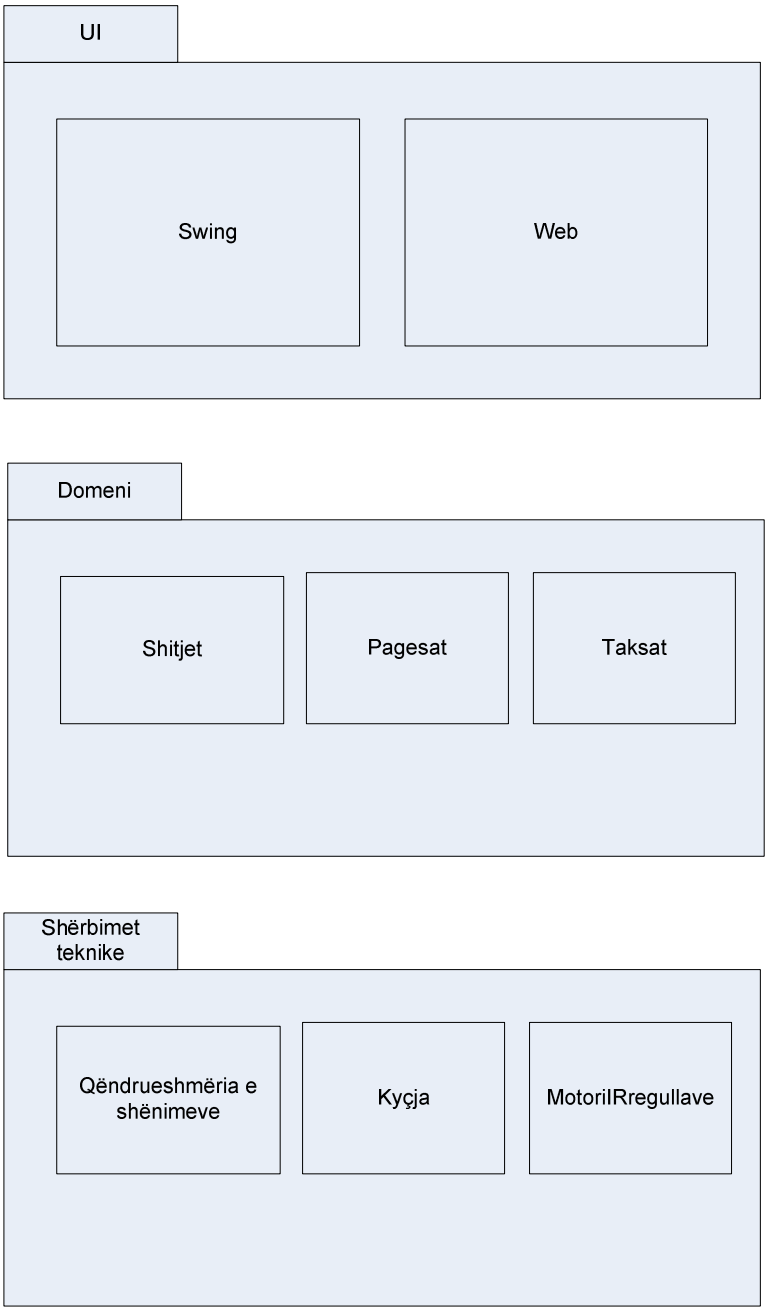
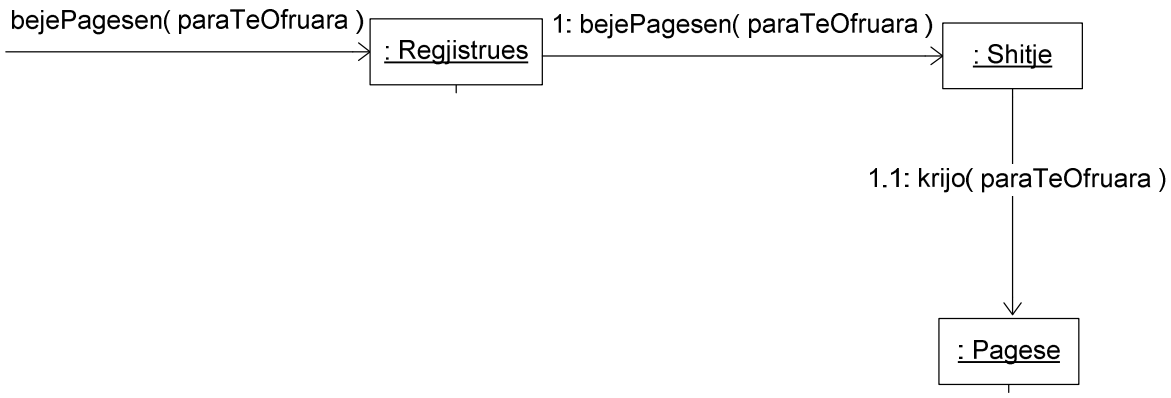


Diagram sekuencial

(kapitulli 15)



Shembull i kodit – rritja e ndryshores

(kapitulli 15)

```
public class Shitje
{
	private Liste<ArtikullRreshtIShitjes> artikujtRreshta =
		new ListeVarg<ArtikullRreshtIShitjes>();

	public TëHolla merreTotalin()
	{
		TëHolla totali = new TëHolla();
		TëHolla nentotali = null;

		for ( ArtikullRreshtIShitjes artikulliRresht : artikujtRreshta)
		{
			nentotali = artikulliRresht.merreNentotalin();
			totali.shtoje(nentotali);
		}

		return totali;
	}

	// ...
}
```

Thirrje asinkronike

(kapitulli 15)

```
public class NisesIOres
{
	public void niseOren()
	{
		Thread t = new Thread( new Ore() );
		t.start();   // thirrja asinkronike e metodes 'run' ne Ore
		System.runFinalization(); // shembull i mesazhit pasues
	}

	// ...

}

// objektet duhet me implementu interfejsin Runnable
// ne Java per m'u perdore ne thread'a te rinj

public class Ore implements Runnable
{
	public void run()
	{
		while (true) // sille perhere ne thread’in (fijen) e vet
		{
			// ...
		}
	}
	// ...
}
```

Disa kontrata me diagrame sekuenciale

(kapitulli 18)

Kontrata K01: bejShitjeTeRe

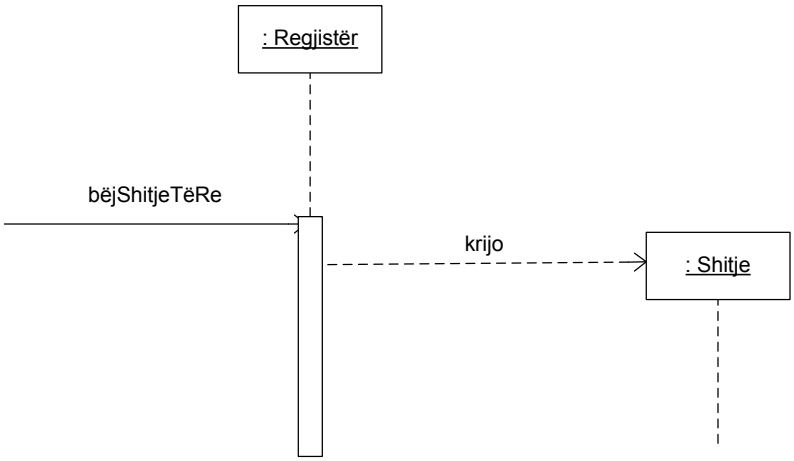
Operacioni: bëjShitjeTëRe()

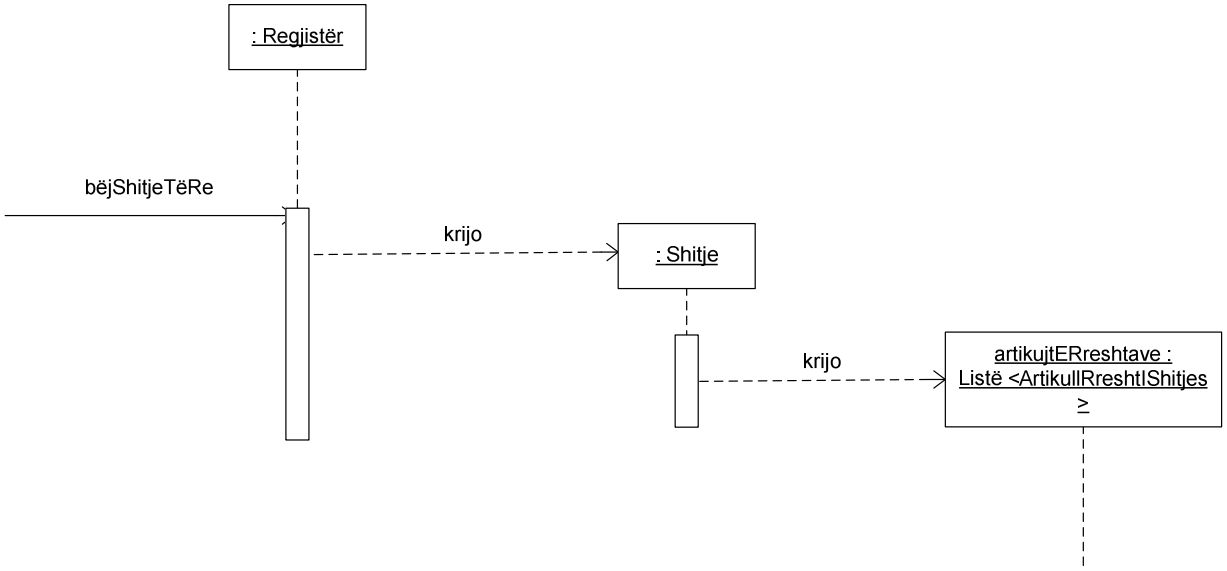
Ndër-referencat: Rastet e Përdorimit: Përpuno Shitjen

Parakushtet: asnjë

Paskushtet:

- Një instancë Shitje s është kriju (krijimi i instancës)
- s është shoqëru me Regjistrin (asociacioni është formu)
- Atributet e s janë inicializu.




: Regjistër
bëjShitjeTëRe
krijo
: Shitje
krijo
artikujtERreshtave :
Listë <ArtikullRreshtIShitjes
>

Kontrata K02: futArtikull

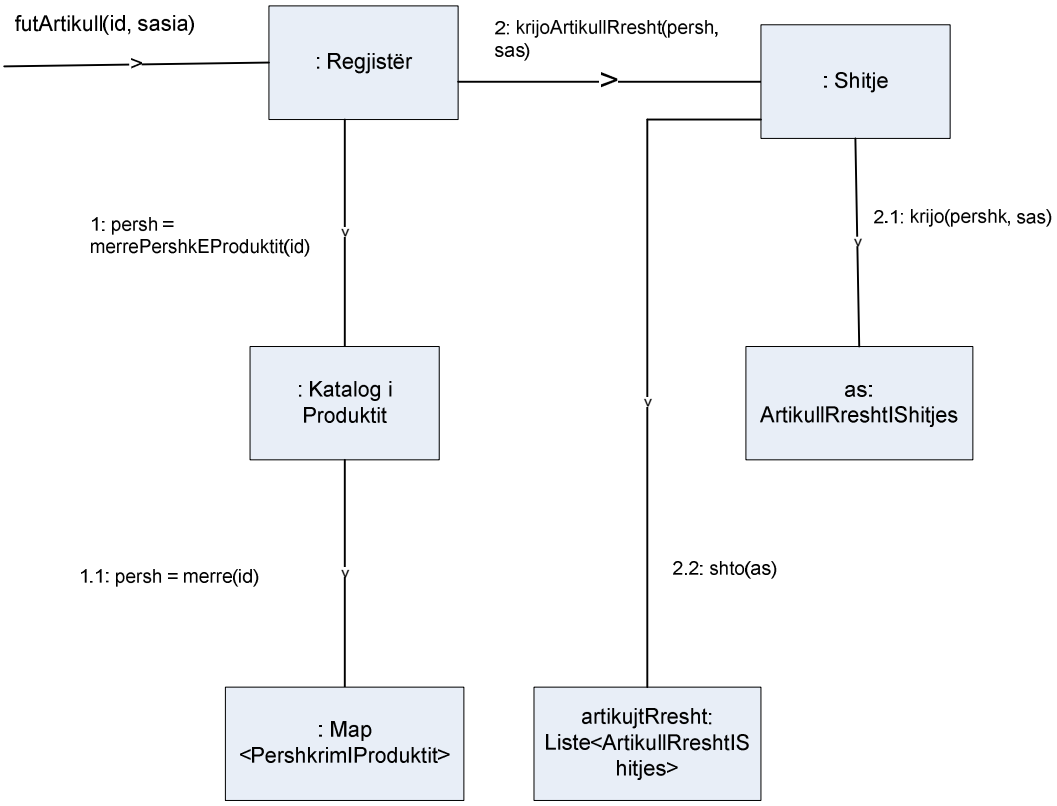
Operacioni: futArtikull(artikullID : ArtikulliID, sasia : integer)

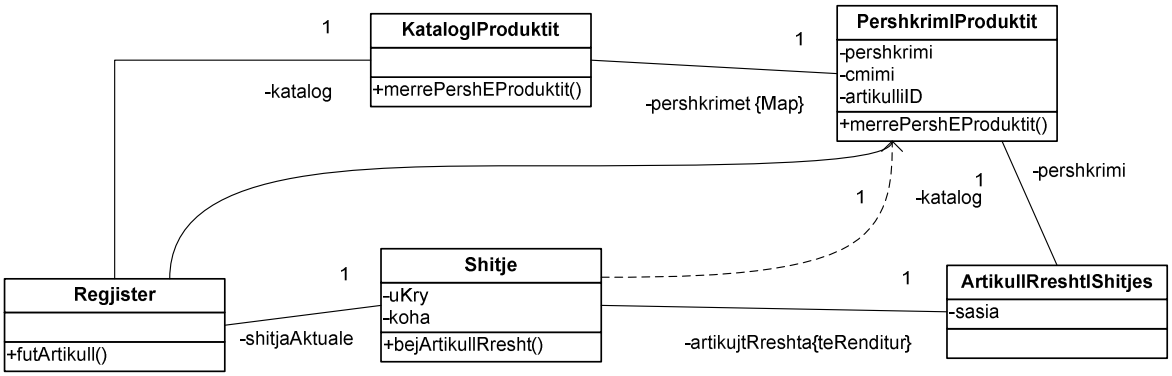
Ndër-referencat: Rastet e Përdorimit: Përpuno Shitjen

Parakushtet: Është një shitje rrugës

Paskushtet:

- Një instancë ArtikullRreshtIShitjes ars është kriju (krijimi i instancës)
- Ars është shoqëru me shitjen aktuale (asociacioni është formu)
- Ars.sasia u bë sasia (ndryshimi i atributeve)
- Ars është shoqëru me një PërshkrimIProduktit, duke u bazu në përputhjen në Id të Artikullit (asociacioni është formu).




KataloglProduktit
+merrePershEProduktit()
PershkrimlProduktit
-pershkrimi
-cmimi
-artikullilD
+merrePershEProduktit()
Regjister
+futArtikull()
Shitje
-uKry
-koha
+bejArtikullResht()
ArtikullReshtlShitjes
-sasia
1
-katalog
-pershkrimet {Map}
-pershkrimi
-shitjaAktuale
-artikujtRreshta{teRenditur}

Kontrata K03: kryejShitjen

Operacioni: kryejShitjen()

Ndër-referencat: Rastet e Përdorimit: Përpuno Shitjen

Parakushtet: Është një shitje rrugës

Paskushtet: Shitja.ështëKry u bë True (ndryshimi i atributit)

Informata e nevojitur për Totalin e Shitjes	**Eksperti i Informatës**
PërshkrimiIProduktit.çmimi	PërshkrimIProduktit
ArtikullRreshtIShitjes.sasia	ArtikullRreshtIShitjes
Të gjithë ArtikujtRreshtaTëShitjes në Shitjen aktuale	Shitja

Kontrata K04: kryejPagesën

Operacioni: kryejPagesën(sasia: Para)

Ndër-referencat: Rastet e Përdorimit: Përpuno Shitjen

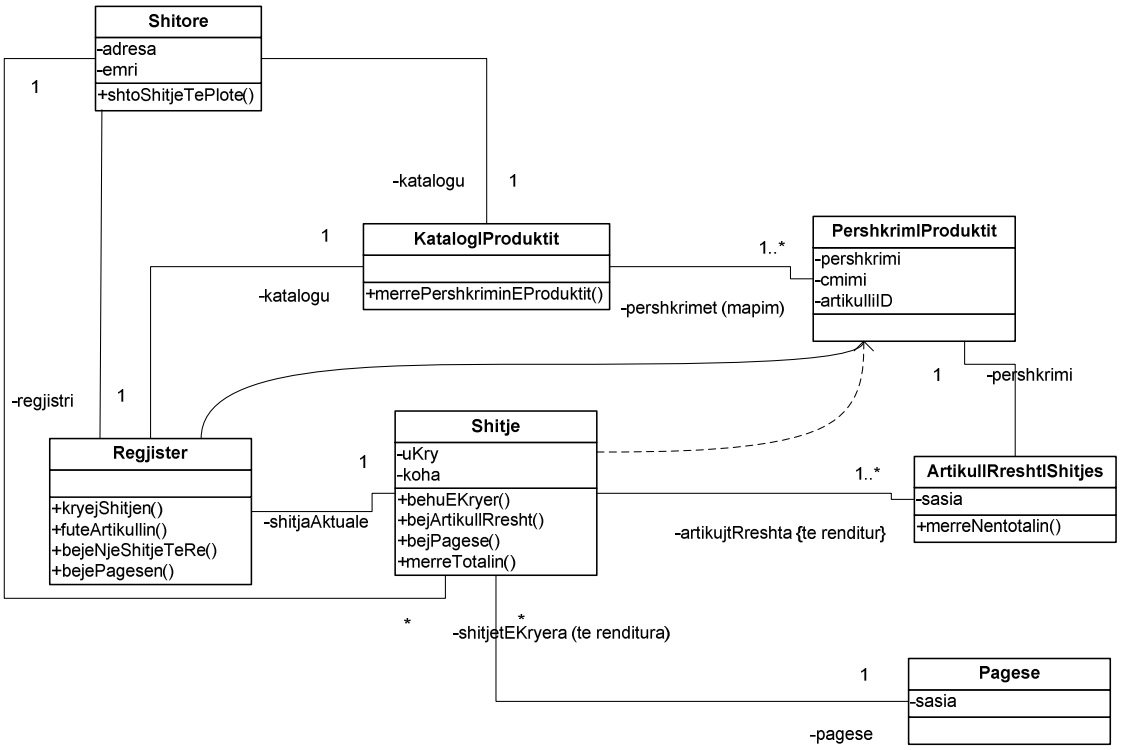
Parakushtet: Është një shitje rrugës

Paskushtet:

- Një instancë Pagesë p është krijju
- p.sasiaEOfruar është bë sasia
- p është shoqëru me shitjen aktuale
- shitja aktuale është asociu me Shitore (për me shtu në ditarin historik të shitjeve të kryera)

Diagram i Dizajnit të Klasave

(kapitulli 18)



Implementim në kod – Hyrje në zgjidhjen NextGen POS

(kapitulli 20)

```
package com.foo.nextgen.domen;

public class Pagese
{
	private Money sasia;

	public Pagese(Money parateEOfruara) { sasia = parateEOfruara; }
	public Money merreSasine() { return sasia; }
}


public class KatalogIProdukteve
{
	private Map<ArtikullID, PershkrimIProduktit>
		pershkrimet = new HashMap()<ArtikullID, PershkrimIProduktit>;

	public KatalogIProdukteve()
	{
		// shenime shembuj
		ArtikullID id1 = new ArtikullID( 100 );
		ArtikullID id2 = new ArtikullID( 200 );
		Money cmimi = new Money( 3 );

		PershkrimIProduktit persh;
		persh = new PershkrimIProduktit( id1, cmimi, "produkti 1" );
		pershkrimet.put( id1, persh );
		persh = new PershkrimIProduktit( id2, cmimi, "produkti 2" );
		pershkrimet.put( id2, persh );
	}

	public PershkrimIProduktit merrePershkriminEProduktit(ArtikullID id)
	{
		return pershkrimet.get( id );
	}
}


public class Regjistrues
{
	private KatalogIProdukteve katalogu;
	private Shitje shitjaAktuale;

	public Regjistrues( KatalogIProdukteve katalogu )
	{
		this.katalogu = katalogu;
	}

	public void kryejShitjen()
	{
		shitjaAktuale.behuEKryer();
	}
```

```
	public void futeArtikullin( ArtikullID id, int sasia )
	{
		PershkrimIProduktit persh =
			katalogu.merrePershkriminEProduktit( id );
		shitjaAktuale.bejArtikullResht( persh, sasia );
	}

	public void bejShitjeTeRe()
	{
		shitjaAktuale = new Shitje();
	}

	public void bejePagesen( Money parateEOfruara )
	{
		shitjaAktuale.bejePagesen( parateEOfruara );
	}
}



public class PershkrimIProduktit
{
	private ArtikulliID id;
	private Money cmimi;
	private String pershkrimi;

	public PershkrimIProduktit
		( ArtikullID id, Money cmimi, String pershkrimi )
	{
		this.id = id;
		this.cmimi = cmimi;
		this.pershkrimi = pershkrimi;
	}

	public ArtikullID merreArtikullinID() { return id; }

	public Money merreCmimin() { return cmimi; }

	public String merrePershkrimin() { return pershkrimi; }
}


public class Shitje
{
	private List<ArtikullRreshtIShitjes> artikujtRreshta =
		new ArrayList()<ArtikullRreshtIShitjes>;
	private Date data = new Date();
	private eshteKryer = false;
	private Pagese pagesa;

	private Money merreBalansin()
	{
		return pagesa.merreSasine().minus( merreTotalin() );
	}

	public void behuEKryer() { eshteKryer = true; }

	public boolean eshteKryer() { return eshteKryer; }
```

```
	public void bejArtikullResht
		( PershkrimIProduktit persh, int sasia )
	{
		artikujtRreshta.add(new ArtikullRreshtIShitjes(persh, sasia));
	}

	public Money merreTotalin()
	{
		Money totali = new Money();
		Money nentotali = null;

		for (ArtikullRreshtIShitjes artikulliResht : artikujtRreshta)
		{
			nentotali = artikulliResht.merreNentotalin();
			totali.add( nentotali );
		}
		return totali;
	}

	public void bejePagesen( Money parateEOfruara )
	{
		pagesa = new Pagese( parateEOfruara );
	}
}


public class ArtikullRreshtIShitjes
{
	private int sasia;
	private PershkrimIProduktit pershkrimi;

	public ArtikullRreshtIShitjes(PershkrimIProduktit persh, int sasia)
	{
		this.pershkrimi = persh;
		this.sasia = sasia;
	}

	public Money merreNentotalin()
	{
		return pershkrimi.merreCmimin().times( sasia );
	}
}

public class Shitore
{
	private KatalogIProdukteve katalogu = new KatalogIProdukteve();
	private Regjistrues regjistruesi = new Regjistrues( katalogu );

	public Regjistrues merreRegjistruesin() { return regjistruesi; }
}
```

Implementim në kod – Hyrje në zgjidhjen Monopoly

(kapitulli 20)

```
package com.foo.monopoly.domen;

public class Fushe
{
	private String emri;
	private Fushe fushaTjeter;
	private int indeksi;

	public Fusha( String emri, int indeksi )
	{
		this.emri = emri;
		this.indeksi = indeksi;
	}

	public void caktoFushenTjeter(Fushe f)
	{
		this.fushaTjeter = f;
	}

	public Fushe merreFushenTjeter()
	{
		return fushaTjeter;
	}

	public String merreEmrin()
	{
		return emri;
	}

	public int merreIndeksin()
	{
		return indeksi;
	}
}


public class Figure
{
	private Fushe lokacioni;

	public Figure( Fushe lokacioni )
	{	this.lokacioni = lokacioni; }

	public Fushe merreLokacionin()
	{
		return lokacioni;
	}
}

public class Zar
{
	public static final int MAX = 6;
	private int vleraEFytyres;
```

```
	public Zar()
	{
		sillu();
	}

	public void sillu()
	{
		vleraEFytyres = (int) ( ( Math.random( ) * MAX ) + 1 );
	}

	public int merreVlerenEFytyres( )
	{
		return vleraEFytyres;
	}
}



public class Tabele
{
	private static final int MADHESIA = 40;
	private List fushat = new ArrayList(MADHESIA);

	public Tabela()
	{
		ndertoFushat();
		lidhiFushat();
	}

	public Fushe merreFushen(Fushe fillimi, int distanca)
	{
		int indeksiIFundit =
			(fillimi.merreIndeksin() + distanca) % MADHESIA;

		return (Fushe) fushat.get(indeksiIFundit);
	}

	public Fushe merreFushenFillestare()
	{
		return (Fushe) fushat.get(0);
	}

	private void ndertoFushat()
	{
		for (int i = 1; i <= MADHESIA; i++)
		{
			nderto(i);
		}
	}

	private void nderto(int i)
	{
		Fushe s = new Fushe("Fusha " + i, i - 1);
		fushat.add(s);
	}
```

```
	private void lidhiFushat()
	{
		for (int i = 0; i < ( MADHESIA - 1); i++ )
		{
			lidhe(i);
		}

		Fushe i_pari = (Fushe) fushat.get(0);
		Fushe i_fundit = (Fushe) fushat.get(MADHESIA - 1);
		i_fundit.caktoFushenTjeter(i_pari);
	}

	private void lidhe(int i)
	{
		Fushe aktualja = (Fushe) fushat.get(i);
		Fushe tjetra = (Fushe) fushat.get(i + 1);
		aktualja.caktoFushenTjeter(tjetra);
	}
}


public class Lojtar
{
	private String emri;
	private Figure figura;
	private Tabele tabela;
	private Zar[] zaret;

	public Lojtar(String emri, Zar[] zaret, Tabele tabela)
	{
		this.emri = emri;
		this.zari = zari;
		this.tabela = tabela;
		figura = new Figure(tabela.merreFushenFillestare());
	}

	public void merreRendin()
	{
		// sille zarin
		int totaliISilljeve = 0;
		for (int i = 0; i < zaret.length; i++)
		{
			zari[i].sillu();
			totaliISilljeve += zari[i].merreVlerenEFytyres();
		}

		Fushe lokIRi = tabela.merreFushen(
			figura.merreLokacionin, totaliISilljeve);
		figura.caktoLokacionin(lokIRi);
	}

	public Fushe merreLokacionin()
	{
		return figura.merreLokacionin();
	}
```

```
	public String merreEmrin()
	{
		return emri;
	}
}


public class LojaMonopol
{
	private static final int TOTALI_I_RUNDEVE = 20;
	private static final int TOTALI_I_LOJTAREVE = 2;
	private List lojtaret = new ArrayList( TOTALI_I_LOJTAREVE );
	private Tabele tabela = new Tabele( );
	private Zar[] zaret = { new Zar(), new Zar() };

	public LojaMonopol( )
	{
		Lojtar l;
		l = new Lojtar("Kale", zaret, tabela );
		lojtaret.add( l );
		l = new Lojtar("Kerr", zaret, tabela );
		lojtaret.add( l );
	}

	public void luajeLojen( )
	{
		for ( int i = 0; i < TOTALI_I_RUNDEVE; i++ )
		{
			luajeRundin();
		}
	}

	public List merriLojtaret( )
	{
		return lojtaret;
	}

	private void luajeRundin( )
	{
		for (Iterator iter = lojtaret.iterator( ); iter.hasNext( ); )
		{
			Lojtar lojtari = (Lojtar) iter.next();
			lojtari.merreRendin();
		}
	}
}
```

Fjalor i publikimit

Collaboration diagram - diagram i bashkëpunimit (kolaborues) - diagram që i tregon ndërveprimet e organizuara rreth strukturës së një modeli, duke i përdorë ose klasifikuesit dhe asociacionet, ose instancat dhe lidhjet. Për dallim nga diagrami sekuencial, një diagram kolaborues i tregon relacionet ndërmjet instancave. Diagramet sekuenciale dhe kolaboruese (të bashkëpunimit) shprehin informata të ngjashme, por i shfaqin ato në mënyra të ndryshme. Shih SSD.

CRC card - Class-Responsibility-Collaborator card - kartelat Përgjegjësitë dhe Bashkëpunëtorët e Klasës janë vegël e analizës që përdoret në dizajnin e softuerit të orientuar kah objektet. Ato janë propozu nga Ward Cunningham and Kent Beck. Ato zakonisht përdoren gjatë përcaktimit të parë se cilat klasa nevojiten dhe si do të veprojnë ato ndërmjet vete.

Kartelat CRC zakonisht krijohen nga kartelat indeks në të cilat shkruhen:

1. Emri i klasës
2. Klasat e saj Mbi dhe Nën (nëse aplikohet)
3. Përgjegjësitë e klasës
4. Emrat e klasëve tjera me të cilat do të bashkëpunojë klasa për m'i plotësu përgjegjësitë e veta.
5. Autori

diagram i bashkëpunimit (kolaborues) - shih collaboration diagram

diagrami sekuencial - shih SSD

DCD - Design Class Diagram - Diagrami i Dizajnit të Klasave i tregon definicionet e klasave të softuerit. Ai bazohet në diagramin e bashkëpunimit (collaboration diagram). Dukshmëria e atributeve tregohet për lidhjet e përhershme. Klasat tregohen të listuara me atributet dhe metodat e veta të thjeshta.

GRASP - General Responsibility Assignment Software Patterns (ose nganjëherë Principles) - Paternat (Modelet, ose nganjëherë Principet) e Përgjithshme Softuerike të Ndarjes së Përgjegjësive. Përdoret në dizajnin e orientuar kah objektet, dhe jep udhëzime për m'iu nda përgjegjësi klasave dhe objekteve.

GUI - Graphical User Interface - Ndërfaqja Grafike e Përdoruesit

GoF- Gang of Four - Banda e Katërshes

JDBC - Java DataBase Connectivity - Lidhshmëria e Javës me Baza të Shënimeve - interfejs i programimit që i lejon aplikacionet Java m'iu qasë një baze përmes gjuhës SQL. Meqë interpretuesit e Javës (Makinat Virtuale të Javës, JVM) janë në dispozicion për të gjitha platformat e mëdha klient, kjo lejon m'u shkru aplikacion me baza i pavarur nga platforma. Më 1996, JDBC ka qenë zgjerimi i parë në platformën Java. JDBC është homologu Java i ODBC të Microsoft.

LRG - Low Representational Gap - Zbraztësia e Ulët e Reprezentimit - Përdor klasa të softuerit që inspirohen nga domeni për me përmirësu kuptueshmërinë, duke përdorë emra dhe terma nga domeni i problemit.

Modeli i Dizajnit - është model i objekteve që e përshkruan realizimin e rasteve të përdorimit, dhe shërben si abstraksion i modelit të implementimit dhe i kodit të tij burimor. Modeli i dizajnit përdoret si e dhënë hyrëse esenciale për aktivitetetet në implementim dhe testim.

Modeli i Domenit - i përmbledh tipet më të rëndësishme të objekteve në kontekst të domenit. Objektet e domenit i përfaqësojnë entitetet që ekzistojnëa po ngjarjet që ndodhin në ambientin në të cilin punon sistemi. Modeli i domenit është nënpjesë e modelit të objekteve të biznesit.

OO - Object Oriented - i Orientuar kah Objektet. Shih OOA/D

OOA - Shih OOA/D

OOA/D - Object Oriented Analysis and Design - Analiza dhe Dizajni i Orientuar kah Objektet. OOAD është qasje e inxhinierimit të softuerit që e modelon një sistem si grup të objekteve ndërvepruese. Secili objekt e përfaqëson një entitet të interesit në sistemin që modelohet, dhe karakterizohet nga klasa e vet, gjendja e vet (elementet shënime), dhe sjellja e vet. Mund të krijohen modele të ndryshme për me tregu strukturën statike, sjelljen dinamike, dhe shpërndarjen gjatë ekzekutimit të këtyre objekteve bashkëpunuese. Ka disa notacione të ndryshme për m'i paraqitë këto modele, siç është UML.

- Analiza e orientuar kah objektet (OOA) shikon në domenin e problemit, me qëllim për me prodhu një model konceptual të informatave që ekzistojnë në zonën që po analizohet. Modelet e analizës nuk i marrin në konsiderim kufizimet e implementimit që munden me ekzistu, si konkurenca, shpërndarja, persistenca, apo se si do të ndërtohet sistemi. Kufizimet e implementimit trajtohen gjatë dizajnit të orientuar kah objektet (OOD). Analiza bëhet para Dizajnit.
- Dizajni i orientuar kah objektet (OOD) e transformon modelin konceptual të prodhuar në analizën e orientuar kah objektet për m'i trajtu kufizimet që imponohen nga arkitektura e zgjedhur dhe çfarëto kufizime jo-funksionale - teknologjike apo ambientore, siç është gjerësia e brezit të transaksioneve, koha e përgjigjjes, platforma e ekzekutimit, ambienti i zhvillimit apo gjuha programuese.

OOD - Object Oriented Design - Dizajni i Orientuar kah Objektet - Shih OOA/D

POS - Point of Sale. Pikë e shitjes. Një biznes apo një vend ku mund të blehet një produkt apo një shërbim.

POSA - Pattern- Oriented Software Architecture - Arkitekturë e Softuerit e Orientuar kah Paternat

RDB - Relational Database - Bazë relacionale e shënimeve

RDD - Responsibility-Driven Design, Dizajn i Udhëhequr nga Përgjegjësitë

RPC - Remote Procedure Call - Thirrje e procedurës nga distanca

Sequence Diagram - shih SSD

SQL - Structured Query Language - Gjuhë e Strukturuar për Kërkesa (Pyetësorë)

SSD - System Sequence Diagram - Diagram Sekuencial i Sistemit - Diagram që i tregon ndërveprimet e objekteve të aranzhuara në sekuencë kohore. Në veçandi, i tregon objektet që marrin pjesë në ndërveprim dhe renditjen (sekuencën) e mesazheve të shkëmbyera. Për dallim nga diagrami i bashkëpunimit (collaboration diagram), diagrami sekuencial i përfshin sekuencat e kohës por nuk i përfshin relacionet e objekteve. Një diagram sekuencial mund të ekzistojë në formë gjenerike (i përshkruan të gjithë skenarët e mundshëm) dhe në formë instancë (e përshkruan një skenar aktual). Diagramet sekuenciale dhe kolaboruese (të bashkëpunimit) shprehin informata të ngjashme, por i shfaqin ato në mënyra të ndryshme. Shih collaboration diagram.

UI - User Interface - Ndërfaqe e Përdoruesit

UML - Unified Modeling Language - Gjuha e Unifikuar e Modelimit

UP - Unified Process - Procesi i Unifikuar

Përmbledhje e GRASP

General Responsibility Assignment Software Patterns or Principles

Paternat apo Principet e Përgjithshme e Përgjithshme Softuerike të Ndarjes së Përgjegjësive

<table>
<tr><th>Paterni/Principi</th><th colspan="2">Përshkrimi</th></tr>
<tr><td rowspan="2">Eksperti i Informatave</td><td colspan="2">Një princip i përgjithshëm i dizajnit të objekteve dhe i ndarjes së përgjegjësive?</td></tr>
<tr><td colspan="2">Ndaja përgjegjësinë ekspertit të informatave – klasës që i ka informatat e nevojshme me plotësu përgjegjësinë.</td></tr>
<tr><td rowspan="4">Krijuesi</td><td colspan="2">Kush krijon? (Vëreni se Fabrika është zgjidhje e shpeshtë alternative.)

Ngaja klasës B përgjegjësinë për me kriju një instancë të klasës A nëse njëra nga këto është e vërtetë:</td></tr>
<tr><td>1. B e përmban A</td><td>4. B e regjistron A</td></tr>
<tr><td>2. B e agregon A</td><td>5. B e përdor ngushtësisht A</td></tr>
<tr><td colspan="2">3. B i ka shënimet inicializuese për A</td></tr>
<tr><td>Kontrolluesi</td><td colspan="2">Cili objekt i pari përtej shtresës UI e pranon dhe e koordinon (“e kontrollon”) një operacion të sistemit?

Ngaja përgjegjësinë një objekti që e përfaqëson njërin nga këto zgjedhje:

1. E përfaqëson “sistemin” e përgjithshëm, një “objekt rrënjë”, një pajisje brenda të cilës ekzekutohet softueri, apo një nënsistem madhor (këto janë të gjitha variacione të një kontrolluesi fasadë).
2. E përfaqëson një skenar të rastit të përdorimit brenda të cilit ndodh operacioni i sistemit (kontrollues i rastit të përdorimit apo i sesionit)</td></tr>
<tr><td>Çiftimi i Ulët (vlerësues)</td><td colspan="2">Si me zvogëlu ndikimin e ndryshimit?

Ndaji përgjegjësitë ashtu që çiftimi (i panevojshëm) të mbetet i vogël.

Përdore këtë princip për m’i vlerësu alternativat.</td></tr>
<tr><td>Kohezioni i Lartë (vlerësues)</td><td colspan="2">Si m’i mbajtë objektet të fokusuara, të kuptueshme, dhe të menaxhueshme, dhe si efekt anësor, me përkrahë Çiftimin e Ulët?

Ndaji përgjegjësitë ashtu që kohezioni të mbetet i lartë. Përdore këtë për m’i vlerësu alternativat.</td></tr>
</table>

Paterni/Principi	Përshkrimi
Polimorfizmi (Shumëformësia)	Kush është përgjegjës kur sjellja ndryshon sipas tipit? Kur alternativat apo sjelljet e afërta ndryshojnë sipas tipit (klasës), ndaja përgjegjësinë për sjelljen – duke i përdorë operacionet polimorfike – tipeve për të cilat ndryshon sjellja.
Fabrikimi i Pastër	Kush është përgjegjës kur je i pashpresë, dhe nuk don me shkelë kohezionin e lartë dhe çiftimin e ulët? Ngaja një grup shumë koheziv të përgjegjësive një klase artificiale appo përshtatshme "të sjelljes" që nuk e përfaqëson një koncept të domenit të problemit – diçka të trilluar, për me përkrahë kohezionin e lartë, çiftimin e ulët, dhe ripërdorimin.
Indireksioni	Si m'i nda përgjegjësitë për me iu shmangë çiftimit të drejtpërdrejtë? Ndaja përgjegjësinë një objekti të ndërmjetëm për me ndërmjetësu ndërmjet komponentave apo serviseve tjera, ashtu që ato nuk janë të çiftuara drejtpërdrejt.
Variacionet e Mbrojtura	Si me ua nda përgjegjësitë objekteve, nënsistemeve, dhe sistemeve ashtu që ndryshimet apo jostabiliteti në këto elemente të mos kenë ndikim të padëshiruar në elementet tjera? Identifikoji pikat e ndryshimit apo jostabilitetit të parashikuar; ndaji përgjegjësitë për me kriju një "interfejs" stabil përreth tyre.