UNIVERZITET METROPOLITAN BEOGRAD

SE311 - PROJEKTOVANJE I ARHITEKTURA SOFTVERA

Upotreba šablona projektovanja softvera

Lekcija 06

PRIRUČNIK ZA STUDENTE

SE311 - PROJEKTOVANJE I ARHITEKTURA SOFTVERA

Lekcija 06

UPOTREBA ŠABLONA PROJEKTOVANJA SOFTVERA

Uvod

UVOD

Statički pogled na softver omogućava primenu šablona projektovanja u okviru dijagrama klasa. Šabloni se ponavljaju po nekoliko puta u procesu projektovanja softvera i na taj način razvojni tim unapređuje znanje i korišćenje specifičnog šablona projektovanja.
Često se dešava da se određeni delovi modela sistema koriste na nekim drugim mestima u sistemu uz izmene ukoliko je to potrebno. Takvi delovi sistema koji se ponavljaju nazivaju se šabloni. Većina takvih šablona je dokumentovana i dostupna svim članovima razvojnog tima koji ga koriste.
U okviru ove lekcije predstavljeni su šabloni projektovanja koji se koriste u procesu modelovanja i projektovanja softvera. Pored tih šablona postoje i druge oblasti u kojima se šabloni koriste kao što su biznis, dijagnostikovanje mehaničkih problema, snimanje fotografija. Šablon projektovanja treba da sadrži rešenje koje efektno može da reši identifikovani problem u određenom kontekstu. Šablon mora biti detaljno dokumentovan i opisan u razumljivom obliku tako da članovi razvojnog tima mogu da definišu kada i kako će koristiti specifični šablon. Primena šablona projektovanja najčešće se uči iz iskustva drugih projektanata.
Pored imena, svaki šablon projektovanja treba da ima i:

- **Kontekst:** Opšta situacija u kojoj se primenjuje šablon projektovanja
- **Problem:** Rečenica ili pasus koja objašnjava glavni problem koji se rešava
- **Ograničenja:** Pitanja ili pretpostavke koje je potrebno uzeti u obzir prilikom rešavanja problema
- **Rešenja:** Preporučeni način rešavanja problema u datom kontekstu
- **Anti-šablon**: Rešenja koja su inferiorna ili ne funkcionišu u datom kontekstu
- **Srodni šabloni**: Šabloni projektovanja koji su slični odabranom šablonu. Mogu se koristiti kao varijacije ili proširenja određenih slučajeva
- **Reference:** Reference odabranog šablona projektovanja

Šablon projektovanja treba da bude prikazan kroz jednostavni dijagram uz koji je potrebno kratko objašnjenje upotrebom narativnog pisanja. Kroz ovu lekciju predstavljeni su šabloni projektovanja koji se mogu predstaviti dijagramima klasa korišćenjem UML jezika. Prva tri šablona nazivaju se šabloni modelovanja jer se pojavljuju u modelima domena mnogo pre faze projektovanja softvera. Ostali opisani šabloni projektovanja su šabloni dizajna jer uključuju detalje koji se pojavljuju tek u fazi projektovanja.

UVOD - ISHODI UČENJA

Ova lekcija vam obezbeđuje sledeće ishode učenja:

- Razumevanje upotrebe šablona Abstraction–Occurrence, General Hierarchy, Player–Role, Delegation, Immutable, Read-Only Interface i Proxy.
- Razumevanje konteksta, problema i ograničenja navedenih šablona.

- Primena šablona i razumevanje primera nepravilne upotrebe navedenih šablona na softverskoj arhitekturi

UVODNI VIDEO LEKCIJE

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 1

Šablon Abstraction–Occurrence

KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA ABSTRACTION-OCCURRENCE

Šablon iz grupe šablona modelovanja koji se najčešće predstavlja korišćenjem dijagrama klasa koji predstavlja deo sistemskog modela domena.

Kontekst: Često se u modelu domena nalazi set objekata koji se nazivaju pojave (occurrences). Skup navedenih objekata deli određene informacije između sebe ali se i razlikuju jedni od drugih. Primeri scenarija u kojima je moguće koristiti ovaj šablon projektovanja su:

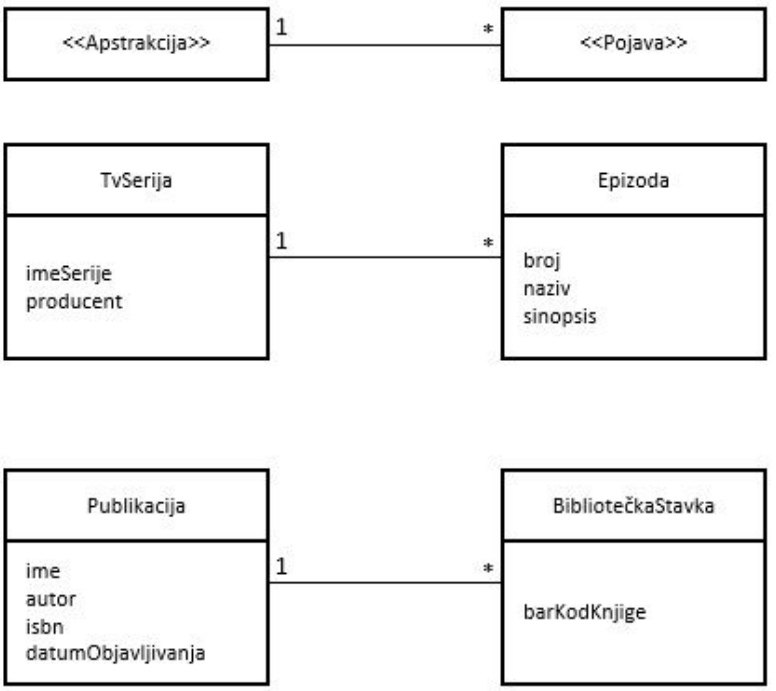
- Sve epizode televizijske serije. Epizode imaju istog producenta, naziv serije a različite priče po epizodi.
- Letovi koji svakog dana poleću za istu destinaciju. Letovi imaju isti broj leta a različite datume, putnike i posadu.
- Kopije iste knjige u biblioteci. Kopije knjiga imaju isti naziv i autora a različit barkod i pozajmljene su od različitih korisnika.

Problem: Koji je najbolji način za prezentovanje takvih setova pojava u dijagramu klasa?
Ograničenja: Potrebno je prikazati članove svakog skupa pojava bez dupliranja zajedničkih informacija. Dupliranje informacija bi iskoristilo prostor i zahtevalo bi menjanje svih pojava u slučaju menjanja zajedničkih informacija. Takođe, potrebno je izbeći rizik od nekonzistentnosti koja bi rezultovana promenu zajedničkih informacija samo u određenim objektima. Potrebno je modelovati rešenje koje omogućava maksimalnu fleksibilnost sistema.

REŠENJE I PRIMER UPOTREBE ŠABLONA ABSTRACTION-OCCURRENCE

Dati su primeri rešenja upotrebe šablona Abstraction-Occurrence

Rešenje: Kreirati klasu <<Apstrakcija>> koja sadrži podatke koji su zajednički za sve članove skupa pojava. Nakon toga, potrebno je kreirati klasu <<Pojava>> koja predstavlja pojave apstrakcije. Ove klase je potrebno povezati sa asocijacijom od jedne do druge klase. Primer je prikazan na slici 1.



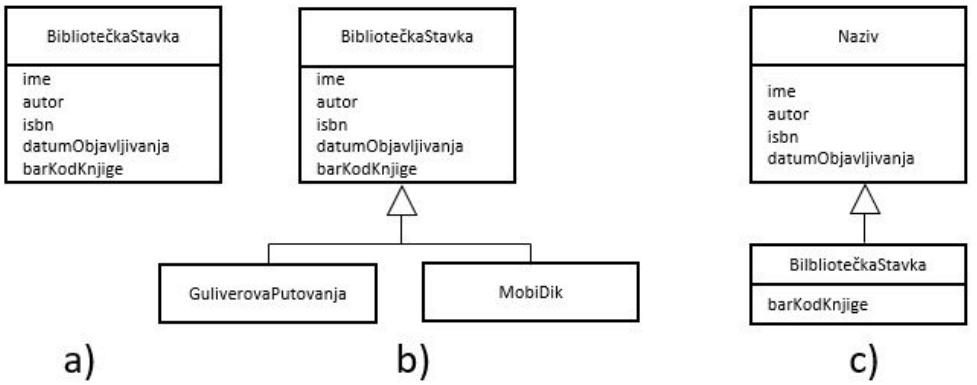
Slika 1.1 Primer upotrebe šablona Abstraction Occurrence [Izvor: Lethbridge [2]]

Primer: Moguće je kreirati klasu <Apstrakcija> pod nazivom "TvSerija" a zatim je potrebno kreirati klasu <pojava> pod imenom "Epizoda". Takođe, moguće je napraviti <apstrakcija> klasu pod imenom "Publikacija" koja će sadržati ime, autora, isbn i datum objavljivanja publikacije. Klasa <pojava> koja odgovara takvoj klasi bila bi "BibliotečkaStavka" koja bi sadržala bar kod publikacije (knjige). (primer na slici 1).

PRIMERI NEPRAVILNE UPOTREBE ŠABLONA ABSTRACTION-OCCURRENCE

Kroz primere nepravilne upotrebe šablona Abstraction-Occurence moguće je uvideti prednosti ali i nedostatke opisanog šablona.

Anti-šablon: Primeri pogrešne upotrebe šablona prikazani su u nastavku na slici 2.
Prikazana je upotreba jedne klase iz koje bi bilo potrebno duplirati informacije na više različitih kopija knjige. Ukoliko imamo knjigu "MobiDik", različite kopije mogu pozajmiti različiti korisnici pa bi onda kopije morale biti odvojene. Knjiga ima isto ime, autora, ISBN broj i datum objavljivanja.



Slika 1.2 Primer nepravilne upotrebe šablona Abstraction-Occurrence [Izvor: Lethbridge [2]]

Drugi primer loše upotrebe predstavljen je na slici 2 pod b) gde je prikazan primer kreiranja odvojenih podklasa za svaki naslov ("GuliverovaPutovanja" i "MobiDik"). Informacije kao što su ime, autor i slično i dalje bi bili duplirani u svakoj podklasi. Ovakav pristup ograničava i fleksibilnost sistema - knjigu nije moguće dodati bez dodavanja nove klase unutar sistema.
Na slici 2 pod c) prikazano je rešenje gde je od klase pojave napravljenja superklasa apstrakcije. Ovakav pristup ne može rešiti uočeni problem jer iako su atributi superklase nasleđeni od podklasa podaci nisu. Na primer, iako je atribut "ime" definisan u superklasi "Naziv" i dalje je potrebno odrediti vrednost ovog atributa za svaki "BibliotečkaStavka".

Poglavlje 2

Šablon General Hierarchy

KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA GENERAL HIERARCHY

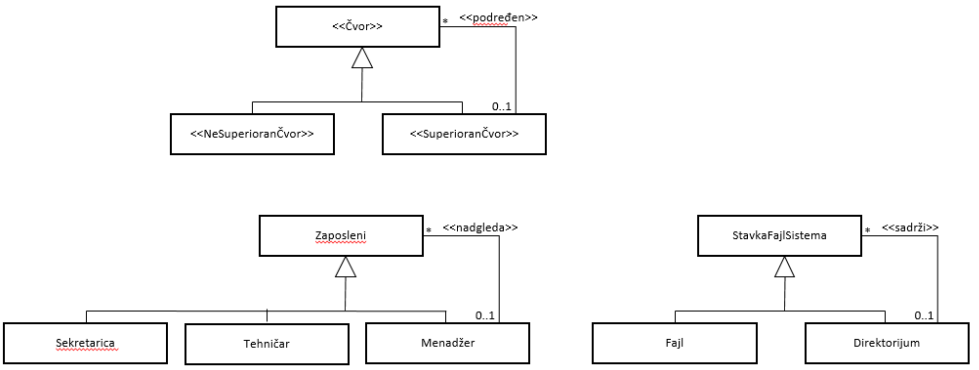
Šablon se može primeniti na skup objekata koji imaju hijerarhijski odnos i mogu biti predstavljeni dijagramom klasa.

Kontekst: Primer može biti odnos između menadžera i članova tima koji se nalaze ispod njega, zatim direktorijuma (foldera), poddirektorijuma i datoteka u njima. Svaki objekat u tako kreiranoj hijerarhiji može imati nula ili više objekata iznad u hijerarhiji i nula ili više objekata ispod njih. Takođe, određeni objekti ne mogu imati objekte ispod sebe u hijerarhiji.
Problem: Kako nacrtati dijagram klasa koji predstavlja hijerarhiju objekata u kojima neki objekti ne mogu imati objekte ispod sebe u hijerarhiji?
Ograničenja: Fleksibilan način predstavljanja hijerarhije koja sprečava da određeni objekti imaju objekte ispod sebe u hijerarhiji. Takođe, potrebno je naglasiti da svi objekti dele zajedničke funkcije.

REŠENJE I PRIMER UPOTREBE ŠABLONA GENERAL HIERARCHY

Primer upotrebe šablona General Hierarchy prikazuje postavljanje hijerarhije unutar različitih sistema.

Rešenje: Napraviti apstraktnu klasu <<Čvor>> koja predstavlja karakteristike koje sadrži svaki objekat u hijerarhiji, jedna takva karakteristika može biti i da svaki čvor može imati nekog iznad sebe u hijerarhiji. Nakon toga, potrebno je napraviti minimum dve podklase klase <<Čvor>>. Jedna od klasa <<SuperioranČvor>> mora biti povezana asocijacijom "podređen"; sa superklasom dok <<NeSuperiorniČvor>> podklasa ne mora biti povezana.
Asocijacija "podređen" može biti opciono na više ("optional-to-many") ili više na više ("many-to-many"). Ako je asocijacija više na više, u takvoj hijerarhiji čvor može imati mnogo objekata ispod sebe.
Primer: Na slici 1 prikazana su tri tipa zaposlenih u organizaciji ali jedino menadžer može da nadgleda podređene u hijerarhiji. Slično tome, fajl sistem, koji je takođe prikazan na slici 1, može sadrži ostale objekte fajl sistema.



Slika 2.1 Primer upotrebe šablona General Hierarchy [Izvor: Lethbridge [2]]

U korisničkom interfejsu, koji je kao primer prikazan na slici 1, objekti paneli mogu sadržati druge kao što je prikazano na slici 2.

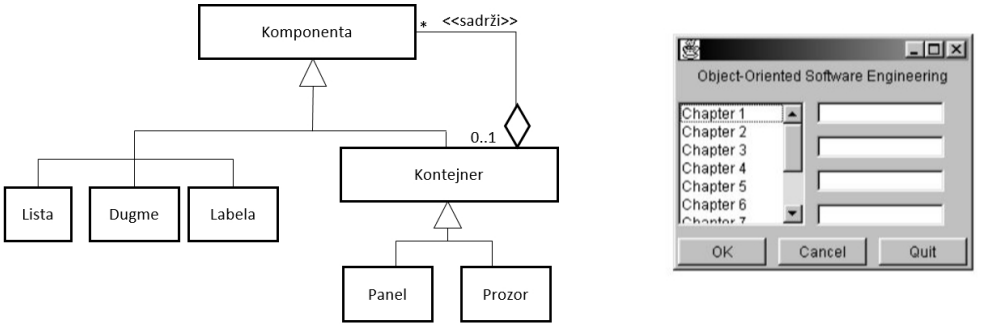
PRIMER UPOTREBE ŠABLONA GENERAL HIERARCHY

Primer pokazuje okvir koji se sastoji od liste, oznake i dva panela.

Prvi panel sa desne strane se sastoji od četiri tekst polja. Drugi panel se sa stoji od tri različita dugmeta.

Anti-šablon: Greška koja se može javiti prilikom upotrebe ovog šablona je upotreba nasleđivanja. Bitno je naglasiti da se ne podrazumeva da svaka hijerarhija mora biti hijerarhija nasleđivanja.

Srodni šabloni: Šablon Composite predstavlja specijalizaciju šablona General Hierarchy. Kod šablona Composite veza između <<SuperioranČvor>> klase i <<Čvor>> klase je agregacija.



Slika 2.2 Upotreba šablona General Hierarchy [Izvor: Lethbridge [2]]

Poglavlje 3

Šablon Player–Role

KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA PLAYER–ROLE

Primena ovog šablona može rešiti problem predstavljanja klasa različitog tipa u okviru klasnog dijagrama.

Kontekst: Uloga ("role") predstavlja skup mogućnosti povezanih sa objektom u određenom kontekstu. Objekat može "igrati" ("play") određene uloge u određenom kontekstu.
Primer može biti, student koji je na Univerzitetu i može biti diplomirani ili ne diplomirani student u određenom trenutku i on može promeniti ulogu iz jedne u drugu. Takođe, student može biti registrovan na kurs u punom vremenu ili u određenom vremenskom intervalu. Student po potrebi može menjati tip vremena koji provodi na kursu.
Problem: Kako najbolje modelovati igrače i uloge tako da igrač može da promeni uloge ili da poseduje više uloga?
Ograničenja: Potrebno je poboljšati enkapsulaciju obuhvatanjem informacija koje se tiču svake odvojene uloge u klasi. Takođe, nije moguće da instanca zameni klasu.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

REŠENJE I PRIMER UPOTREBE ŠABLONA PLAYER-ROLE

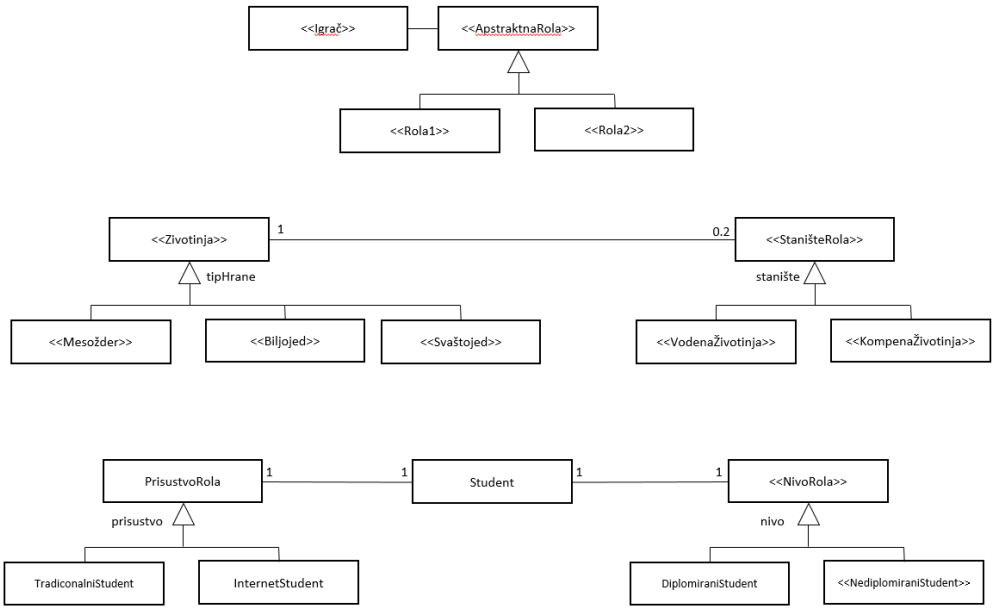
Primenom šablona Player-Role moguće je definisati različite uloge koje objekat određena klasa može izvršavati.

Rešenje: Potrebno je kreirati klasu <<Igrač>> koja prezentuje objekte koji "igraju" ("play") različite uloge. Pored toga, kreirati asocijaciju od ove klase ka apstraktnoj klasi <<ApstraktnaRola>> koja je superklasa seta svih mogućih uloga. Podklasa ove klase <<Rola>> obuhvata sve funkcije povezane sa različitim ulogama. Ako "igrač" može "igrati" samo jednu ulogu u isto vreme, veza između "igrača" i "uloge" može biti jedan na jedan ili će biti jedan na više.
Umesto da bude apstraktna klasa, klasa "Rola" može biti interfejs. Jedini nedostatak ovakvog pristupa je da klasa "Rola" obično sadrži mehanizam povezan sa podklasama koji im omogućava da pristupe informacijama iz klase < >. U tom slučaju potrebno je jedino napraviti < > kao interfejs ukoliko navedeni mehanizam nije potreban. Primeri: Na slici 1 prikazana su tri primera. Primer 1 prikazuje različite životinje koje mogu biti vodene, kopnene ili oboje.

Takođe, moguće je koristiti i klasu "Rola" koja beleži da li je životinja mesojed, biljojed ili svaštojed. Poslednji primer na slici 1 pokazuje da je moguće imati dve odvojene "Rola" superklase.
Student ima status prisustva (puno vreme ili određeno vreme) kao i status da li je diplomirani ili ne diplomirani student. Dve navedene vrste statusa mogu se promeniti tokom trajanja objekta "Student". Šablon Player-Role sa dve asocijacije jedan-na-jedan omogućavaju stvaranje studenta sa punim radnim vremenom, studenta vanrednog studiranja ili bilo koja druga kombinacija koja je na raspolaganju.
Anti-šablon: Jedan način za implementaciju uloga je jednostvno grupisanje svih mogućnosti u jednu <<Igrač>> klasu bez upotrebe klase <<Rola>>. Na ovaj način stvara se prekompleksna klasa i veliki deo orijentacije objekta je izgubljen.Takođe, moguće je kreirati uloge kao podklase klase <<Igrač>> i tada je potrebno primeniti višestruko nasleđivanje ili poslati zahtev instanci da promeni klasu.
Srodni šabloni: Šablon Abstracion-Occurrence ima sličnu strukturu kao Player-Role šablon. "Igrač" ima veliki broj uloga koje su povezane sa njim kao što je to slučaj sa apstrakcijom i pojavama.

PRIMER UPOTREBE ŠABLONA PLAYER-ROLE

Na slici je dat primer upotrebe šablona Player-role.



Slika 3.1 Primer upotrebe šablona projektovanja Player-Role [Izvor: Lethbridge [2]]

Poglavlje 4

Šablon Delegation

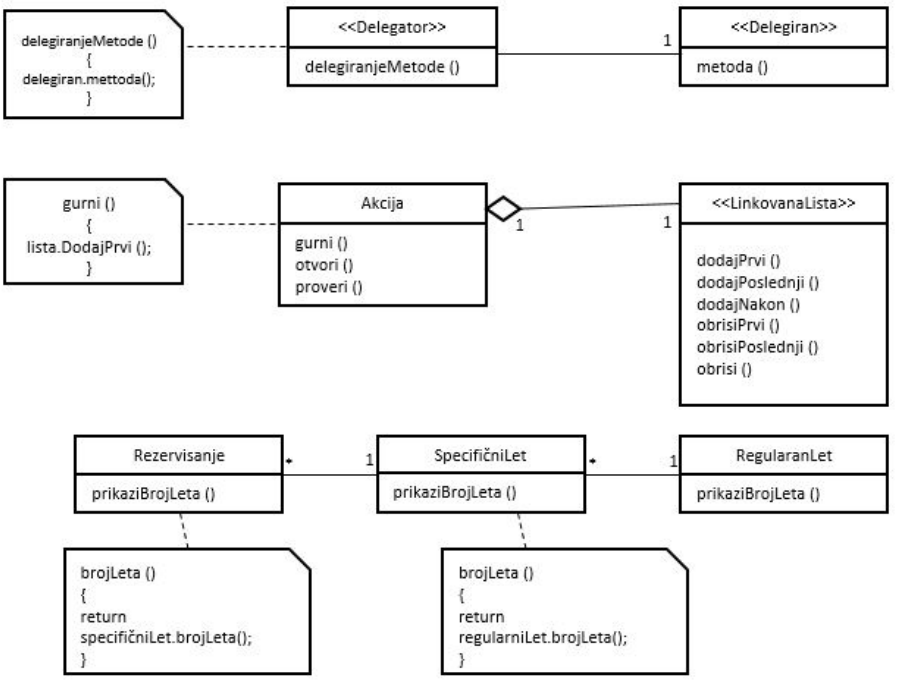
KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA DELEGATION

Šablon Delegation omogućava fleksibilan način projektovanja sistema.

Kontekst: Primena ovog šablona vrši se ukoliko je potrebno iskoristiti operaciju u klasi a ta operacija već postoji implementirana u okviru druge klase. Nije prihvatljivo od klase napraviti podklasu i naslediti operaciju ukoliko se ne koriste sve metode druge klase.
Problem: Kako je moguće najefikasnije iskoristiti metod koji već postoji u drugoj klasi?
Ograničenja: Ponovno korišćenje metoda omogućava minimiziranje razvojnih troškova. Takođe, vrši se smanjivanje veza između klasa i vrši se osiguravanje izvršavanja posla u najprikladnijoj klasi za takav posao.
Rešenje: Potrebno je napraviti metod u klasi <delegator> koja vrši samo jednu stvar: poziva metodu iz susedne klase <delegiran> na taj način omogućavajući ponovnu upotrebu metode za koju je odgovorna klasa <delegator>. Ovaj način primene šablona "Delegation" prikazan je na slici 1. Obično, za delegiranje se koristi asocijacija između "Delegator" i "Delegiraj". Ponekad je moguće napraviti i novu asocijaciju samo da bi se moglo koristiti delegiranje ali pod uslovom da se tom novom asocijacijom ne usložnjava kompletan sistem. Delegacija se može videti kao pružanje selektivnog nasleđivanja.
Primeri: Kao što je prikazano na slici 1, klasu "Akcija" je moguće jednostavno kreirati iz postojeće klase kolekcije kao što je "LinkovanaLista". Metoda "gurni" u okviru klase "Akcija" može jednostavno pozvati "dodajPrvit" metodu iz "LinkovanaLista", "otvori" metoda može pozvati "obrišiPrvi" metodu i "jePrazan" metoda bi delegirala metodu istog imena. Druge metode iz "LinkovanaLista" klase neće biti korišćene jer se ne mogu koristiti u "Akcija" klasi.
Primer u donjem delu slike 1 prikazuje dva nivoa delegiranja u okviru avio sistema. Klasa "Rezervisanje" ima "brojLeta" metodu koja ne radi ništa drugo osim što vrši delegiranje u metodu istog imena u blizini, u klasi "SpecifičniLet". Ovo zauzvrat delegira "RegularanLet".
Umesto da se koristi delegiranje, uobičajeno je da se koristi generalizacija i nasleđivanje metode koja će ponovo biti korišćena.

REŠENJE I PRIMER UPOTREBE ŠABLONA DELEGATION

Prilikom primene šablona Delegation potrebno je paziti na usložnjavanje sistema i modelovanih klasa.



Slika 4.1 Primer upotrebe šablona projektovanja Delegation [Izvor: Lethbridge [2]]

Anti-šablon: Šablon Delegation obuhvata tri principa koja podstiču
fleksibilan način projektovanja:
favorizovanje asocijacije umesto nasleđivanja, kada prednost nasleđivanja nije potrebna
izbegavanje dupliranja programskog koda
pristup informacijama samo u blizini
Srodni šabloni: Šablon Adapter i Proxy šablon koriste delegiranje.

Poglavlje 5

Šablon Immutable

KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA IMMUTABLE

Šablon Immutable omogućava nepromenljiv objekat koji ima stanje koje se nikad ne menja nakon njegovog kreiranja.

Kontekst: Bitan razlog za korišćenje nepromenljivog objekta je što drugi objekti mogu verovati svom sadržaju bez neočekivanih izmena.
Problem: Kako kreirati klasu čije su instance nepromenljive?
Ograničenja: Nepromenljivost mora biti izvršena. Nisu potrebni nikakvi nedostaci koji dozvoljavaju "ilegalnu" modifikaciju nepromenljivog objekta.
Rešenje: Potrebno je proveriti da li je konstruktor nepromenljive klase jedino mesto gde se vrednosti varijabli postavljaju ili modifikuju. Uveriti se da metode bilo koje instance nemaju neželjene efekte promenom varijabli instance ili metoda pozivanja. Ako metoda koja bi inače izmenila varijablu instance mora biti prisutna, onda ona mora vratiti novu instancu klase.
Primer: U klasi koju nije moguće izmeniti "Point" vrednosti "x"i "y" biće postavljene od strane konstruktora i nikada neće biti modifikovane. Ako je dozvoljena operacija "translate" ka "Point", nova instanca može biti kreirana. Objekat koji zahteva "translate" operaciju treba da koristi novu prevedenu tačku ("translated point").
Srodni šabloni: Šablon "Read-only Interface" omogućava iste mogućnosti kao i šablon Immutable.

Poglavlje 6

Šablon Read-Only Interface

KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA READ-ONLY INTERFACE

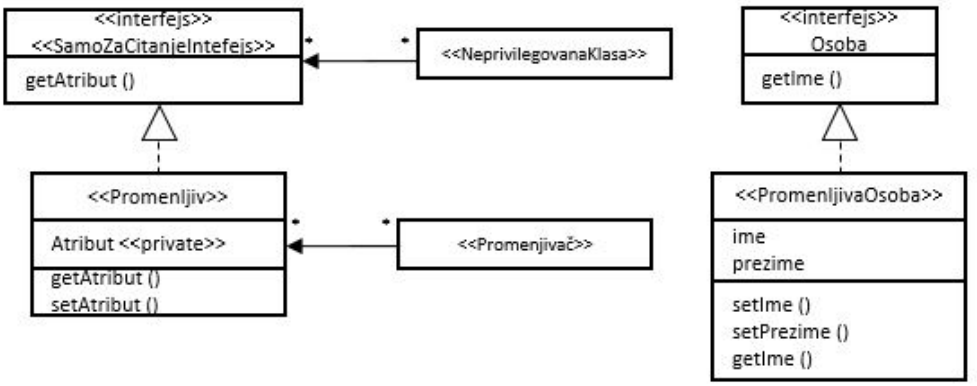
Primenom šablona Read-Only Interface moguće je definisati klase sistema koje mogu da modifikuju atribute objekata koji su nepromenljivi.

Kontekst: Ovaj šablon je usko povezan sa šablonom Immutable. Ukoliko je ponekad potrebno kreirati određene privilegovane klase koje će moći da modifikuju atribute objekata su inače nepromenljivi.
Problem: Kako kreirati situaciju gde određene klase vide klasu koju je moguće samo čitati ("read only") dok druge klase mogu da izvrše određenu modifikaciju.
Ograničenja: Programski jezici kao što je Java omogućavaju kontrolu pristupa koristeći javne, zaštićene i privatne ključne reči. Omogućavanje javnog pristupa dozvoljava javno čitanje i pisanje.

REŠENJE I PRIMER UPOTREBE ŠABLONA READ-ONLY INTERFACE

Šablon Read-Only Interface može biti korišćen da šalje podatke ka objektima u grafičkom korisničkom interfejsu.

Rešenje: Kreirati klasu <<Promenljiv>> kako bi se kreirala bilo koja druga klasa. Proslediti instance ove klase metodama koje treba da izvrše izmene. Nakon toga, kreirati javni interfejs koji će biti nazvan <<SamoZaČitanjeInterfejs>> koji ima jedino operaciju čitanja ka <<Promenljiv>> koja je jedina operacija koja uzima njegove vrednosti. Moguće je proslediti instance od <<SamoZaČitanjeInterfejs>> ka metodama koje ne treba da izvršavaju izmene a takođe čuvaju objekte od neočekivanih izmena. Klasa <<Promenljiv>> implementira <<SamoZaČitanjeInterfejs>>. Ovaj primer prikazan je na slici 1.
Primer: Slika 1 prikazuje interfejs "Osoba" koji može biti korišćen od različitih delova sistema koji nemaju prava da modifikuju podatke. Klasa "PromenljivaOsoba"postoji u paketu koji štiti od neautorizovane modifikacije.
Ovaj šablon omogućava da se ne vrše neautorizovane izmene podataka.



Slika 6.1 Primer upotrebe šablona Read-only Interface [Izvor: Lethbridge [2]]

Anti-šablon: Potrebno je napraviti klasu koja samo može biti pročitana kao podklasa klase <mutable> klase.

Poglavlje 7

Šablon Proxy

KONTEKST, PROBLEM I OGRANIČENJA ŠABLONA PROXY

Šablon se predstavlja klasnim dijagramom i služi za prikazivanja aspekata primene softverske arhitekture.

Kontekst: Često je potrebno dosta vremena kako bi se pristupilo instanci klase. Takve klase nazivaju se klase teške kategorije ("heavyweight classes"). Instance od klasa teške kategorije mogu uvek biti u bazi podataka. Da bi se instance koristile u programu potrebno je da ih konstruktor učita sa podacima iz baze podataka. Slično tome objekti teške kategorije mogu postojati samo na severu, pre korišćenja objekta klijent treba da pošalje zahtev a nakon toga sačeka da objekat stigne. U obe situacije postoji vremensko kašnjenje i kompleksan mehanizam koji uključuje stvaranje objekata u memoriji. Mnogi drugi objekti u sistemu možda imaju potrebu da pozovu ili koriste instance klasa teške kategorije.
Uobičajeno je da sve klase domena budu klase teške kategorije.
Problem: Na koji način je moguće smanjiti potrebu učitavanja memorije velikog broja objekata teške kategorije iz baze podataka ili servera ukoliko oni ne bi bili potrebni? Još jedan problem koji se može javiti je: Ukoliko se učita jedan objekat iz baze podataka ili sa servera, na koji način je moguće izbeći učitavanje drugih objekata koji su povezani sa njim?
Ograničenja: Potrebno je da svi objekti u domenskom modelu budu dostupni za programe koji će se koristiti za izvršavanje različitih odgovornosti sistema. Takođe je važno da objekti ostaju i pokreću se u okviru istog programa. U velikom sistemu bilo bi nepraktično da svi objekti budu stavljeni u memoriju kada god se program pokrene. Veličina memorije je takođe ograničena, potrebno je puno vremena za učitavanje baze podataka u memoriju i mali broj objekata u samoj bazi podataka koji će biti potreban za bilo koji određeni program. Čuvanje ovih objekata u memoriji takođe bi otežavalo više programa da dele iste objekte.

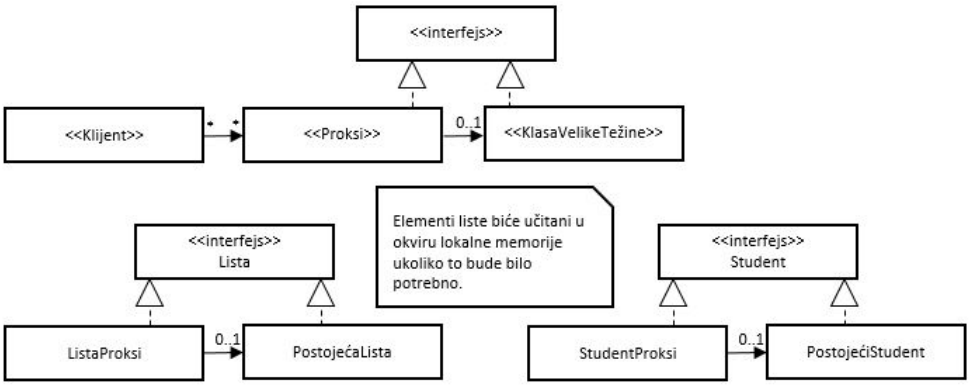
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

REŠENJE I PRIMER UPOTREBE ŠABLONA PROXY

Šablon Proxy je korišćen u mnogim softverskim arhitekturama.

Rešenje: Potrebno je stvoriti jednostavnu verziju klase <<KlasaVelikeTežine>>. Tako kreirana klasa biće nazvana <<Proksi>>. Takva klasa ima isti interfejs kao i <<KlasaVelikeTežine>> pa programeri koji rade na implementaciji mogu deklarisati varijable

bez brige da li će klasa <<Proksi>> ili njegova <<KlasaVelikeTežine>> verzija biti stavljena u varijablu. Primer je na slici 1.



Slika 7.1 Primer upotrebe šablona Proxy [Izvor: Lethbridge [2]]

Objekat <<Proksi>> obično se ponaša kao čuvar. Ukoliko neko pokuša da izvrši operaciju automatski se delegira operacija ka <<KlasaVelikeTežine>> klasi. Kada je potrebno <<Proksi>> preuzima velike zadatke od strane <<KlasaVelikeTežine>> objekta.
Primer: Na slici 1 prikazan je primer upotrebe šablona Proksi. Varijabla sadrži listu i takva varijabla sadrži ListaProksi jer bi bilo jako kompleksno uvlačiti kompletnu listu objekata u memoriju i takva lista često neće biti potrebna. Kada operacija pristupi listi ListaProksi može kreirati instancu od PostojećaLista u memoriji. U drugu ruku, ListaProksi može biti sposobna da odgovori na određene upite kao što su broj elemata u listi bez pozivanja PostojećaListat.
Ukoliko je recimo PostojećaLista lista studenata, takvi objekti mogu biti "proksi" u tom slučaju instance od StudentProksi. Takođe, instance od PostojećiStudent mogu biti pokretane samo u slučaju potrebe.
Anti-šablon: Strategija koja se koristi u malim sistemima omogućava da se cela baza podataka učita u memoriju kada se program pokrene.
Srodni šabloni: Šablon Delegation je sličan šablonu Proksi i može se koristiti zajedno sa ovim šablonom.

PROXY PATTERN (VIDEO)

Trajanje: 8:12 minuta.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 8

Objektni klijent-server okvir u projektovanju

FABRIKA KLIJENT KONEKCIJE

Objektni klijent-server okvir (OCSF) omogućava jednostavan način za brzo postavljanje klijent-server aplikacije. Primenom šablona poboljšava se fleksibilnost sistema.

Prvo proširenje osnovnog okvira je dodavanje Fabrike za obradu klijentskih veza. U svrhu razumevanja korisnosti ovog mehanizma potrebno je prvo pregledati upravljanje klijentskim vezama na serveru. Svaki put kada se novi klijent povezuje sa serverom kreira se "KonekcijaNaKlijenta" objekat. Ovaj objekat definiše nit koja upravlja svim komunikacijama sa tim klijentom. Sve poruke primljene od klijenta prenose se na metod "obradiPorukuOdKlijenta" u podklasu "AbstraktniServer". Ovaj metod je sinhronizovan tako da su dve "KonekcijaNaKlijenta" niti potrebne za pristup istom resursu.
Ukoliko je potrebno, rukovanje klijentima može biti održano u verziji "obradiPorukuOdKlijenta" kroz posebnu podklasu "KonekcijaNaKlijenta". Procesiranje poruka sa različitih klijenata može se vršiti istovremeno ukoliko se izvrši pravilna sinhronizacija.
Da bi se omogućilo da klasa servera vrši instanciranje podklasa specifičnih za aplikaciju "KonekcijaNaKlijenta", OCSF nudi opcioni mehanizam fabrike. To su dva ključa. Prvi ključ je interfejs pod nazivom "ApstraktnaKonekcijaFabrike" (prikazan na slici 1). Potrebno je kreirati klasu fabrike koja implementira "createKonekcija" metodu u ovom interfejsu. Takva fabrička klasa će kreirati instacu od sopstvene podklase od "KonekcijaNaKlijenta".
Drugi ključ je metoda "setConnectionFactory" koja se nalazi u "ApstraktniServer" klasi.
Da bi se koristio fabrički mehanizam OCSF-a potrebno je uraditi sledeće:
1. **Kreirati podklasu iz "KonekcijaNaKlijenta".** Njegov konstruktor mora imati isti potpis kao "KonekcijaNaKlijenta" i on mora pozvati konstruktora "KonekcijaNaKlijenta" koristeći super ključnu reč.
2. **Kreirati klasu fabrike koja jednostavno definiše metod** "createKonekcija" funkciju interfejsa "ApstraktnaKonekcijaFabrike". Metoda bi izgledala ovako:

```
protected ConnectionToClient createConnection(
ThreadGroup group, Socket clientSocket,
AbstractServer server) throws IOException
{
return new Connection(group,clientSocket,server);
```

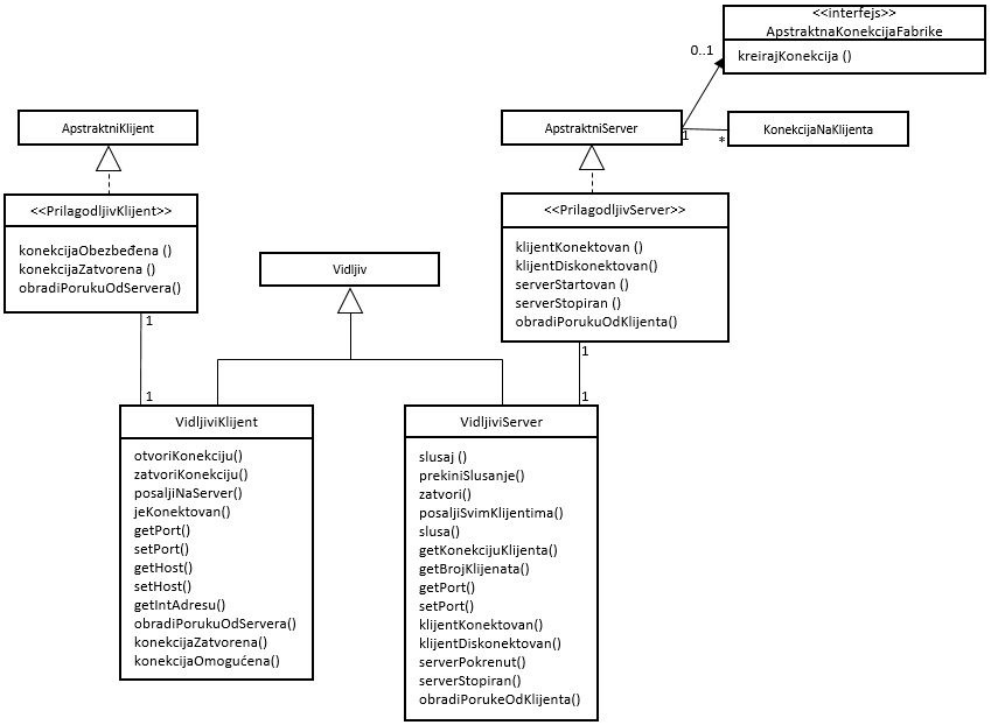
3. **Urediti server tako da napravi poziv pre početka slušanja**.

```
setConnectionFactory(new MyConnectionFactory());
```

PRIMETAN SLOJ

Drugo proširenje OCSF okvira je dodavanje primetnog sloja.

Poruku koju primi klijent obrađuje podklasa "ApstraktniKlijent" koju implementira metoda "obradiPorukuOdServera". Svaki put kada se razvije nova aplikacija, klasa "ApstraktniKlijent" mora biti podklasa.
Observer šablon pruža alternativni mehanizam za razvoj klijenta. Bilo koji broj <<Observer>> klasa može tražiti da bude obavešten u slučaju kada se nešto "interesantno" dešava na klijentu kao na primer dolazak poruke za zatvaranje konekcije. U tu svrhu, potrebno je imati podklasu "ApstraktniKlijent" koja je <<Observable>>. Na slici 1 korišćen je Adapter šablon. Prošireni OCSF ima klase "VidljiviKlijent". To je isti interfejs kao "ApstraktniKlijent" osim što postoji podklasa "Vidljiv". Takođe je i Adapter: delegira metode kao što su "pošaljiNaServer", "setPort" itd. ka instancama konkretne podklase od "ApstraktniKlijent" koja je nazvana "PrilagodljiviKlijent". Projektanti korišćenjem "VidljiviKlijent" klase nikada ne moraju da znaju da postoji klasa "PrilagodljiviKlijent".



Slika 8.1 Primer OCSF sa ekstenzijama za primenu šablona Observable i Factory [Izvor: Lethbridge [2]]

IMPLEMENTACIJA OBSERVABLE SLOJA

Dati su primeri implementacije serverske strane Observable sloja.

Klasa "PrilagodljivKlijent" koja je konkretna podklasa "ApstraktniKlijent" klase omogućava konkretnu implementaciju od obradiPorukuOdServera. Takođe, omogućava implementaciju metoda connectionClosed i "konekcijaOmogućena". Tri navedena načina povratnog poziva se prenose na "VidljiviKlijent"t. Strukura je sledeća:

```
callbackMethod()
{
observable.callbackMethod();
}
```

Tu je uvek i jedan na jedan veza između "PrilagodljivKlijent" i "VidljivKlijent" klase. Instance od ove dve klase moraju postojati.

Svi servisi metoda u "PrilagodljivKlijent"t (kao što je otvoriKonekciju) jednostavno su delegirani ka "PrilagodljivKlijent" Imaju sledeću strukturu:

```
serviceMethod()
{
return adaptable.serviceMethod();
}
```

Metoda obradiPorukuOdServera u klasi "VidljivKlijent" je implementirana kao:

```
public void handleMessageFromServer(Object message)
{
setChanged();
notifyObservers(message);
}
```

Druge metode povratnog poziva u "VidljivKlijent" klasi, kao što je metoda konekcijaZatvorena, ne rade ništa. Projektant treba da kreira podklasu klase "VidljivKlijent" koja može implementirati konekcijaZatvorena kao:

```
public void connectionClosed()
{
setChanged(); notifyObservers("connectionClosed");
}
```

Serverska strana se implementira analogno osim što se instanca "KonekcijaNaKlijenta" može poslati posmatračima ("observerima"). Prednosti korišćenja sloja OCSF-a su:
1. Različite vrste poruka mogu se obraditi kroz različite klase posmatrača ("observera"). Različiti delovi korisničkog interfejsa mogu se sami ažurirati kada se primaju određene poruke.
2. Programeri koji koriste "VidljivKlijent" ili "VidljivServer" ne moraju imati detaljnu specifikaciju ovih klasa, već samo informaciju da postoje.

KORIŠĆENJE OBSERVABLE SLOJA

Korišćenjem Observable sloja moguće je izvršiti povezivanje sa drugim klasama sistema.

U slučaju da je potrebno povezati klasu sa observable slojem OCSF-a procedura je sledeća:
1. Kreirati klasu koja implementira Observer interfejs (potrebno je naglasiti da ovo nije

podklasa od bilo koje klase okvira)
2. Registrovati instancu nove klase uz pomoć konstruktora:

```
public MessageHandler(Observable client)
{
client.addObserver(this);
...
}
```

3. Definisati metodu za ažuriranje (update) u novoj klasi. Potrebno je omogućiti da klasa reaguje jedino na poruke određenog tipa .U sledećem primeru klasa je zainteresovana samo za poruke koje su iz klase SomeClass.

```
public void update(Observable obs, Object message)
{ if (message instanceOf SomeClass)
{
// process the message
}
}
```

Ukoliko je poruka String, može se dodati uslov u if bloku šta raditi sa porukom.

POTEŠKOĆE I RIZICI PRILIKOM PRIMENE ŠABLONA PROJEKTOVANJA

Potrebno je proceniti da li upotreba šablona može unaprediti sistem u okviru koga se primenjuje.

Primer može biti: Nije uvek potrebno primeniti šablon Facade u svakom podsistemu, dodavanjem dodatne klase moguće je uslozniti sistem a takođe i instance klasa u podsistemu mogu biti opterećene podacima.
Uvek je potrebno razumeti prednosti šablona i pažljivo donositi odluke o projektovanju softvera primenom šablona.
Razvijanje šablona je teško. Pisanje dobrih šablona zahteva značajan i dugotrajan rad. Često se primenom šablona mogu doneti nepravilne odluke koje mogu proces projektovanja odvesti u pogrešnom smeru.
Nije potrebno pisati šablone za druge projektante softvera, potrebno je pisati šablon tako da odgovara upotrebi u konkretnom sistemu koji se razvija. Takođe, svaki šablon je potrebno detaljno dokumentovati i specificirati kako bi se održao u kasnijim fazama razvoja softvera.

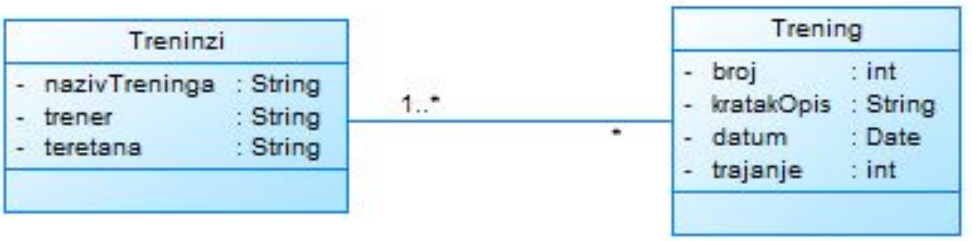
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 9

Pokazna vežba-Upotreba šablona projektovanja

PRIMER UPOTREBE ŠABLONA ABSTRACTION-OCCURRENCE

Dat je primer upotrebe šablona Abstracion-Occurrence.



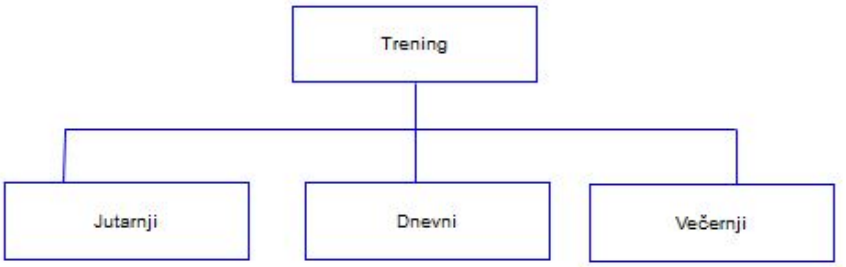
Slika 9.1 Primer upotrebe šablona Abstraction-Occurrence [Izvor: Nebojša Gavrliović]

Na slici 1 dat je primer upotrebe šablona Abstraction-Occurrence. Primer obuhvata klasu "Treninzi" koja ima atribute: NazivTreninga, Trener i Teretana i klasu "Trening" koja ima atribute: Broj, KratakOpis, Datum i Trajanje. Navedene klase dele informacije između sebe bez dupliranja zajedničkih informacija. (10 min)

Preporuka je da se za predstavljanje šablona Abstraction-Occurrence koristi klasni dijagram ili free model(preporučeni alati PowerDesigner ili draw.io).

PRIMER UPOTREBE ŠABLONA GENERAL HIERARCHY

Dat je primer upotrebe šablona General Hierarchy.



Slika 9.2 Primer upotrebe šablona General Hierarchy [Izvor: Nebojša Gavrliović]

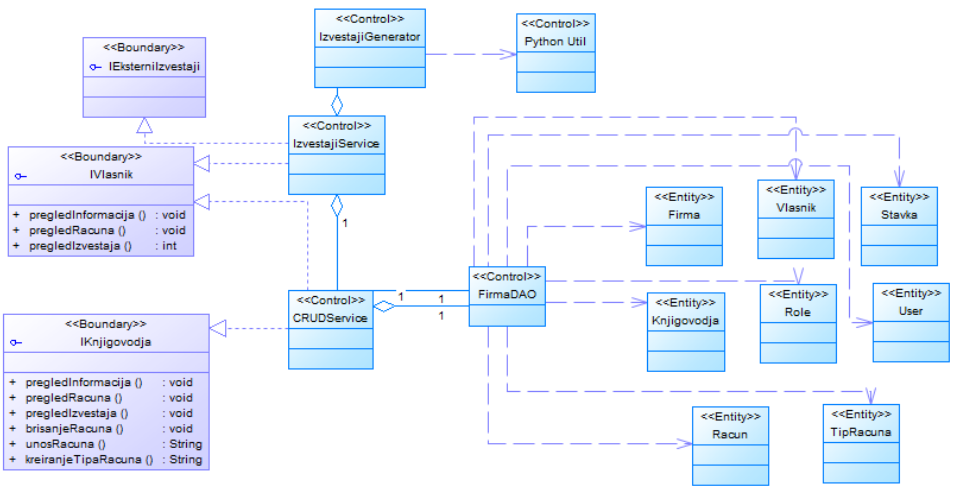
Na slici 2 dat je primer upotrebe šablona General Hierarchy. Predstavljena su tri tipa treninga: jutarnji, dnevni i večernji. Objekat "Trening" u predstavljenoj hijerarhiji ima ispod sebe objekte: "Jutarnji", "Dnevni" i "Večernji". Podaci u objektima: "Jutarnji", "Dnevni" i "Večernji" se razlikuju shodno različitim tipovima treninga a osnovne informacije o treningu dobijene su iz objekta "Trening".(10 min)

Preporuka je da se za predstavljanje šablona General Hierarchy koristi klasni dijagram ili free model(preporučeni alati PowerDesigner ili draw.io).

PRIMER UPOTREBE ŠABLONA GENERAL HIERARCHY - SOFTVER ZA VOĐENJE KNJIGOVODSTVA

Primer prikazuje upotrebu šablona General Herarchy.

Na slici je prikazan primer upotrebe šablona General Hierarchy u okviru softvera za vođenje knjigovodstva. (15 min)



Slika 9.3 Šablon General Hierarchy primenjen na softveru za vođenje knjigovodstva [Izvor: Nebojša Gavrliović]

PRIMER UPOTREBE ŠABLONA PLAYER–ROLE

Dat je primer upotrebe šablona Player–Role.

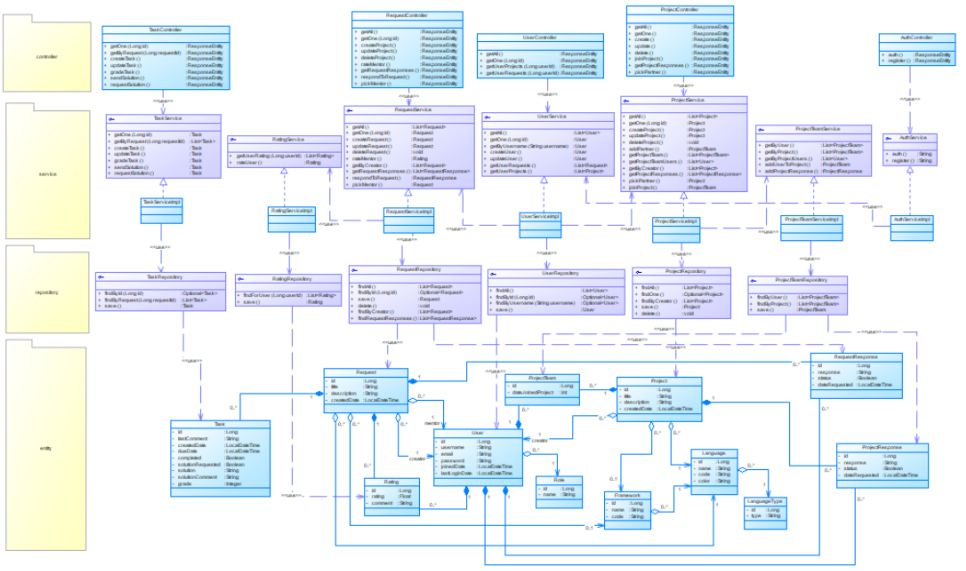
Slika 9.4 Primer upotrebe šablona Player–Role [Izvor: Nebojša Gavrliović]

Primer upotrebe šablona Player-Role prikazan je na slici 4. Prikazane su dve različite uloge (role) klijenta. Klijent može biti "Rekreativac", "Amater" ili "Profesionalni sportista" a u isto vreme može biti svakodnevni član koji svakog dana dolazi na trening ili periodični član koji u određenim vremenskim intervalima dolazi na treninge. (10 min)

Preporuka je da se za predstavljanje šablona Player-Role koristi klasni dijagram ili free model (preporučeni alati PowerDesigner ili draw.io).

PRIMER SISTEMA ZA MENTORSTVO - KLASNI DIJAGRAM

Dat je primer klasnog dijagrama sistema za mentorstvo. Preuzeti dijagram u aktivnosti nakon objekta učenja vežbi i identifikovati primenu šablona projektovanja.



Slika 9.5 Klasni dijagram sistema za mentorstvo [Izvor: Nebojša Gavrliović]

Poglavlje 10

Individualna vežba - Zadaci

ZADATAK ZA SAMOSTALNI RAD

Na osnovu definisanih korisničkih zahteva modelovati klasni dijagram i prikazati upotrebu različitih šablona.

Softver je namenjen za zaposlene u medicinskoj ustanovi (medicinska sestra, doktor, direktor klinike).
Svaki korisnik se prijavljuje i na osnovu prijave ima različite opcije korišćenja sistema. Opcije su sledeće:

- Prijavljivanje na sistem klinike
- Otvaranje kartona pacijenta
- Pristup kartonu pacijenta
- Slanje pacijentovih podataka iz kartona lekaru
- Prepisivanje terapije
- Izdavanje recepta za leka
- Davanje uputa za dalje lečenje
- Pretraživanje zaposlenih
- Potpisivanje naloga, zapisnika i pravilnika klinike
- Prikaz pravilnika klinike

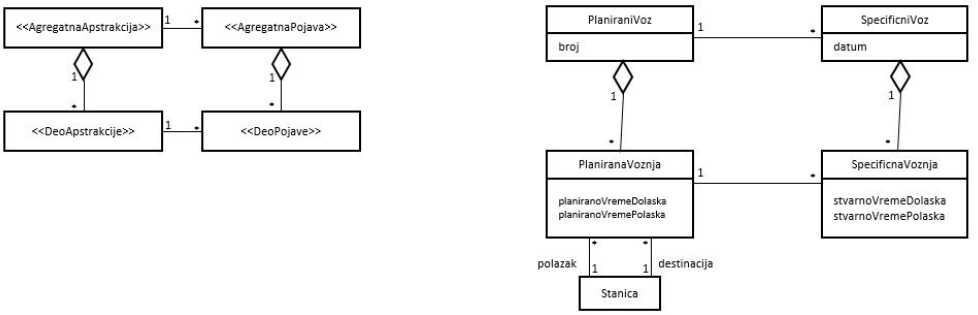
Korišćenjem opisanog scenarija potrebno je:

- Prikazati upotrebu šablona Abstraction-Occurrence (25 min)
- Prikazati upotrebu šablona General Hierarchy (25 min)
- Prikazati upotrebu šablona Player-Role (25 min)
- Prikazati upotrebu šablona Delegation (25 min)

ZADACI ZA SAMOSTALNI RAD - ABSTRACTION-OCCURRENCE

Na osnovu prethodno datih primera primene šablona Abstraction-Occurrence potrebno je uraditi zadatak.

Primeniti šablon Apstrakcija-Pojava ("Abstraction-Occurrence") u sledećim situacijama. Za svaku situaciju pokazati dve povezne klase, vezu između klasa i atribute u svakoj klasi. (10min)



Slika 10.1 Zadatak za samostalni rad šIzvor: Lethbridge [2]đ

ZADACI ZA SAMOSTALNI RAD - GENERAL HIERARCHY

Na osnovu prethodno datih primera primene šablona Abstraction-Occurrence potrebno je uraditi zadatke.

Zadatak 1: Primeniti šablon General Hierarchy na softverskoj arhitekturi proizvoljnog sistema. Modelovati odgovarajući dijagram klasa i pridržavati se definisanog načina predstavljanja šablona General Hierarchy. (10min)

Zadatak 2: Detaljno opisati identifikovane klase u primenjenom šablonu. (10min)

ZADACI ZA SAMOSTALNI RAD - PLAYER-ROLE

Na osnovu prethodno datih primera primene šablona Player-Role potrebno je uraditi zadatak.

Zadatak 1: Primeniti šablon Player-Role na softverskoj arhitekturi proizvoljnog sistema. Modelovati odgovarajući dijagram klasa i pridržavati se definisanog načina predstavljanja šablona Player-Role. (20min)

Zadatak 2: Detaljno opisati identifikovane klase u primenjenom šablonu. (10min)

ZADACI ZA SAMOSTALNI RAD - DELEGATION

Na osnovu prethodno datih primera primene šablona Delegation potrebno je uraditi zadatak.

Zadatak 1: Primeniti šablon Delegation na softverskoj arhitekturi proizvoljnog sistema. Modelovati odgovarajući dijagram klasa i pridržavati se definisanog načina predstavljanja šablona Delegation. (20min)

Zadatak 2: Detaljno opisati identifikovane klase u primenjenom šablonu. (10min)

ZADACI ZA SAMOSTALNI RAD - READ-ONLY INTERFACE

Na osnovu prethodno datih primera primene šablona Read-Only potrebno je uraditi zadatak.

Zadatak 1: Primeniti šablon Read-Only na softverskoj arhitekturi proizvoljnog sistema. Modelovati odgovarajući dijagram klasa i pridržavati se definisanog načina predstavljanja šablona Read-Only. (30min)

Zadatak 2: Detaljno opisati identifikovane klase u primenjenom šablonu. (10min)

ZADACI ZA SAMOSTALNI RAD - PROXY

Na osnovu prethodno datih primera primene šablona Proxy potrebno je uraditi zadatak.

Zadatak 1: Primeniti šablon Proxy na softverskoj arhitekturi proizvoljnog sistema. Modelovati odgovarajući dijagram klasa i pridržavati se definisanog načina predstavljanja šablona Proxy. (20min)

Zadatak 2: Detaljno opisati identifikovane klase u primenjenom šablonu. (10min)

ZADACI ZA DISKUSIJU NA VEŽBI

Otvorite diskusiju po svakom od dole postavljenih pitanja.

1. Na koji način nepravilna upotreba šablona projektovanja može da utiče na projektovanje i razvoj softvera?(10min)

2. Na koji način projektant softvera može da predvidi rizike u toku primene određenog šablona projektovanja?(10min)

3. Šta predstavljaju srodni šabloni određenom šablonu projektovanja? Na koji način ih je moguće kombinovati?(10min)

4. Šta su šabloni modelovanja a šta šabloni projektovanja softvera? Objasniti razliku.(10min)

5. Da li je u svakoj situaciji korisno koristiti šablone projektovanja? Dati obrazloženje odgovora.(10min)

Poglavlje 11

Domaći zadatak br.6

PRAVILA ZA IZRADU DOMAĆEG ZADATKA

Student zadatake treba samostalno da uradi i da dostavi asistentu.

Domaći zadatak br.6: Prikazati upotrebu šablona Abstraction-Occurrence unutar sistema. Upotrebu šablona predstaviti klasnim dijagramom.
Pri radu, koristite Power Designer.

Domaći zadaci su dobijaju se po definisanim instrukcijama. Domaći zadatak se imenuje: SE311-DZ06-ImePrezime-brIndeksa gde su vrednosti Ime, Prezime i br.Indeksa vaši podaci. Domaći zadatak je potrebno poslati na adresu asistenta:

nebojsa.gavrilovic@metropolitan.ac.rs (tradicionalni studenti iz Beograda i internet studenti)
jovana.jovic@metropolitan.ac.rs (tradicionalni studenti iz Niša)

sa naslovom (subject mail-a) SE311-DZ06. Potrebno je poslati modelovane dijagrame i dokument sa opisom dijagrama ili odgovorima na pitanja.

Napomena: Domaći zadaci treba da budu realizovani u zadatku navedenom razvojnom okruženju i da predstavljaju jedinstveno rešenje svakog studenta. Prepisivanje i preuzimanje rešenja sa interneta ili od drugih studenata strogo je zabranjeno.

Poglavlje 12

Zaključak

ZAKLJUČAK

- Primenom šablona projektovanja omogućeno je kreiranje preciznijih i detaljnih modela klasa
- Šabloni projektovanja pomažu u izbegavanju grešaka i čine sistem fleksibilnim i jednostavnijim
- Delegation šablon projektovanja sprečava prekomerno međusobno povezivanje različitih delova sistema
- Abstraction Occurrence i Player Role šabloni projektovanja omogućavaju razdvajanja delova sistema
- Proxy šablon projektovanja omogućavaju programeru da u procesu implementacije koristi objekte drugih klasa
- Read only šablon projektovanja pomaže u zaštiti objekata od nepredviđenih promena

LITERATURA

U ovoj lekciji korišćena je kao obavezna literatura, referenca 2, poglavlje 6 (2. izdanje).

Obavezna literatura:

1. Onlajn nastavni materijal na predmetu SE311 Projektovanje i arhitektura softvera, , školska 2018/19, Univerzitet Metropolitan
2. T.C.Lethbrigde, R. Lagariere - Object-Oriented Software Engineering - Practical Software Development using UML and Java - 2005, poglavlje 6 (strane od 221-252)

Dopunska literatura:

1. D. Budgen, Software Design, 2nd ed.,Addison-Wesley, 2003.
2. P. Clements et ad. , Documenting Software Architectures: Views and Beyond, 2nd ed., Pearson Education
3. Ian Sommervile, Software Engineering, Tenth Edition, Pearson Education Inc.,2016. ili 9th Edition, 2011
4. P.B. Kruchten, “The 4+1 View Model of Architecture,” IEEE Software, vol. 12, no. 6, 1995, pp. 42–55.
5. E. Gamma et al., Design Patterns:Elements of Reusable Object-Oriented Software, 1st ed., Addison-Wesley Professional, 1994.
6. J. Nielsen, Usability Engineering, Morgan Kaufmann, 1993.
7. Service-oriened Architecture, Concept, Technology, and Design, autora T. Erl u izdanju Pretince Hall, 2005, ISBN 0-13-185858-0.

8. P. Stevens, Using UML – Software Engineering with Objects and Components, Second Edition, Assison-Wesley, Pearson Eduction, 2006
9. R. Pressman, Software Engineering – A Practioner's Approach, Seventh Edition, McGraw Hill Higher Education, 2010

Veb lokacije :

- http://www.software-engin.com
- http://www.uml.org/