Sadržaj

Predgovor autora

Ova knjiga nije uvod u tehnologiju objektno orijentisanog projektovanja. Za to već postoje mnoge dobre knjige. U ovoj knjizi se polazi od pretpostavke da ste prilično iskusni bar u jednom objektno orijentisanom programskom jeziku i da imate neko iskustvo u objektno orijentisanom projektovanju. Nikako ne bi trebalo da uzimate rečnik čim se pomenu „tipovi" i „polimorfizam" ili nasleđivanje „interfejsa" umesto nasleđivanja „implementacije".

S druge strane, ovo nije ni napredan tehnički traktat. Ovo je knjiga o **uzorcima za projektovanje** u kojoj se opisuju jednostavna i elegantna rešenja specifičnih problema u projektovanju objektno orijentisanog softvera. Uzorci za projektovanje sadrže rešenja koja su se vremenom razvijala i proširivala. Prema tome, to nisu projekti koji se obično prave od prve . Oni odražavaju neizreciva doterivanja projekata i programa nastala dok su se razvijaoci borili da postignu veću fleksibilnost i bolje mogućnosti višekratne upotrebljivosti softvera. Uzorci za projektovanje obuhvataju ta rešenja u sažetom i lako primenjivom obliku.

Uzorcima za projektovanje nisu potrebne neobične mogućnosti jezika niti fantastični programerski trikovi kojima zapanjujete prijatelje i rukovodioce. Sve se može primeniti u standardnim objektno orijentisanim jezicima, ali zahtevaju malo više truda od *ad hok* rešenja. Međutim, taj dodatni trud se svaki put isplati zbog veće fleksibilnosti i mogućnosti ponovne upotrebe.

Kada jednom shvatite uzorke za projektovanje i doživite sa njima neko iskustvo tipa „Aha!" (a ne samo „A?"), nikada više nećete na isti način misliti o objektno orijentisanom projektovanju. Usvojićete znanje uz čiju će pomoć vaši vlastiti projekti postati fleksibilniji, modularni, više puta upotrebljivi i razumljivi – zar se niste zbog toga i opredelili za objektno orijentisanu tehnologiju?

Malo upozorenje i ohrabrenje: nemojte da brinete ako prilikom prvog čitanja ne shvatite knjigu u potpunosti. Ni mi nismo sve shvatili prilikom prvog pisanja! Imajte na umu da ovo nije knjiga koja se jednom pročita i ostavi na policu. Nadamo se da ćete joj se stalno vraćati po primere dizajna i po inspiraciju.

Ova knjiga je dugo nastajala. Videla je četiri zemlje, tri ženidbe njenih autora i rođenje dvoje dece (koja nisu u rodu). Mnogo je ljudi učestvovalo u njenom razvijanju. Posebnu zahvalnost dugujemo Bruceu Andersonu, Kentu Becku i Andrëu Weinandu za njihove ideje i savete. Takođe zahvaljujemo onima koji su pregledali radne verzije rukopisa: Roger Bielefeld, Grady Booch, Tom Cargill, Marshall Cline, Ralph Hyre, Brian

Kernighan, Thomas Laliberty, Mark Loreny, Arthur Riel, Doug Schmidt, Clovis Tondo, Steve Vinoski i Rebecca Wirfs-Brock. Zahvalni smo i timu u izdavačkoj kući Addison-Wesley na pomoći i strpljenju: Kate Habib, Tiffany Moore, Lisa Raffaele, Pradeepa Siva i John Wait. Posebno hvala Carlu Kessleru, Dannyu Sabbahu i Marku Wegmanu iz IBM Researcha na njihovoj istrajnoj podršci ovom radu.

Na kraju, ali ne i manje važno, zahvaljujemo svima na Internetu i još dalje koji su komentarisali verzije uzoraka, nudili reči podrške i ubeđivali nas da je to što radimo vredno truda. Tu spadaju, između ostalih, Jon Avotins, Steve Berczuk, Julian Berdych, Matthias Bohlen, John Brant, Allan Clarke, Paul Chisholm, Jens Coldewey, Dave Collins, Jim Coplien, Don Dwiggins, Gabriele Elia, Doug Felt, Brian Foote, Denis Fortin, Ward Harold, Hermann Hueni, Nayeem Islam, Bikramjit Kalra, Paul Keefer, Thomas Kofler, Doug Lea, Dan LaLiberte, James Long, Ann Luise Luu, Pundi Madhavan, Brian Marick, Robert Martin, Dave McComb, Carl McConnell, Christine Mingins, Hanspeter Mössenböck, Eric Newton, Marianne Ozkan, Roxan Payette, Larry Podmolik, George Radin, Sita Ramakrishnan, Russ Ramirez, Alexander Ran, Dirk Riehle, Bryan Rosenburg, Aamod Sane, Duri Schmidt, Robert Seidl, Xin Shu i Bill Walker.

Mi ne smatramo da je ova kolekcija uzoraka potpuna i statična; to je pre snimak naših trenutnih razmišljanja o projektovanju. Komentari su dobrodošli, bilo da je reč o kritici naših primera, referisanju i poznatim primerima upotrebe koji su nam promakli ili uzorcima za projektovanje koje je trebalo uključiti. Možete da nam pišete na Addison-Wesley ili šaljete elektronsku poštu na adresu `design-patterns@cs.uiuc.edu`. Sada postoji i Web stranica na adresi `http://st-www.cs.uiuc.edu/users/patterns/DPBook/DPBook.html`

za naknadne informacije i ažuriranja.

Mountain View, California E.G

Montreal, Quebec R.H.

Urbana, Illinois R.J.

Hawthorne, New York J.V.

August 1994

Predgovor izdavača

Sve dobro strukturirane objektno orijentisane arhitekture su pune uzoraka. Zaista, jedan od načina na koje merim kvalitet objektno orijentisanog sistema jeste procena da li su razvijaoci obratili dovoljno pažnje uobičajenoj saradnji njegovih objekata. Ako se prilikom razvijanja sistema usredsredite na takve mehanizme, dobićete manju, jednostavniju i razumljiviju arhitekturu nego ako ignorišete te uzorke.

Značaj uzoraka u proizvodnji složenih sistema odavno je priznat u drugim disciplinama. Konkretno, Christopher Alexander i njegove kolege su možda prvi predložili da se jezik uzoraka upotrebi u arhitekturi zgrada i gradova. Njegove ideje i drugi doprinosi sada su ukorenjeni u zajednici objektno orijentisanog softvera. Ukratko, koncept uzoraka za projektovanje u softveru predstavlja ključ koji pomaže razvijaocima da iskoriste stručnost drugih kvalifikovanih arhitekata.

U ovoj knjizi Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides uvode principe uzoraka za projektovanje i zatim daju katalog tih uzoraka. Prema tome, ova knjiga ima dva važna doprinosa. Prvo, pokazuje ulogu koju uzorci mogu da igraju u arhitekturi složenih sistema. Drugo, predstavlja veoma poučnu referencu za niz dobro projektovanih uzoraka koje razvijalac praktičar može da primeni u izgradnji vlastitih aplikacija.

Čast mi je što imam priliku da neposredno sarađujem sa nekim autorima ove knjige na zadacima projektovanja arhitektura. Mnogo sam od njih naučio, a mislim da ćete i vi kada pročitate ovu knjigu.

Grady Booch

Rukovodeći naučnik, Rational Software Corporation

Uputstvo za čitaoce

Ova knjiga se sastoji od dva glavna dela. U prvom delu (poglavlja 1 i 2) opisuje se šta su to uzorci za projektovanje i kako oni pomažu u projektovanju objektno orijentisanog softvera. Uključena je studija slučajeva projektovanja kojom se prikazuje kako se uzorci za projektovanje primenjuju u praksi. Drugi deo knjige (poglavlja 3, 4 i 5) predstavlja katalog samih uzoraka za projektovanje.

Katalog zaprema najveći deo knjige. Poglavlja dele uzorke za projektovanje u tri tipa: uzorke za pravljenje, strukturne uzorke i uzorke ponašanja. Katalog možete da koristite na više načina. Možete da ga pročitate od početka do kraja ili možete da listate od uzorka do uzorka. Drugi pristup je da prostudirate jedno od poglavlja. Tako ćete lakše uočiti po čemu se razlikuju tesno povezani uzorci.

Referisanje od uzorka na uzorak možete da koristite kao logičku putanju po katalogu. Ovakvim pristupom shvatićete odnose među uzorcima, kako se mogu kombinovati i koji uzorci dobro funkcionišu jedni sa drugima. Ovo referisanje je grafički prikazano na slici 1.1 (strana 12).

Još jedan način da se čita katalog je da se koristi pristup upravljanja problemom. Preskočite do odeljka 1.6 (strana 24) i čitajte o nekim uobičajenim problemima u projektovanju više puta upotrebljivog objektno orijentisanog softvera; zatim pročitajte uzorke koji rešavaju te probleme. Neki pročitaju ceo katalog, a *zatim* koriste pristup upravljanja problemom da bi te uzorke primenili u svojim projektima.

Ako niste iskusni objektno orijentisani dizajner, počnite od najjednostavnijih i najobičnijih uzoraka:

- Apstraktna fabrika (Abstract Factory, strana 87)
- Adapter (139)
- Sastav (Composite, 163)
- Dekorater (Decorator, 175)
- Proizvodni metod (Factory Method, 107)
- Nadzornik (Observer, 293)
- Strategija (Strategy, 315)
- Šablonski metod (Template Method, 325)

Teško je naći objektno orijentisan sistem koji ne koristi bar neki od ovih uzoraka dok veliki sistemi koriste skoro sve. Ovaj podskup će vam pomoći da razumete pojedinačne uzorke za projektovanje, kao i opšte principe dobrog objektno orijentisanog projektovanja.

Poglavlje 1

Uvod

Projektovanje objektno orijentisanog softvera je teško, a projektovanje *više puta upotrebljivog* objektno orijentisanog softvera je još teže. Morate naći odgovarajuće objekte, urediti ih u klase sa ispravnom granularnošću, definisati interfejse klasa i hijerarhije nasleđivanja i uspostaviti ključne odnose među njima. Projekat bi trebalo da bude specifičan za konkretan problem, ali i dovoljno uopšten da prihvati buduće probleme i zahteve. Takođe je dobro izbeći ponovno projektovanje ili ga bar svesti na najmanju moguću meru. Iskusni objektno orijentisani projektanti reći će vam da je teško, ako ne i nemoguće, iz prvog pokušaja napraviti „dobar" više puta upotrebljiv i fleksibilan dizajn. Da bi napravili projekat, oni obično nekoliko puta pokušavaju ponovo da ga upotrebe i svaki put ga menjaju.

Ipak, iskusni objektno orijentisani projektanti prave dobre projekte. Za to vreme su novi projektanti zbunjeni količinom ponuđenih opcija, pa se vraćaju tehnikama koje nisu objektno orijentisane, a koje su ranije koristili. Potrebno je vreme da bi početnici naučili u čemu je suština dobrog objektno orijentisanog dizajna. Iskusni projektanti očigledno znaju nešto što neiskusni ne znaju. Šta je to?

Iskusni projektanti znaju da *ne treba* svaki problem rešavati od početka. Umesto toga oni ponovo koriste rešenja koja su uspešno koristili ranije. Kada nađu dobro rešenje, i uvek ga ponovo koriste. Takvo iskustvo ih i čini ekspertima. Zbog toga ćete u mnogim objektno orijentisanim sistemima nailaziti na iste uzorke klasa i objekata koji komuniciraju. Ovi uzorci rešavaju specifične probleme projektovanja i omogućavaju fleksibilnost, eleganciju i ponovnu upotrebljivost objektno orijentisanih dizajna. Oni projektantima pomažu da ponovo upotrebe uspešne projekte tako što će novi projekat zasnovati na prethodnom iskustvu. Projektant, koji poznaje takve uzorke, može ih odmah primeniti na rešenje problema i ne mora ponovo da ih otkriva.

Evo jedne analogije koja ilustruje situaciju. Pisci romana i komada retko planiraju zaplet od nule. Umesto toga oni koriste uzorke kao što su „tragično nesavršen heroj" (Magbet, Hamlet, itd.) ili „romantična priča" (bezbroj romantičnih romana). Na isti način, objektno orijentisani projektanti koriste uzorke kao što su „predstavljanje stanja objektima" ili „dekorisanje objekata tako da im se lako dodaju i uklanjaju karakteristike". Kad jednom znate uzorak, mnoge odluke o projektu slede automatski.

Svi znamo koliko u projektovanju vredi iskustvo. Koliko puta ste pri projektovanju imali osećaj „već viđenog" – taj osećaj da ste neki problem već rešili, ali ne znate tačno gde i kako? Kad biste se setili detalja prethodnog problema i kako ste ga rešili, mogli biste ponovo da upotrebite iskustvo, a ne da ga ponovo otkrivate. Međutim, obično ne beležimo u dovoljnoj meri iskustva u projektovanju softvera da bi drugi mogli da ih koriste.

Svrha ove knjige je da zabeleži iskustva projektovanja objektno orijentisanog softvera u vidu uzoraka za projektovanje. Svaki uzorak za projektovanje sistematski imenuje, objašnjava i razrađuje neki značajan problem koji se ponavlja u objektno orijentisanim sistemima. Naš cilj je da uhvatimo iskustvo projektanta u obliku koji drugi mogu efikasno da upotrebe. U tu svrhu smo dokumentovali neke od najvažnijih uzoraka za projektovanje i predstavili ih u obliku kataloga.

Uzorci za projektovanje olakšavaju ponovno korišćenje uspešnih modela i arhitektura. Kada se proverene tehnike prikažu u obliku uzoraka za projektovanje, one su pristupačnije razvijaocima novih sistema. Uzorci za projektovanje pomažu da se izaberu alternative koje omogućavaju da sistem bude više puta upotrebljiv i da se izbegnu alternative koje nije moguće više puta upotrebiti. Uzorci za projektovanje mogu čak da unaprede dokumentaciju i održavanje postojećih sistema zato što daju eksplicitnu specifikaciju međudejstva klasa i objekata i njihovu osnovnu namenu. Jednostavno rečeno, uzorci za projektovanje pomažu projektantu da brže napravi „ispravan" projekat.

Nijedan od uzoraka za projektovanje prikazanih u ovoj knjizi ne opisuje nove ili neproverene modele. Uključili smo samo one koji su više puta primenjeni u različitim sistemima. Većina njih nije nikada ranije dokumentovana. Oni ili spadaju u folklor objektno orijentisane zajednice ili su elementi nekih uspešnih objektno orijentisanih sistema, a početnik iz toga ne može lako da uči. Prema tome, mada ovi uzorci za projektovanje nisu novi, mi smo ih sabrali na nov i pristupačan način: kao katalog uzoraka za projektovanje u konzistentnom formatu.

I pored obimnosti ove knjige, ona obuhvata samo delić onoga što znaju eksperti. Ovde nema uzoraka vezanih za konkurentnost, distribuirano programiranje ili za programiranje u realnom vremenu. Nema ni uzoraka aplikacija specifičnih za pojedina područja. Nema objašnjenja kako se prave korisnički interfejsi, kako se pišu upravljački programi za uređaje ili kako se koristi objektno orijentisana baza podataka. Svaka od ovih oblasti ima vlastite uzorke i vredelo bi da neko napravi i njihove kataloge.

1.1 Šta je to uzorak za projektovanje?

Christopher Alexander kaže: „Svaki uzorak opisuje problem koji se stalno ponavlja u našem okruženju i zatim opisuje suštinu rešenja problema tako da se to rešenje može upotrebiti milion puta, a da se dva puta ne ponovi na isti način". [AIS+77, strana *x*]. Mada je Alexander govorio o uzorcima za zidanje gradova, to što je rekao važi i za uzorke objektno orijentisanog projektovanja. Naša rešenja su izražena pomoću objekata i interfejsa umesto zidova i vrata, ali suština obe vrste uzoraka je rešenje problema u svom kontekstu.

Uopšteno gledano, uzorak ima četiri bitna elementa:

1. **Ime uzorka** koristimo da bismo u nekoliko reči opisali problem, njegova rešenja i njegove posledice. Kada uzorku damo ime, odmah nam se uvećava rečnik projektovanja. To nam omogućava projektovanje na većem nivou apstrakcije. Kada imamo rečnik uzoraka, možemo o njima diskutovati sa kolegama, u dokumentaciji, pa čak i sami. Tako je lakše razmišljati o projektovanju i prenositi drugima taj model i njegove ocene. Pronalaženje dobrih imena bio je jedan od najtežih poslova pri izradi ovog kataloga.

2. **Problem** je opis situacije u kojoj se uzorak koristi. Opisuje se problem i njegov kontekst. Moguć je opis specifičnih problema, kao kada treba predstaviti algoritam objektom. Problem može da opisuje strukture klasa ili objekata čije osobine nagoveštavaju kruto projektovanje. Ponekad problem sadrži spisak uslova potrebnih da bi se primenio taj uzorak.

3. **Rešenje** opisuje elemente koji čine dizajn, njihove odnose, odgovornosti i saradnju. Ono ne opisuje određen konkretan projekat ili implementaciju pošto je uzorak kao šablon koji se može primeniti u mnogim različitim situacijama. Umesto toga, uzorak daje apstraktan opis problema projektovanja i kako se on rešava opštim uređenjem elemenata (u našem slučaju, klasa i objekata).

4. **Posledice** su rezultati i ocene primene uzorka. Mada se posledice često ne pominju u opisima odluka o projektovanju, one su bitne za procenu alternativa i za razumevanje gubitaka i dobitaka od primene uzorka.

   U softveru se posledice često mere prostorom i vremenom. Moguća su i pitanja jezika i implementacije. Pošto je ponovna upotrebljivost u objektno orijentisanom projektovanju često važan faktor, posledice uzorka obuhvataju njegov uticaj na fleksibilnost sistema, na njegovu proširivost ili prenosivost. Eksplicitno navođenje ovih posledica pomaže u njihovom razumevanju i njihovoj proceni.

Od gledišta zavisi šta jeste, a šta nije uzorak. Što je za nekoga uzorak može za drugoga biti primitivan element gradnje. U ovoj knjizi smo se skoncentrisali na uzorke na određenom nivou apstrakcije. *Gotova rešenja* se ne bave povezanim listama ili heš tabelama koje se mogu kodirati u klase i ponovo koristiti kakve jesu. S druge strane, uzorci nisu ni složeni projekti specifični za neko područje, za celu aplikaciju ili podsistem. Uzorci za projektovanje u ovoj knjizi su *opisi objekata i klasa koje komuniciraju i prilagođene su rešavanju opšteg problema projektovanja u određenom kontekstu*.

Uzorak za projektovanje imenuje, apstrahuje i identifikuje ključne aspekte zajedničke strukture zbog kojih je pogodan za više puta upotrebljivi objektno orijentisani projekat. Uzorak za projektovanje identifikuje klase i primerke, njihove uloge i međudejstvo i distribuciju njihovih odgovornosti. Svaki uzorak za projektovanje koncentriše se na određeni problem ili pitanje objektno orijentisanog projektovanja. On opisuje kada se uzorak primenjuje, da li se može primeniti s obzirom na ostala ograničenja, kao i posledice i ocenu njegove upotrebe. Pošto dizajn treba kad – tad implementirati, uzorak za projektovanje

sadrži i primere programa na jezicima C++ i (ponekad) Smalltalk kao ilustraciju implementacije.

Mada uzorci za projektovanje opisuju objektno orijentisane projekte, oni se zasnivaju na praktičnim rešenjima koja su implementirana na glavnim objektno orijentisanim programskim jezicima, kao što su Smalltalk i C++, a ne na proceduralnim jezicima (Pascal, C, Ada) ili dinamičnijim objektno orijentisanim jezicima (CLOS, Dylan, Self). Izabrali smo Smalltalk i C++ iz praktičnih razloga: naše svakodnevno iskustvo je na tim jezicima i oni postaju sve popularniji.

Izbor programskog jezika je značajan zato što utiče na tačku gledišta. Naši uzorci polaze od mogućnosti jezika nivoa Smalltalk i C++ i taj izbor određuje šta se može, a šta ne može lako implementirati. Da smo pošli od proceduralnih jezika, mogli smo da uključimo uzorke za „nasleđivanje", „enkapsuliranje" i „polimorfizam". Slično tome, neki od naših uzoraka već su direktno primenjeni u manje raširenim objektno orijentisanim jezicima. CLOS, na primer, ima višestruke-metode zbog kojih ima manje potrebe za uzorcima kao što je Posetilac (Visitor, strana 331). U suštini, između Smalltalka i C++a su bitne razlike, tako da se neki uzorci lakše izražavaju na jednom jeziku nego na drugom. (Pogledajte, na primer, uzorak Iterator, strana 257).

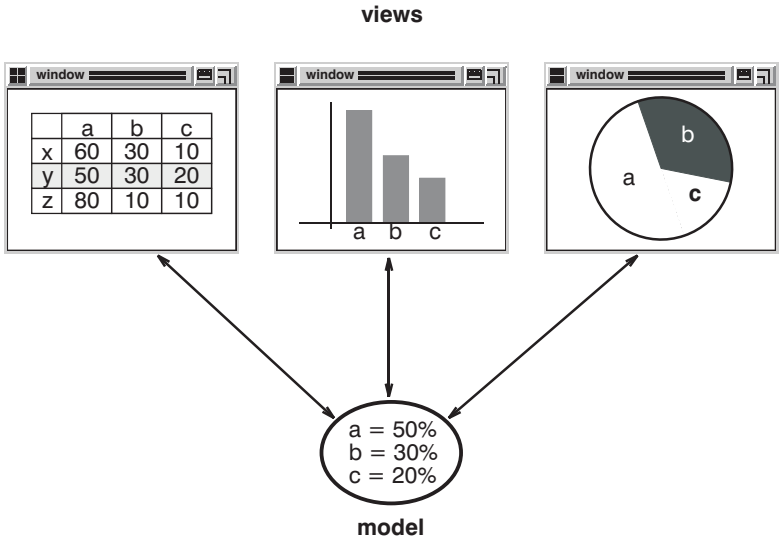
1.2 Uzorci za projektovanje u Smalltalk MVC jeziku

Trijada klasa Model/Prikaz/Kontroler (Model/View/Controller, MVC) [KP88] se koristi za izgradnju korisničkih interfejsa za Smalltalk-80. Ako pogledate uzorke za projektovanje u MVC jeziku, biće vam jasnije šta podrazumevamo pod terminom „uzorak".

MVC se sastoji od tri vrste objekata. Model je objekat aplikacije, Prikaz je ekranska prezentacija, a Kontroler definiše način na koji korisnički interfejs reaguje na akciju korisnika. Dok nije postojao MVC, u projektima korisničkih interfejsa su se ovi objekti uglavnom spajali. MVC ih rastavlja da bi se povećala fleksibilnost i mogućnost višekratne upotrebe.

MVC rastavlja prikaze od modela i uspostavlja među njima protokol pretplate/obaveštenja. Prikaz mora da obezbedi da njegov izgled zavisi od stanja modela. Kad god se promene podaci modela, model obaveštava prikaze koji od njega zavise. Kao odgovor, svaki prikaz dobija mogućnost da se ažurira. Ovaj pristup omogućava da se jednom modelu pridruži više prikaza da bi se dobile različite prezentacije. Možete takođe da napravite nove prikaze istog modela bez izmena u samom modelu.

Sledeći dijagram pokazuje model u tri prikaza. (Radi jednostavnosti smo izostavili odgovarajuće kontrolere). Model sadrži neke vrednosti podataka, a prikazi u vidu radnog lista, histograma i kružnog dijagrama prikazuju te podatke na različite načine. Model komunicira sa svojim prikazima kada mu se menjaju vrednosti, a prikazi komuniciraju sa modelom da bi došli do tih vrednosti.



Na prvi pogled, ovaj primer prikazuje dizajn koji razdvaja prikaze od modela. Ali taj dizajn može da se primeni na opštiji problem rastavljanja objekata tako da promene u jednom objektu mogu da utiču na veći broj drugih objekata bez potrebe da promenjeni objekat poznaje detalje drugih objekata. Ovaj opštiji dizajn opisan je u uzorku za projektovanje Posmatrač (Observer, strana 293).

Još jedno svojstvo MVC jezika jeste da se prikazi mogu ugnezditi. Na primer, kontrolni panel sa dugmadima može se implementirati kao složeni prikaz koji sadrži ugneždene prikaze dugmadi. Korisnički interfejs za posmatrača objekata može da se sastoji od ugneždenih prikaza koji se mogu ponovo koristiti u programu za otklanjanje grešaka. MVC podržava ugneždene prikaze pomoću klase CompositeView, potklase klase View. Objekti klase CompositeView se ponašaju kao objekti klase View; složeni prikaz se može koristiti svuda gde i prikaz, ali takođe sadrži i upravlja ugneždenim prikazima.

I ovde bismo mogli to da shvatimo kao dizajn koji dozvoljava da se složeni prikaz tretira kao jedna od njegovih komponenti, ali se taj dizajn može primeniti na opštiji problem do kojeg dolazi kad god hoćemo da grupišemo objekte i da postupamo sa grupom kao sa pojedinačnim objektom. Taj opštiji dizajn je opisan u uzorku za projektovanje Sastav (Composite, 163). On omogućava pravljenje hijerarhije klasa u kojoj neke potklase definišu osnovne objekte (npr. Button), a druge klase definišu složene objekte (CompositeView) koji prikupljaju osnovne objekte u složenije.

MVC takođe dozvoljava da menjate način na koji prikaz reaguje na ulaz od korisnika bez menjanja vizuelne prezentacije. Mogli biste, na primer, da promenite reagovanje na tastaturu ili da uvedete pomoćni meni umesto komandnih tastera. MVC enkapsulira mehanizam reagovanja u objektu Kontroler. Hijerarhija klasa kontrolera već postoji, pa se lako pravi novi kontroler kao varijacija postojećeg.

Prikaz koristi instancu potklase Kontroler za implementiranje određene strategije reagovanja; da bi se implementirala drugačija strategija, jednostavno

zamenite tu instancu drugom vrstom kontrolera. Čak je moguće promeniti kontroler prikaza u vreme izvršavanja da bi prikaz promenio način reagovanja na ulaz od korisnika. Na primer, prikaz može da se onemogući tako da ne prihvata ulaz jednostavno na taj način što mu se dodeli kontroler koji ignoriše ulazne događaje.

Odnos Prikaz-Kontroler je primer uzorka za projektovanje Strategija (Strategy, strana 315). Strategija je objekat koji predstavlja algoritam. Koristan je kada algoritam hoćete da zamenite bilo statički ili dinamički, kada imate mnogo varijanti algoritma ili kada algoritam sadrži složene strukture koje želite da enkapsulirate.

MVC koristi i druge uzorke za projektovanje kao što je Proizvodni metod (Factory Method, strana 107) kojim se određuje klasa kontrolera za prikaz i Dekorater (Decorator, strana 175) da bi se omogućilo pomeranje sadržaja prikaza. Međutim, glavni odnosi MVC jezika dati su u uzorcima za projektovanje Posmatrač, Sastav i Strategija.

1.3 Opisivanje uzoraka za projektovanje

Kako opisujemo uzorke za projektovanje? Grafička sredstva nisu dovoljna iako su važna i korisna. Ona jednostavno prikazuju konačni proizvod procesa projektovanja u vidu odnosa među klasama i objektima. Da bismo ponovo koristili isti dizajn, moramo da zabeležimo odluke, alternative i procene koje su do njega dovele. Konkretni primeri su takođe važni zato što vam pomažu da vidite dizajn na delu.

Mi opisujemo uzorke za projektovanje pomoću konzistentnog formata. Opis svakog uzorka je podeljen na odeljke prema sledećem šablonu. Šablon daje informacijama ujednačenu strukturu, tako da se uzorci za projektovanje lakše uče, porede i koriste.

Ime uzorka i klasifikacija

Ime uzorka sažeto izlaže suštinu uzorka. Dobro ime je od suštinskog značaja pošto ono ulazi u rečnik projektovanja. Klasifikacija uzorka izvedena je po šemi koju uvodimo u odeljku 1.5.

Namena

Kratak iskaz koji odgovara na sledeća pitanja: šta uzorak za projektovanje radi? Kakvo mu je obrazloženje i namena? Na koje se konkretno pitanje ili problem projektovanja uzorak odnosi?

Poznat takođe kao

Ostala poznata imena uzorka, pod uslovom da postoje.

Motivacija

Scenario koji ilustruje problem projektovanja i način na koji strukture klasa i objekata u uzorku rešavaju taj problem. Scenario će vam pomoći da shvatite apstraktniji opis uzorka koji sledi.

Primenjivost

Na koje situacije se uzorak može primeniti? Koji su primeri lošeg projektovanja koje uzorak može da ispravi? Kako prepoznati te situacije?

Struktura

Grafički prikaz klasa u uzorku dat je u notaciji zasnovanoj na Tehnici objektnog modeliranja (*Object Modeling Technique*) [RBP+91]. Koristimo takođe i dijagrame interakcije [JCJO92, Boo94] za ilustrovanje nizova zahteva i saradnje među objektima. Ove notacije su detaljno opisane u dodatku B.

Učesnici

Klase i/ili objekti koji učestvuju u uzorku za projektovanje kao i njihove odgovornosti.

Saradnja

Kako učesnici sarađuju da bi izvršili svoje odgovornosti.

Posledice

Kako uzorak zadovoljava svoju namenu? Koji su nedostaci i koristi od korišćenja uzorka? Koji aspekt strukture sistema može nezavisno da se menja?

Implementacija

Kojih zamki, saveta ili tehnika bi trebalo da budete svesni prilikom implementiranja uzorka? Ima li nekih pitanja zavisnih od programskog jezika?

Primer koda

Fragmenti koda koji ilustruju kako bi se uzorak mogao implementirati na jeziku C++ ili Smalltalk.

Poznati primeri korišćenja

Primeri uzorka koji se nalaze u stvarnim sistemima. Uvek dajemo bar dva primera iz različitih oblasti.

Povezani uzorci

Koji uzorci za projektovanje su u tesnoj vezi sa ovim uzorkom? Koje su značajne razlike? Uz koje druge uzorke bi trebalo koristiti ovaj uzorak?

Dodaci sadrže neke osnovne informacije koje će vam pomoći da shvatite uzorke i diskusije koje ih prate. Dodatak A je glosar termina koje koristimo. Već smo pomenuli dodatak B u kojem su predstavljene različite notacije. Aspekte notacija

ćemo opisivati i dok ih budemo uvodili u sledećem tekstu. Na kraju, dodatak C sadrži izvorni kôd osnovnih klasa koje se koriste u primerima koda.

1.4 Katalog uzoraka za projektovanje

Katalog koji počinje na strani 79 sadrži 23 uzorka za projektovanje. Njihova imena i namene navodimo ovde da biste imali pregled. Brojevi u zagradama predstavljaju stranicu na kojoj se nalazi taj uzorak (to je konvencija koje se držimo u celoj knjizi).

Apstraktna fabrika (Abstract Factory) (87) Predstavlja interfejs za pravljenje familija logički ili funkcionalno povezanih objekata bez navođenja njihovih konkretnih klasa.

Graditelj (Builder) (97) Razdvaja izgradnju složenog objekta od njegove reprezentacije da bi isti proces pravljenja mogao da proizvede različite reprezentacije.

Proizvodni metod (Factory Method) (107) Definiše interfejs za pravljenje objekta, ali prepušta potklasi odluku o klasi primerka. Proizvodni metod dozvoljava klasi da prepusti pravljenje primerka potklasi.

Prototip (Prototype) (117) Određuje vrste objekata koji se prave koristeći prototipski primerak i pravi nove objekte kopiranjem tog prototipa.

Unikat (Singleton) (127) Obezbeđuje da klasa ima samo jedan primerak i daje mu globalnu tačku pristupa.

Adapter (Adapter) (139) Konvertuje interfejs klase u drugi interfejs koji klijenti očekuju. Adapter omogućava saradnju klasa koje inače ne bi mogle da sarađuju zbog nekompatibilnih interfejsa.

Most (Bridge) (151) Izdvaja apstrakciju od njene implementacije da bi se one mogle nezavisno menjati.

Sastav (Composite) (163) Sastavlja objekte u strukturu stabla da bi se predstavile hijerarhije delova i celina. Sastav omogućava klijentima ujednačeno postupanje sa pojedinačnim objektima i sa sastavima objekata.

Dekorater (Decorator) (175) Dinamički pridružuje objektu dodatne odgovornosti. Dekorateri su fleksibilna alternativa mehanizmu nasleđivanja radi proširivanja funkcionalnosti.

Fasada (Facade) (185) Pruža ujednačen interfejs za skup interfejsa jednog podsistema. Fasada definiše interfejs višeg nivoa da bi se podsistem lakše koristio.

Muva (Flyweight) (195) Koristi deljenje za efikasnu podršku velikog broja sitnijih objekata.

Proksi (Proxy) (207) Predstavlja zamenu ili posrednika za pristupanje drugom objektu radi ostvarivanja kontrole pristupa.

Lanac odgovornosti (Chain of Responsability) (223) Izbegava neposredno vezivanje pošiljaoca zahteva sa njegovim primaocem dajući šansu da više objekata rukuje zahtevom. Povezuje objekte primaoce i prosleđuje zahtev niz lanac dok ga neki objekat ne obradi.

Komanda (Command) (233) Enkapsulira zahtev kao objekat, čime omogućava parametrizaciju klijenata sa različitim zahtevima, redovima za čekanje, ili zahtevima za formiranje dnevnika rada, i podržava operacije čije se dejstvo može poništiti.

Interpretator (Interpreter) (243) Za dati jezik definiše predstavljanje njegove gramatike uz interpretator koji koristi tu predstavu za tumačenje iskaza navedenog jezika.

Iterator (Iterator) (257) Predstavlja sredstvo za sekvencijelno pristupanje elementima objekta agregata bez izlaganja njegove unutrašnje reprezentacije.

Posrednik (Mediator) (273) Definiše objekat koji enkapsulira međudejstva skupa objekata. Posrednik potpomaže slabo vezivanje, jer sprečava eksplicitno međusobno referisanje objekata i omogućava nezavisno menjanje njihovih međudejstava.

Podsetnik (Memento) (283) Bez narušavanja enkapsulacije, preuzima i ispoljava unutrašnje stanje objekta da bi se objekat mogao kasnije vratiti u to stanje.

Posmatrač (Observer) (293) Definiše zavisnost jedan prema više među objektima da bi prilikom promene stanja jednog objekta svi zavisni objekti bili obavešteni i automatski ažurirani.

Stanje (State) (305) Omogućava da objekat promeni ponašanje kada mu se promeni unutrašnje stanje. Izgledaće kao da je objekat promenio klasu.

Strategija (Strategy) (315) Definiše familiju algoritama, enkapsulira svakog od njih i omogućava da se mogu međusobno zamenjivati. Strategija omogućava da se algoritam menja nezavisno od klijenata koji ga koriste.

Šablonski metod (Template Method) (325) Definiše skelet algoritma neke operacije prepuštajući implementaciju nekih koraka potklasi. Šablonski metod dozvoljava da potklase redefinišu neke korake algoritma ne menjajući strukturu algoritma.

Posetilac (Visitor) (331) Predstavlja operaciju koju treba izvršiti nad elementima strukture objekta. Posetilac dozvoljava definisanje nove operacije bez izmena klase elemenata nad kojima vrši operaciju.

1.5 Organizovanje kataloga

Uzorci za projektovanje se razlikuju po granularnosti i stepenu apstrakcije. Pošto ih ima mnogo, potreban nam je neki način da ih organizujemo. U ovom odeljku klasifikujemo uzorke za projektovanje da bismo mogli da pominjemo familije vezanih uzoraka. Klasifikacija pomaže bržem učenju uzoraka iz kataloga, a može i da usmeri napore u pravcu stvaranja novih uzoraka.

		Namena		
		Pravljenje	**Struktura**	**Ponašanje**
Domen	**Klasa**	Proizvodni metod (Factory Method) (107)	Adapter (Adapter) (klasa) (139)	Interpretator (Interpreter) (243) Šablonski metod (Template Method) (325)
	Objekat	Apstraktna fabrika (Abstract Factory) (87) Graditelj (Builder) (97) Prototip (Prototype) (117) Singlton (Singleton) (127)	Adapter (Adapter) (objekat) (139) Most (Bridge) (151) Sastav (Composite) (163) Dekorater (Decorator) (175) Fasada (Facade) (185) Muva (Flyweight) (195) Proksi (Proxy) (207)	Lanac odgovornosti (Chain of Responsability) (223) Komanda (Command) (233) Iterator (Iterator) (257) Posrednik (Mediator) (273) Podsetnik (Memento) (283) Posmatrač (Observer) (293) Stanje (State) (305) Strategija (Strategy) (315) Posetilac (Visitor) (331)

Tabela 1.1: Prostor uzoraka za projektovanje

Uzorke za projektovanje klasifikujemo na osnovu dva kriterijuma (tabela 1.1). Prvi kriterijum, namena, označava šta uzorak radi. Uzorci mogu imati namenu pravljenja, strukture ili ponašanja. Uzorci za pravljenje bave se postupkom izrade objekata. Strukturni uzorci se bave sastavljanjem klasa i objekata. Uzorci ponašanja opisuju kako klase ili objekti međusobno utiču jedni na druge i kako dele odgovornosti.

Drugi kriterijum, domen, označava da li se uzorak primenjuje pre svega na klase ili na objekte. Uzorci klasa bave se odnosima između klasa i njihovih potklasa. Ti odnosi se uspostavljaju nasleđivanjem i zato su statični – fiksiraju se tokom prevođenja. Uzorci objekata se bave odnosima između objekata koji mogu da se menjaju u vreme izvršavanja i dinamičniji su. Skoro svi uzorci do neke mere koriste nasleđivanje. Prema tome, jedino se uzorci označeni kao „uzorci klasa" usredsređuju na odnose klasa. Primetićete da većina uzoraka ima domen Objekat.

Uzorci klasa za pravljenje prepuštaju neki deo pravljenja objekata potklasama dok uzorci objekata za pravljenje prepuštaju to drugom objektu. Strukturni uzorci klasa koriste nasleđivanje za sastavljanje klasa, dok strukturni uzorci objekata opisuju načine grupisanja objekata. Uzorci ponašanja klasa koriste nasleđivanje za opisivanje algoritama i toka kontrole, dok uzorci ponašanja objekata opisuju kako grupa objekata sarađuje na zadatku koji ni jedan objekat ne može sam da obavi.

Postoje i drugi načini da se organizuju uzorci. Neki uzorci se često koriste zajedno. Na primer, Sastav se često koristi uz Iterator ili Posetioca. Neki uzorci su alternative: Prototip je često alternativa Apstraktnoj fabrici. Neki uzorci daju slične dizajne iako imaju različite namene. Na primer, dijagrami struktura uzoraka Sastav i Dekorater su slični.

Još jedan način da se organizuju uzorci za projektovanje jeste prema načinu na koji referišu jedni na druge u svojim odeljcima „Povezani uzorci". Ti odnosi su grafički prikazani na slici 1.1.

Jasno je da ima mnogo načina da se uzorci za projektovanje organizuju. Ako razmišljate o uzorcima na različite načine, bolje ćete uvideti njihovu funkciju, njihove razlike i kada da ih upotrebite.

1.6 Kako uzorci za projektovanje rešavaju probleme projektovanja

Uzorci za projektovanje rešavaju mnoge svakodnevne probleme na koje nailaze objektno orijentisani projektanti, i to na različite načine. Evo nekoliko tih problema i načina na koje ih uzorci za projektovanje rešavaju.

Pronalaženje odgovarajućih objekata

Objektno orijentisani programi se sastoje od objekata. **Objekat** je sastavljen od podataka i od procedura nad tim podacima. Procedure se obično nazivaju **metodima** ili **operacijama**. Objekat izvršava operaciju kada dobije **zahtev** (ili **poruku**) od klijenta.

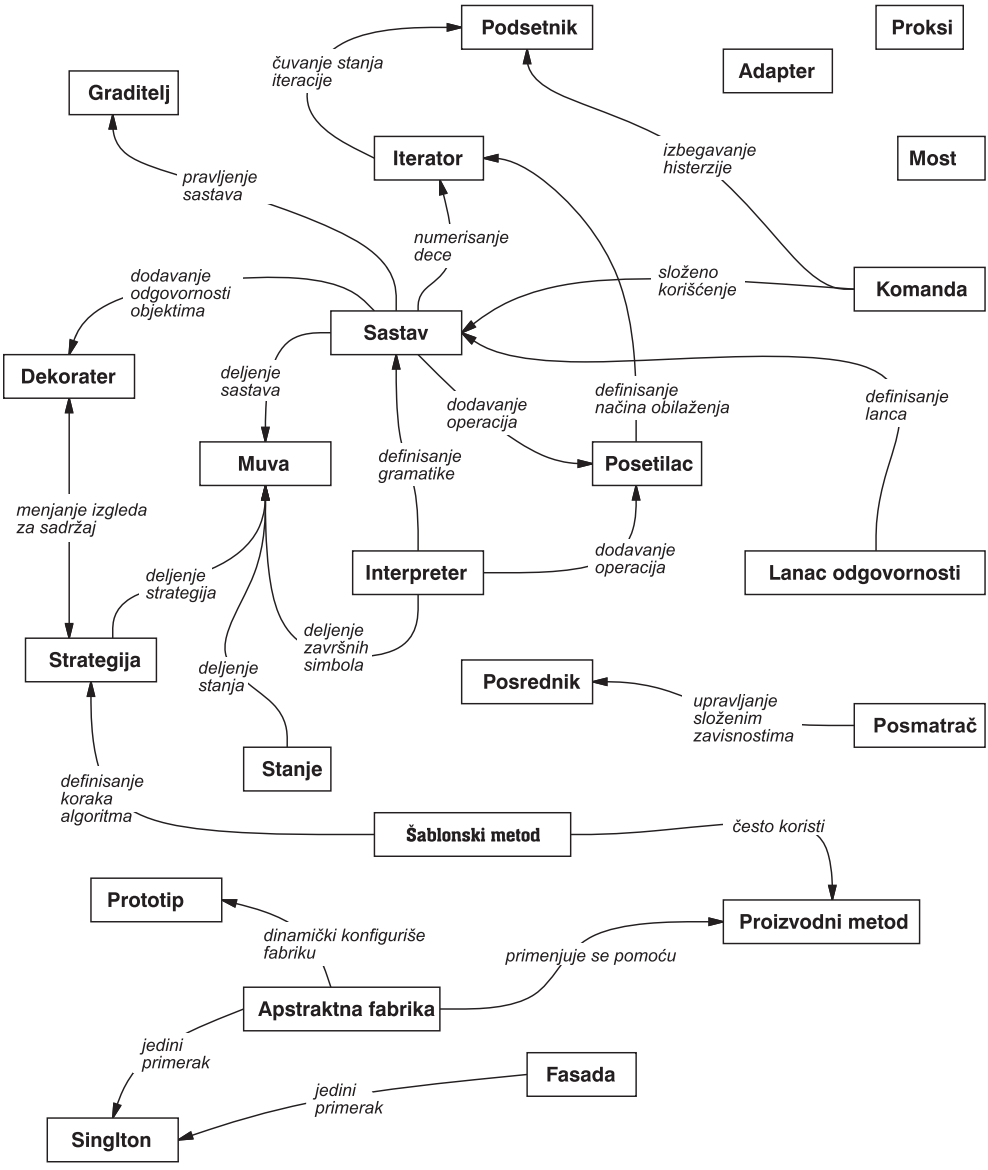
Zahtevi su *jedini* način da se objekat navede da izvrši operaciju. Operacije su *jedini* način da se promene unutrašnji podaci objekta. Zbog tih ograničenja kažemo da je unutrašnje stanje objekta **enkapsulirano**; ne može mu se prići neposredno i njegova reprezentacija nije vidljiva izvan objekta.

Teži deo objektno orijentisanog projektovanja je rastavljanje sistema na objekte. Zadatak je težak zato što su u igri mnogi faktori: enkapsuliranje, granularnost, zavisnosti, fleksibilnost, performanse, razvoj, višekratna upotrebljivost i tako dalje. Svi ti faktori utiču na rastavljanje, često na suprotstavljene načine.

Metodologije objektno orijentisanog projektovanja nude različite pristupe. Možete da napišete postavku problema, izdvojite imenice i glagole i napravite odgovarajuće klase i operacije. Možete i da se skoncentrišete na saradnju i odgovornosti u sistemu. Možete i da modelirate stvarni svet i da objekte koje nađete u analizi prevedete u model. Uvek će biti sporenja oko toga koji je pristup bolji.

Mnogi objekti u projektima potiču iz analize modela. Međutim, u objektno orijentisanim projektima često se nađu klase koje nemaju pandan u stvarnom svetu. Neke od njih su klase nižeg nivoa, kao što su nizovi. Druge su mnogo višeg nivoa. Na primer, uzorak Sastav (Composite) (163) uvodi jednu apstrakciju za uniformno pristupanje objektima koja nema fizički pandan. Strogo modeliranje

stvarnog sveta vodi sistemu koji odražava današnju realnost, ali se ona možda ne slaže sa sutrašnjom realnošću. Apstrakcije koje se pojavljuju prilikom projektovanja ključne su za fleksibilnost dizajna.



Slika 1.1: Odnosi između uzoraka

Uzorci za projektovanje vam pomažu da prepoznate manje očigledne apstrakcije i objekte koji mogu da ih obuhvate. Na primer, objekti koji predstavljaju proces ili algoritam ne postoje u prirodi, ali predstavljaju ključni deo fleksibilnog projektovanja. Uzorak Strategija (315) opisuje kako da se implementiraju međusobno zamenjive familije algoritama. Uzorak Stanje (305) predstavlja svako stanje entiteta kao poseban objekat. Na takve objekte se retko nailazi tokom analize, pa čak i u ranim fazama projektovanja; oni se otkrivaju kasnije u pokušajima da se postigne fleksibilnost i višekratna upotrebljivost.

Određivanje granularnosti objekata

Objekti mogu mnogo da se razlikuju po veličini i po broju. U stanju su da predstavljaju sve, od samog hardvera, pa do celih aplikacija. Kako se donosi odluka o tome šta bi trebalo da bude objekat?

Uzorci za projektovanje se bave i ovim pitanjem. Uzorak Fasada (185) opisuje kako treba predstaviti čitave podsisteme kao objekte, a uzorak Muva (195) opisuje kako treba podržati veliki broj objekata najfinije granularnosti. Drugi uzorci za projektovanje opisuju specifične načine razlaganja objekta na manje objekte. Uzorci Apstraktna fabrika (87) i Graditelj (97) daju objekte čija je jedina odgovornost da prave druge objekte. Uzorci Posetilac (331) i Komanda (233) daju objekte čije su jedine odgovornosti da implementiraju zahtev na drugom objektu ili na grupi objekata.

Određivanje interfejsa objekata

Svaka operacija deklarisana objektom navodi ime operacije, objekte koje uzima za parametre i vraćeni rezultat operacije. To se naziva **potpisom** operacije. Skup svih potpisa definisanih operacijama objekta naziva se **interfejsom** objekta. Interfejs objekta opisuje potpun skup zahteva koji se mogu uputiti objektu. Objektu se može uputiti svaki zahtev koji odgovara nekom potpisu u interfejsu objekta.

Tip je naziv koji označava konkretan interfejs. Kažemo da objekat ima tip „Prozor" ako prihvata sve zahteve za operacije definisane u interfejsu sa imenom „Prozor". Objekat može imati više tipova, a veoma različiti objekti mogu imati zajednički tip. Jedan deo interfejsa objekta može da pripada jednom tipu, a drugi delovi drugim tipovima. Za dva objekta istog tipa mogu da budu zajednički samo delovi interfejsa. Interfejsi mogu da sadrže druge interfejse kao podskupove. Za tip kažemo da je **podtip** drugog tipa ako njegov interfejs sadrži interfejs njegovog **nadtipa**. Često kažemo da podtip *nasleđuje* interfejs svog nadtipa.

Interfejsi su od suštinskog značaja u objektno orijentisanim sistemima. Objekti se prepoznaju samo po svojim interfejsima. Nema nikakvog načina da se sazna nešto o objektu ili da se od njega bilo šta zahteva, a da se ne prođe kroz njegov interfejs. Interfejs objekta ne govori ništa o njegovoj implementaciji – različiti objekti mogu slobodno na različite načine da implementiraju zahteve. To znači da dva objekta, sa sasvim različitim implementacijama, mogu da imaju identične interfejse.

Kada se zahtev uputi objektu, izvršavanje konkretne operacije zavisi i od zahteva i od objekta. Različiti objekti koji podržavaju identične zahteve mogu imati

različite implementacije operacija koje ispunjavaju ove zahteve. Pridruživanje u vreme izvršavanja jednog zahteva jednom objektu i njegovim operacijama poznato je kao **dinamičko vezivanje**.

Dinamičko vezivanje znači da vas postavljanje zahteva ne obavezuje na određenu implementaciju sve do vremena izvršavanja. Prema tome, možete da pišete programe koji očekuju objekat sa određenim interfejsom znajući da će svaki objekat sa odgovarajućim interfejsom prihvatiti zahtev. Štaviše, dinamičko vezivanje omogućava da se objekti sa identičnim interfejsima međusobno zamenjuju u vreme izvršavanja. Ova zamenjivost poznata je kao **polimorfizam** i predstavlja jedan od ključnih koncepata objektno orijentisanih sistema. On omogućava da klijent ne mora mnogo da zna o drugim objektima osim to da podržavaju određeni interfejs. Polimorfizam pojednostavljuje definisanje klijenata, razdvaja objekte jedne od drugih i dozvoljava variranje njihovih međusobnih odnosa u vreme izvršavanja.

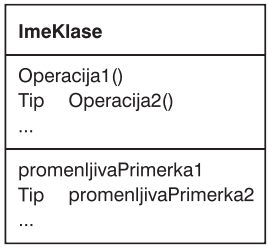
Uzorci za projektovanje pomažu definisanje interfejsa identifikacijom ključnih elemenata i vrsta podataka koji se šalju preko interfejsa. Uzorak za projektovanje može da vas uputi i u to šta *ne treba* da stavite u interfejs. Uzorak Podsetnik (283) je dobar primer. On opisuje kako treba da se enkapsulira i sačuva unutrašnje stanje objekta tako da objekat kasnije može da se vrati u isto stanje. Uzorak postavlja uslov da objekti Podsetnici moraju da definišu dva interfejsa: jedan ograničen interfejs, koji dozvoljava klijentima da zadržavaju i kopiraju podsetnike, i jedan privilegovani interfejs, koji može da koristi samo prvobitni objekat da bi sačuvao i ponovo preuzeo stanje iz podsetnika.

Uzorci za projektovanje takođe određuju odnose među interfejsima. Konkretno, oni često zahtevaju da neke klase imaju slične interfejse ili postavljaju ograničenja za interfejse nekih klasa. Na primer, uzorci Dekorater (175) i Proksi (207) zahtevaju da interfejsi objekata Dekorater i Proksi budu identični dekorisanim i kontrolisanim objektima. U uzorku Posetilac (331), interfejs Posetioca mora da oslika sve klase i objekte koje posetioci mogu da posete.

Određivanje implementacije objekata

Do sada smo malo rekli o tome kako se, u stvari, definiše objekat. Implementacija objekta se definiše njegovom **klasom**. Klasa određuje unutrašnje podatke objekta i njihovo predstavljanje i definiše operacije koje objekat može da izvrši.

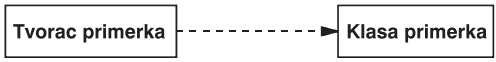
Naša notacija zasnovana na OMT-u (ukratko opisana u dodatku B) prikazuje klasu kao pravougaonik u kojem je ime klase upisano polucrnim pismom. Operacije se upisuju uobičajenim pismom ispod imena klase. Podaci koje klasa definiše dolaze posle operacija. Ime klase, operacije i podaci razdvojeni su crtama:



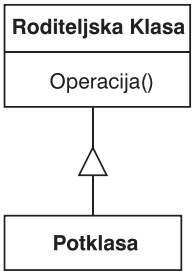
Tipovi rezultata operacija, kao i tipovi instanci promenljivih, nisu obavezni pošto ne polazimo od pretpostavke o jeziku sa statičkom implementacijom tipova.

Objekti se prave **instanciranjem** klase. Za objekat se kaže da je **instanca, odnosno primerak** klase. Proces instanciranja klase obezbeđuje memorijski prostor za unutrašnje podatke objekta (koji se sastoje od **promenljivih primerka**) i povezuje odgovarajuće operacije sa podacima. Instanciranjem klase mogu da se naprave i mnogi slični primerci objekta.

Isprekidana strelica znači da klasa instancira objekte neke druge klase. Vrh strelice okrenut je klasi napravljenih objekata.



Nove klase se mogu definisati postojećim klasama pomoću mehanizma **nasleđivanja klasa**. Kada **potklasa** nasleđuje **roditeljsku klasu,** ona sadrži definicije svih podataka i operacija definisanih u roditeljskoj klasi. Objekti koji predstavljaju primerke potklase sadržaće sve podatke definisane u potklasi i u roditeljskim klasama i moći će da izvršavaju sve operacije definisane u ovoj potklasi i njenim roditeljima. Odnos potklase označavamo vertikalnom linijom i trouglom:

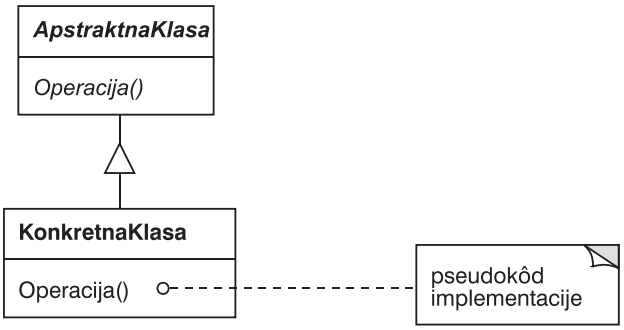


Apstraktna klasa je klasa čija je glavna namena definisanje zajedničkog interfejsa za njene potklase. Apstraktna klasa prepušta deo ili celokupnu implementaciju operacijama definisanim u potklasama; prema tome, apstraktna klase se ne može instancirati. Operacije koje apstraktna klasa deklariše, ali ne

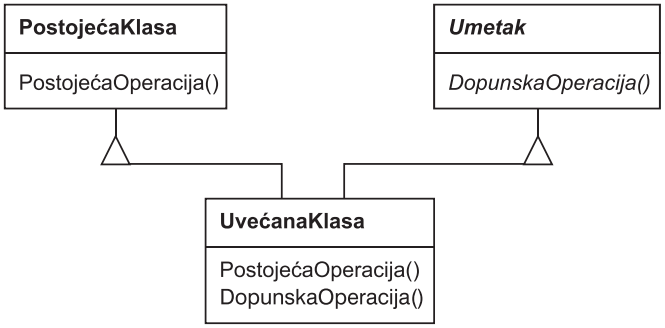
implementira, nazivaju se **apstraktnim operacijama**. Klase koje nisu apstraktne nazivaju se **konkretnim klasama**.

Potklase mogu da razrade ili redefinišu ponašanja roditeljskih klasa. Konkretnije, klasa može da **nadjača** operaciju definisanu u roditeljskoj klasi. Nadjačavanje omogućava da potklasa obradi zahtev umesto roditeljske klase. Nasleđivanje klase omogućava definisanje klasa jednostavnim proširivanjem drugih klasa čime je olakšano pravljenje familija objekata sa srodnim funkcijama.

Imena apstraktnih klasa pišu se kurzivom da bi se one razlikovale od konkretnih klasa. Kurziv se koristi i za označavanje apstraktnih operacija. Na dijagramu može da se javi i pseudokôd za implementiranje operacije; u tom slučaju se kôd upisuje u okvir sa savijenim uglom i povezuje isprekidanom linijom sa operacijom koju implementira.



Klasa umetak je klasa čija je namena da drugim klasama obezbedi opcioni interfejs ili funkcije. Slična je apstraktnoj klasi po tome što nije namenjena instanciranju. Klase umeci zahtevaju višestruko nasleđivanje:



Odnos nasleđivanja klase i nasleđivanja interfejsa

Važno je da se shvati razlika između *klase* objekta i njegovog *tipa*.

Klasa objekta definiše kako se objekat implementira. Klasa definiše unutrašnje stanje objekta i implementaciju njegovih operacija. S druge strane, tip objekta se odnosi samo na njegov interfejs – skup zahteva na koje može da odgovori. Objekat može da ima mnogo tipova, a objekti različitih klasa mogu da budu istog tipa.