

# Osnove objektnog modela

Softverski sistemi za industrijske potrebe su odavno postali toliko kompleksni da jedan čovek više nema nikakve šanse da se sa njima izbori sam. Ova kompleksnost ima nekoliko uzroka. Prvo, domen problema je uglavnom kompleksan sam po sebi i često ga je vrlo teško razumeti u meri koja garantuje njegovo uspešno modelovanje. Ozbiljan problem je i to što se eksperti iz različitih domena npr. iz domena problema i domena projektovanja softvera često ne razumeju dovoljno dobro.

Drugo, teškoće u upravljanju procesom razvoja su vrlo velike. Današnji softverski sistemi dostižu veličinu od nekoliko stotina hiljada, pa i miliona linija i mogu da imaju na hiljade modula. Zbog toga je koordinacija projektantskog tima i dobra podela posla od velike važnosti.

Treće, diskretni sistemi, a softverski sistemi su upravo takvi, ponašaju se različito od analognih sistema. Kod ovih drugih, male promene ulaza uvek proizvode male promene izlaza, dok kod diskretnih sistema mala promena ulaza može sistem odvesti u potpuno drugačiji smer, u stanje kardinalno različito od prethodnog. Kompleksni sistemi mogu imati izuzetno veliki broj stanja.

I konačno, pristup koji je korišćen u analizi i projektovanju takvih sistema, nije mogao da se nosi sa tolikom kompleksnošću. Sve je to dovelo do krize softvera, iz koje je jedini izlaz bio uspostavljanje modela sa potpuno različitim pristupom razvoju softvera. Rešenje je pronadjeno u objektnoorijentisanom modelu. Njegovi osnovni koncepti su *apstrakcija*, *enkapsulacija*, *modularnost* i *hijerarhija*.

## 1 Apstrakcija

Apstrakcija je jedan od fundamentalnih principa pomoću koga se ljudi bore sa kompleksnošću. Suština apstrakcije je u uočavanju esencijalnih karakteristika objekta koji se posmatra i njegovom razlikovanju od ostalih objekata na temelju uočenih osobina. Ona se fokusira na spoljašnji izgled objekta, zanemarujući potpuno njegovu unutrašnjost, tj. implementaciju. Naravno, osobine koje se uočavaju zavise od samog posmatrača i od njegovih nam-

era. Na primer, veterinar će posmatrajući neku životinju, recimo kućnog ljubimca, zapažati njene osobine koje su povezane sa njenim zdravljem, dok će neko drugi na toj istoj životinji posmatrati osobine koju su povezane sa njenom lepotom.

Zadatak projektanta softvera na samom početku razvoja softverskog sistema jeste da uoči i jasno definiše osnovne apstrakcije iz domena problema. Ova faza je od vitalnog značaja za naredne etape projektovanja.

U objektnim programskim jezicima (misli se na jezike kao što su Java, Smalltalk i C++), svakoj apstrakciji iz domena problema odgovara po jedna klasa. Klase su prototip iz koga se kreiraju primerci (instance). Objekti u programskom jeziku su primerci klase, i do njih se dolazi instanciranjem (stvaranjem primerka klase).

## 2 Enkapsulacija

Enkapsulacija je koncept potpuno komplementaran apstrakciji. Apstrakcija fokusira na zapažanje osobina i ponašanja objekata, a enkapsulacija se fokusira na implementaciju koja će dovesti do željenog ponašanja.

Bitan aspekt enkapsulacije je i sakrivanje nebitnih osobina klase, tj. sakrivanje strukture klase. Svaka klasa ima dva dela: interfejs i implementaciju. interfejs klase obuhvata samo spoljašnji izgled klase i preko njega se ustanovljava da li apstrakcija ima željeno ponašanje. Sa druge strane, implementacija obuhvata realizaciju svih mehanizama koji dovode do tog željenog ponašanja.

U programskim jezicima, implementacija obuhvata kreiranje klase koje su u prethodnom postupku, tj. apstrakciji, identifikovane. Svaka klasa ima svoje podatke-članove i svoje funkcije-članice (jednim imenom, to su članovi klase). iz klase se mogu, kao pomoću svojevrsnog šablona, proizvesti instance (objekti), čijom se komunikacijom ostvaruje funkcionalnost celog sistema. Sama enkapsulacija se realizuje razdvajanjem javnih i privatnih delova klase (jezici obezbeđuju jednostavne mehanizme za to). Javni delovi klase - obično su to funkcije članice, čine interfejs te klase ka okolini i mehanizam preko koga se pristupa instancama te klase. Privatni delovi (uglavnom podaci-članovi koji izražavaju stanje objekta), obuhvataju implementacione detalje koji za korisnika klase nisu važni i stoga on za njih ne mora da zna.

Ovakva podela više predstavlja pomoć u korišćenju klase nego li odbranu

od njenog neispravnog korišćenja. Jer, zlonamerni korisnik će uvek moći, uz malo domišljatosti, da dodje do privatnog dela klase i da na taj način slomi njenu enkapsulaciju.

### 3 Modularnost

Modularnost je koncept povezan isključivo sa samim programom, tj. domen problema više ne igra nikakvu ulogu, a sastoji se u formiranju modula koji mogu da se prevode odvojeno, ali koji ipak imaju dobro definisane veze prema drugim modulima. U tradicionalnom strukturiranom dizajnu, modularizacija programa se uglavnom sastojala od grupisanja potprograma dok je u objektnom dizajnu problem malo drugačiji. Ovde je, naime, prvo potrebno uočiti logičke celine koje čini izvestan broj klasa i onda formirati module. Razni jezici se vrlo različito odnose prema modulima. Moduli C++-a nisu ništa drugo do fajlovi koji se odvojeno prevode. Interfejsi modula se stavljaju u tzv. *header* fajlove (ekstenzija **h**) a njihova implementacija u odvojene (**.cpp**) fajlove. Moduli programskog jezika Ada jesu paketi (**package**) koji imaju specifikaciju i telo. Slična je situacija i sa jezikom Java, koji takodje koristi pakete, s tim što je moguće i formiranje paketa u paketu.

### 4 Hijerarhija

I na kraju, postavlja se pitanje šta raditi kada je broj apstrakcija identifikovanih u domenu problema toliki da je njima teško upravljati. Za to je potrebno novo logičko grupisanje apstrakcija - hijerarhija.

Postoje dva osnovna načina grupisanja apstrakcija. Prva je „jeste“ (*“is a” hierarchy*) ili hijerarhija klasa, a druga je „deo od“ hijerarhija (*“part of” hierarchy*) ili hijerarhija objekata.

Hijerarhija klasa se u programskim jezicima ostvaruje preko mehanizama nasleđivanja (*inheritance*), a sastoji se u tome da jedna apstrakcija može biti vrsta neke druge apstrakcije sa dodatnim osobinama. Na primer, ako smo u domenu problema identifikovali apstrakcije koje ćemo nazvati „prevozno sredstvo“, „šinsko vozilo“ i „brod“, onda o druge dve apstrakcije možemo razmišljati kao o specifičnim vrstama prve. Opravdanje za to nalazimo u činjenici da šinsko vozilo i brod jesu prevozna sredstva. Klase koje odgovaraju

ovim apstrakcijama povezane su na taj način što su klase koje odgovaraju šinskom vozilu i brodu izvedene iz klase koja odgovara prevoznom sredstvu (tj. nasledjuju njenu strukturu i ponašanje), nasledjujući tako kompletnu funkcionalnost te klase i dodajući eventualno neke svoje osobenosti koje ih čine autonomnim entitetima.

Navešćemo još jedan primer. Neka smo identifikovali sledeće apstrakcije u domenu problema: *biljka*, *cvet*, *voće*, *jabuka*. Jasno je da je cvet vrsta biljke i da njihove klase treba da budu u *is a* relaciji. Jabuka je vrsta voća, ali nije vrsta cveća. Sa druge strane, jabuka ima cvet. Stoga, klase koje odgovaraju jabuci i voću su takodje u *is a* relaciji, ali klase jabuke i cveta nisu. Ove dve klase zapravo formiraju "part of" hijerarhiju, koja se ogleda u tome da svaka instanca klase jabuke sadrži instance (po svoj prilici više njih) klase cveta. Ovaj primer je bitan zbog još jednog detalja. Naime, teško da ćemo doći u situaciju da instanciramo klasu koja odgovara biljkama generalno. Ovakva mogućnost u programskim jezicima realizuje se pomoću apstraktnih klasa – klasa koje ne mogu imati instance.

## 5 Tipiziranje

Pored ova četiri osnovna koncepta ističu se još tri koncepta objektnog modela. To su tipiziranje, konkurentnost i perzistentnost.

Koncept tipa je prilično star i poznat je još od prvih programskih jezika. Objektni jezici idu dalje tako što kod njih svaka klasa definiše novi tip pa se klase kao koncept jezika i tipovi često poistovećuju. Medjutim, to nije baš sasvim tačno. Tačno je da svaka klasa uvodi jedan novi tip čijim se instanciranjem dolazi do objekata te klase, ali problemi nastaju kada se govori o jednakosti tipova. Ako se vratimo na gornji primer sa biljkama, videćemo da svaka jabuka jeste biljka pa ćemo sa jabukom moći da uradimo sve ono što možemo sa biljkom generalno. Obrnuto ne važi, jer svaka biljka ne mora imati osobine jabuke, pa se ni tip koji uvodi klasa biljaka ne može poistovetiti sa tipom koji uvodi klasa jabuka. U obrnutom smeru ove dve klase se mogu u potpunosti izjednačiti.

## 6 Statičko vezivanje

Sa pojmom tipa je u tesnoj vezi nekoliko veoma važnih koncepata objektnog modela. To su statičko i dinamičko vezivanje, kao i polimorfizam.

Statičko vezivanje podrazumeva da objekti odgovaraju na poruke koje im se šalju shodno svojim tipovima koji su definisani u vreme kompajliranja. Dinamičko vezivanje podrazumeva da objekti odgovaraju na poruke prema svom tipu koji imaju u vreme izvršavanja programa.

Evo primera. Neka su date klase **figura**, **kvadrat** i **krug** i neka su klase **kvadrat** i **krug** izvedene iz klase **figura**. Neka klasa **figura**, kao osnovna klasa, definiše dve funkcije, recimo **centar** i **obim** i neka klase **kvadrat** i **krug** redefinišu obe ove funkcije, pri čemu je nekako naglašeno da funkcija **obim** podleže dinamičkom vezivanju (C++ za to nudi specifikator **virtual**). Posmatrajmo sada jedan deo programa:

```
figura *a=new figura();
kvadrat *b=new kvadrat();
krug *c=new krug();
figura *d=new krug();
a→ centar(); //slucaj 1
b→ centar(); //slucaj 2
c→ centar(); //slucaj 3
d→ centar(); //slucaj 4
```

Prva tri slučaja su primeri statičkog vezivanja u kojima dolazi do poziva funkcija **centar** klase **figura** i **kvadrat**, odnosno do poziva funkcije **obim** klase **krug**, respektivno. U poslednjem slučaju dolazi do poziva funkcije **obim** klase **krug**, jer se ovde radi o dinamičkom povezivanju.

## 7 Priroda objekata

Objekat bi mogao da se definiše kao entitet koji modeluje neki deo realnosti i koji ima stanje, ponašanje i identitet. Struktura i ponašanje sličnih objekata se definišu preko njegovih klasa. Stoga se izrazi instanca (primerak) klase i objekat mogu koristiti kao sinonimi.

Prilikom modelovanja realnog problema, kao kandidati za objekte nameću se ne samo opipljive ili vidljive stvari već i razne misaone konstrukcije.

Stanje objekta obuhvata sve njegove osobine kao i trenutne vrednosti svake od tih osobina. Na primer, osobina. Na primer, osobina bicikla jeste da ima mehanizam za promenu brzine. Ovo je statička osobina bicikla. Dinamička vrednost ove osobine jeste trenutna brzina u kojoj se mehanizam nalazi.

U objektnim programskim jezicima, stanje objekata izražava se preko varijabli koje definiše klasa objekta.

Svaki objekat se ponaša na izvestan način. Kada se nad njim preduzme neka akcija (u objektnoj terminologiji kaže se i „kada mu se pošalje poruka“), one reaguje promenom svog stanja. Stoga se može reći da stanje objekta predstavlja kumulativni rezultat njegovog ponašanja.

Postoji dosta izraza koje razni jezici koriste za mehanizam preduzimanja neke akcije nad objektom. Smalltalk koristi izraz prosledjivanje poruka objektu, C++ govori o pozivanju funkcija članica objekta, dok se u jeziku Java radi o pozivanju metoda objekta. Svi ovi izrazi se odnose na isto – poziv funkcije koja je definisana u klasi objekta. Svi ovi izrazi se odnose na isto – poziv funkcije koja je definisana u klasi objekta. Neće poziv svake funkcije dovesti do promene stanja objekta, jer neke funkcije samo očitavaju trenutno stanje objekta, ne menjajući ga pri tome. U skladu sa ovim, često se pravi podela metoda na:

- **modifikatore** to su metodi koji menjaju stanje objekta,
- **selektore** metodi koji pristupaju stanju objekta ali ga ne menjaju, i
- **iteratore** metodi koji pristupaju svim delovima objekta na unapred definisani način.

Posebnu važnost imaju funkcije koje se koriste za kreiranje i uništavanje objekata kada oni više nisu potrebni (konstruktori i destruktori). U njima se definišu mehanizmi koji dovode do nastanka objekta, odnosno do pravilnog oslobadjanja prostora koji je objekat zauzimao.

## 8 Veze izmedju objekata

Objekat koji je usamljen i ne saradjuje sa drugim objektima ne znači gotovo ništa. Tek kroz medjusobnu saradnju objekata postize se željena funkcional-

nost. Veze između objekata obuhvataju sve pretpostavke koje objekti čine jedan o drugom, kao i sve operacije koje pri tom mogu da izvrše. Postoje dve vrste veza između objekata. To su linkovi i agregacije.

Linkovi obuhvataju sve fizičke ili konceptualne veze između objekata. kao učesnici u linku, objekti mogu da budu:

- **aktivni objekti** tj. oni koji operišu nad drugim objektima dok se nad njima nikad ne operiše.
- **serveri** objekti koji nikad ne operišu nad drugim objektima, već se uvek operiše nad njima, i
- **agenti** objekti koji mogu da operišu nad drugim objektima ali se može operisati i nad njima.

Rečeno terminologijom programskih jezika, ove veze se ostvaruju kroz pozivanje metoda nad nekim objektom, pri čemu se kao argument tog metoda prosledjuje neki drugi objekat. Objekat koji se predaje kao argument ponaša se kao server. Objekat čiji se metod poziva igra ulogu aktivnog objekta mada može da bude i agent.

Naravno, da bi jedan objekat poslao poruku drugom, njemu ovaj drugi objekat mora da bude vidljiv. Postoji nekoliko načina na koji jedan objekat može da bude vidljiv drugom. On može da bude:

- globalan za svog klijenta,
- parametar neke od klijentovih operacija,
- deo klijentskog objekta ili
- lokalno deklarisan u nekoj od operacija klijenta

Agregacija podrazumeva da je jedan objekat deo drugog učestvujući na taj način u formiranju njegovog sveukupnog stanja. Ovakva mogućnost definiše se tako što klasa objekta sadrži podatak-član tipa klase prvog od njih. Objekat koji sadrži druge objekte često se naziva agregatom ili kontejnerom, dok se objekti koje on sadrži nazivaju njegovim atributima.

## 9 Priroda klasa

Pojmovi klase i objekta su tesno povezani. O objektima se ne može govoriti bez osvrta na njihove klase. Dok objekat predstavlja konkretan entitet koji postoji u vremenu i prostoru, *klasa predstavlja apstrakciju, tj. suštinu objekta, pa se može reći da su klase skupovi objekata koji dele zajedničku strukturu i ponašanje.*

Svaka klasa može da se posmatra kao dvokomponentna celina. Te dve njene komponente jesu njen interfejs i njena implementacija.

Interfejs klase čini njen spoljašnji izgled sakrivajući strukturu klase i sastoji se od deklaracija operacija koje mogu da se izvrše nad instancama te klase. Treba napomenuti da u interfejs klase mogu biti uključene i deklaracije drugih klasa, varijabli ili konstanti koji kompletiraju posmatranu apstrakciju. Za razliku od toga, implementacija klase jeste njen unutrašnji izgled koji obuhvata tajne njenog ponašanja.

Članovi klase formiraju interfejs klase, odnosno njenu implementaciju u skladu sa specifikatorima pristupa tim članovima. Generalno, programski jezici definišu tri specifikatora i to su javni *public*, zaštićeni *protected* i privatni *private*. specifikator. Po pravilu, ono što je javno dostupno je svakoj klasi, klasama koje su izvedene iz nje i prijateljima klase (klasa sama definiše koje klase smatra svojim prijateljima) a ono što je privatno dostupno je samo sâmoj klasi i njenim prijateljima.

## 10 Veze izmedju klasa

Veze izmedju klasa su veoma kompleksna tema koja je od velike važnosti za efikasno funkcionisanje programa. Postoji nekoliko vrsta veza, a programski jezici uglavnom obezbeđuju podršku za sledeće:

## 11 Asocijacija

Asocijacija jeste veza izmedju dve klase u kojoj jedna klasa koristi usluge druge klase na taj način što sadrži pokazivače ili reference na objekte druge klase. Veoma je važno da klasa ne sadrži same objekte druge klase već pokazivače na njih. Stoga objekti te druge klase postoje nezavisno od klase

koja ih koristi. Ukoliko klasa sadrži pokazivač na samo jedan objekat druge klase, radi se o *jedan-jedan* asocijaciji. U slučaju većeg broja pokazivača radi se o *jedan-jedan* asocijaciji, dok se u slučaju da obe klase sadrže veći broj pokazivača na objekte svake od njih radi o *više-više* asocijaciji.

Na primer, klasa koja reprezentuje prodavnicu može da sadrži pokazivače na veći broj objekata klase koja reprezentuje proizvode. Sa druge strane klasa proizvoda može da sadrži pokazivač na samo jedan objekat klase prodavnica, jer jedan proizvod može biti prodat u samo jednoj prodavnici.

## 12 Nasledjivanje

Nasledjivanje je veza izmedju klasa u kojoj jedna klasa nasledjuje strukturu i ponašanje druge klase (prosto nasledjivanje) ili više klasa (višestruko nasledjivanje). Klasa iz koje se nasledjuje se naziva superklasom ili osnovnom klasom, dok se klasa koja nasledjuje naziva izvedenom klasom ili subklasom. Dakle, može se reći da nasledjivanje definiše "is a" hijerarhiju izmedju klasa. Prilikom donošenja odluke da li klasa A treba da bude izvedena iz klase B uvek treba konsultovati pravilo: „ako A nije vrsta iz B tada A ne treba da bude izvedeno iz B“.

Izvedene klase mogu da nadgrade ili, što je redje, da redefinišu ponašanje svoje osnovne klase.

Klase koje formiraju hijerarhiju nasledjivanja formiraju strukturu stabla u kojoj su najspecifičnije klase u listovima dok se generalnije klase nalaze u čvorovima toga stabla. Klase u listovima mogu da se instanciraju za razliku od generalnih klasa kod kojih to nije uvek slučaj. Programski jezici uglavnom dopuštaju da se formiraju apstraktne klase koje obezbeđuju zajednički interfejs za skup klasa koje će iz njih biti izvedene pri čemu nema smisla da se i one same instanciraju. U takvoj situaciji je dozvoljeno kreirati pokazivače ili reference na objekte izvedenih klasa čiji će tip zapravo biti – pokazivač na osnovnu, apstraktnu klasu.

Upravo ova osobina, da pokazivači sa tipom „pokazivač na osnovnu klasu“ mogu ukazivati na objekte izvedenih klasa, omogućava jedan od najsnažnijih mehanizama objektnog modela – polimorfizam, tj. svojstvo da objekat izvršava operaciju na način svojstven izvedenoj klasi kojoj pripada, mada mu se pristupa kao objektu osnovne klase.

Pokazivaču na objekte osnovne klase se uvek može dodeliti vrednost pokazivača na objekte izvedenih klasa ali obratno nije moguće, jer će ne-minovno dovesti do gubitka informacija. Deo izvedene klase koji ne postoji u osnovnoj klasi bi nepovratno bio izgubljen pa jezici zahtevaju da programer uveri kompajler da upravo to želi. Ovo se postiže uglavnom uz pomoć tzv. *cast* operatora.

## 13 Agregacija

Agregacija je tip veze u kome jedna klasa sadrži bilo same objekte (fizičko sadržavanje, sadržavanje po vrednosti), bilo reference ili pokazivače na objekte druge klase (sadržavanje po referenci). Bez obzira o kom se tipu agregacije radi, bitno je da ove dve klase formiraju *celina/deo*. Sadržavanje po vrednosti ne može biti unakrsno tj. obe klase ne mogu sadržavati objekte tipa one druge klase, dok je u slučaju sadržavanja po referenci ova mogućnost sasvim regularna.

## 14 Korišćenje

Korišćenje označava činjenicu da jedna klasa koristi usluge druge klase i to na taj način što njeni metodi kao svoje argumente imaju objekte druge klase ili reference na njih.

*Iz knjige: Z. Despotović, „Uvod u Javu”, PC Program, Beograd, 1998*