

UML modelovanje

Autori:
Goran Savić
Milan Segedinac

1. Povezivanje klasa

Već smo videli da između klasa možemo uspostaviti različite relacije. Ovde ćemo sistematizovati tipove veza između klasa. U nastavku su objašnjeni različiti tipovi veza.

Asocijacija

Veza asocijacije je tip veze u kojem jedna klasa referencira drugu. Dakle, klasa kroz neki od atributa ima pristup objektu druge klase. Na primer, ako za grad imamo informaciju u kojoj državi se nalazi, tada ćemo u klasi Grad definisati atribut koji predstavlja referencu na objekat klase Drzava. Realizacija ovakve asocijacije u programskom jeziku Java je data u primeru.

```
public class Grad {  
    Drzava drzava;  
    ...  
}
```

Takođe, ako bismo na primer za studenta evidentirali listu ispita koje je položio, i to je veza asocijacije. Ovakva veza asocijacije je u primeru prikazana realizovana u programskom jeziku Java.

```
public class Student {  
    List<Ispit> ispiti;  
    ...  
}
```

Ako jedna klasa definiše vezom asocijacije referencu ka drugoj klasi, ne postoji prepreka da i ta druga klasa ima definisanu vezu asocijacije ka prvoj klasi. Ako samo jedna od klasa ima vezu ka drugoj klasi, onda takvu vezu asocijacije zovemo **jednosmerna asocijacija**. **Dvosmerna asocijacija** je asocijacija u kojoj obe klase ukazuju jedna na drugu. U narednom primeru je pokazano kako ranije jednosmerna asocijacija između klasa Grad i Drzava može postati dvosmerna time što bi se u klasi Drzava evidentirali gradovi koji se nalaze u određenoj državi.

```
public class Drzava {  
    List<Grad> gradovi;  
    ...  
}
```

Pomenimo i to da nema prepreke da vezom asocijacije klasa uspostavi vezu ka samoj sebi. Ovo zovemo **refleksivna asocijacija**. Tada klasa kao atribut sadrži svoje instance. Pogledajmo primer refleksivne asocijacije u kojoj osoba ima informaciju o svom ocu, ali i deci.

```
public class Osoba {  
    Osoba otac;  
    List<Osoba> deca;  
    ...  
}
```

I kod refleksivne asocijacije možemo da govorimo o smeru asocijacije. U prethodno prikazanom primeru asocijacija je dvosmerna, jer asocijacija predstavlja vezu roditelj-dete između osoba.

UML modelovanje

Obzirom da se za svaku osobu evidentira i njen otac i njena deca, asocijacija je dvosmerna. Ako bismo jedan od ova dva atributa uklonili, asocijacija bi postala jednosmerna.

Iako su u prethodnim primerima sve prikazane veze bile veze asocijacije, možemo da primetimo da postoji razlika između njih. U prvom primeru, grad ukazuje na jednu državu u kojoj se nalazi. U naredna dva primera, prikazane klase ukazuju na više objekata, pa su predstavljeni listom (u prvom slučaju su to objekti klase Ispit, a u drugom klase Grad). Gledano iz perspektive klase koja sadrži vezu asocijacije, broj objekata druge klase koji učestvuju u vezi asocijacije naziva se **kardinalitet** (eng. *cardinality*). Obzirom da postoji minimalni i maksimalni mogući broj objekata u vezi, ove dve granične vrednosti nazivamo donji, odnosno gornji kardinalitet. Jednim imenom donji i gornji kardinalitet nazivamo mnogostrukost (eng. *multiplicity*).

Na primer, ako grad ukazuje na državu i ako može postojati grad za koji nije definisana država, tada je donji kardinalitet ove veze asocijacije 0. Na nivou implementacije u Javi, to bi značilo da odgovarajući atribut ima vrednost null. Gornji kardinalitet ove veze je 1, jer grad u određenom trenutku može da ukazuje na najviše jednu državu u kojoj se nalazi. Na nivou programskog koda, u odgovarajući atribut bi bila upisana referenca na instancu klase Drzava. Slično, ako posmatramo vezu asocijacije koja predstavlja informaciju da student ima listu položenih ispita, tada je donji kardinalitet 0 (jer student može imati 0 položenih ispita), a gornji kardinalitet nije ograničen (može imati više položenih ispita). U tabeli su navedene moguće vrednosti za mnogostrukost u vezi asocijacije. Mnogostrukost je označena vrednošću donjeg i gornjeg kardinaliteta.

| Mnogostrukost | | Opis |
|--------------------|---------------------|---------------------------|
| Donji kardinalitet | Gornji kardinalitet | |
| 0 | 1 | Nijedan ili jedan objekat |
| 1 | 1 | Tačno jedan objekat |
| 0 | * | Nijedan ili više objekata |
| 1 | * | Jedan ili više objekata |

Asocijacija je generalni pojam kojim opisujemo svaku vezu u kojoj atribut klase referencira druge objekte. Postoje i podtipovi asocijacije kojima imenujemo specifične varijante veze asocijacije. Ovi podtipovi su veza **agregacije** i veza **kompozicije**. I agregaciju i kompoziciju koristimo kada želimo da predstavimo vezu između neke celine i njenog dela. Ipak, postoji razlika u pogledu scenarija korišćenja ove dve veze.

Veza agregacije između dve klase se koristi kada entitet koji je predstavljen jednom klasom sadrži entitet (ili entitete) predstavljene drugom klasom. Na primer, jedno školsko odeljenje sadrži učenike koji školu pohađaju u tom odeljenju.

Sa druge strane, kompozicija se takođe koristi kada jedan entitet sadrži drugi, ali je veza između njih čvršća u odnosu na agregaciju. Naime, kompozicija se koristi kada od celine zavisi i životni vek dela. Kod kompozicije, celina je zadužena za kreiranje i uništavanje dela. Takođe, deo ne može da postoji nezavisno od celine. Primer veze kompozicije je veza između studenta kao celine i položenih ispita kao jednog dela evidencije o studentu. Evidencija liste studentovih položenih ispita je potpuno vezana za evidenciju tog studenta i ne može da postoji nezavisno od postojanja studenta u evidenciji.

UML modelovanje

Razlika između podtipova asocijacije postoji samo na nivou značenja veze kako bi model sistema bio što ekspresivniji. Na nivou Java programskog koda ove razlike ne postoje i svaka veza asocijacije se predstavlja kao referenca na drugu klasu.

Zavisnost

Veza tipa zavisnost označava da implementacija jedne klase zavisi od druge klase. Dakle, ako jedna klasa u bilo kojem kontekstu koristi drugu klasu (instancira drugu klasu, poziva njene metode, pristupa njenim atributima, dobija njenu instancu kao parametar metode), tada postoji veza zavisnosti između ove dve klase. To znači da promena na drugoj klasi utiče na rad prve klase koja od nje zavisi. Ovo je manje čvrsta veza od asocijacije koja je zahtevala da klasa ima referencu na drugu klasu u vidu svog atributa. Dat je primer veze zavisnosti između klase Krug i klase Math, koju klasa Krug koristi da bi izračunala obim kruga.

```
public class Krug {
    private double r;
    public double getObim() {
        return 2 * 2 * Math.PI;
    }
}
```

Generalizacija

Veza **generalizacije** između dve klase je uspostavljena kada jedna klasa nasleđuje drugu. Dakle, klasa koja je podtip je **specijalizacija** klase koja je nadtip. Obzirom da je ranije bilo dosta reči o nasleđivanju, ovde ćemo se samo podsetiti Java programskog koda kojim definišemo ovu vezu. Primer ilustruje vezu nasleđivanja između generalnog entiteta računar i prenosivog računara kao specifične realizacije ovog entiteta.

```
public class PrenosiviRacunar extends Racunar {
    ...
}
```

Realizacija

Ako imamo hijerarhiju tipova, nadtip može da bude i interfejs. Tada za podtip kažemo da implementira interfejs. Veza u kojoj klasa implementira određeni interfejs se naziva veza realizacije. Dakle, podtip je realizacija ponašanja definisanog interfejsom koji mu je nadtip. U narednom primeru ćemo se podsetiti Java sintakse za definisanje veze realizacije. Primer ilustruje realizaciju poređenja studenta sa drugim objektima, što je ponašanje definisano interfejsom Comparable.

```
public class Student implements Comparable<Student> {
    ...
}
```

2. UML

Veoma je važno da, osim implementacije sistema predstavljene nekim programskim jezikom, bude predstavljen i dizajn sistema u nekoj preglednijoj notaciji. Radi lakšeg razumevanja prirode sistema, obično se u formi dijagrama ilustruju njegovi različiti aspekti. Kreiranje ovih dijagrama predstavlja

UML modelovanje

deo procesa dizajna sistema koji se vrši pre same implementacije. Takođe, ovakvi dijagrami su važan deo dokumentacije sistema.

UML (Unified Modeling Language) predstavlja grafički jezik za predstavljanje dizajna sistema. Ovaj grafički jezik je generalne namene, tako da nije ograničen na neki određen tip sistema ili programski jezik. UML je široko prihvaćen formalizam ove namene, pa ćemo se ovde upoznati sa njegovim najznačajnijim konstruktima. Obzirom da se putem UML notacije mogu predstaviti različiti aspekti sistema, UML specificira različite tipove dijagrama, koje možemo svrstati u dve grupe.

Prva grupa dijagrama opisuje strukturu sistema. Najčešće korišćeni tipovi dijagrami koji pripadaju ovoj grupi su:

- dijagram klasa – prikazuje klase u sistemu, njihove atribute i metode, kao i veze između klasa
- dijagram komponenti – prikazuje od kojih se komponenti sistem sastoji i u kakvoj su vezi komponente međusobno
- dijagram paketa – prikazuje pakete u sistemu i veze između njih
- *deployment* dijagram – prikazuje infrastrukturu za postavljanje sistema u rad. Ovo uključuje hardversku infrastrukturu, kao i softverske sisteme koji se izvršavaju na specificiranom hardveru

U drugu grupu spadaju dijagrami koji opisuju ponašanje sistema. Iz ove grupe se najčešće koriste:

- dijagram slučajeva korišćenja (eng. *use case diagram*) – opisuje scenario korišćenja sistema kroz specifikaciju akcija koje sistem može da izvrši na zahtev korisnika ili drugih sistema
- dijagram aktivnosti – prikazuje tok aktivnosti pri izvršavanju određene akcije
- dijagram sekvence – prikazuje vremensku sekvencu operacija koje komponente sistema izvode pri međusobnoj interakciji
- dijagram stanja – prikazuje stanja u kojima komponente sisteme mogu da budu i akcije kojima se prelazi iz jednog stanja u drugo

U nastavku ćemo detaljnije prikazati dijagram klasa i dijagram slučajeva korišćenja.

Dijagram klasa

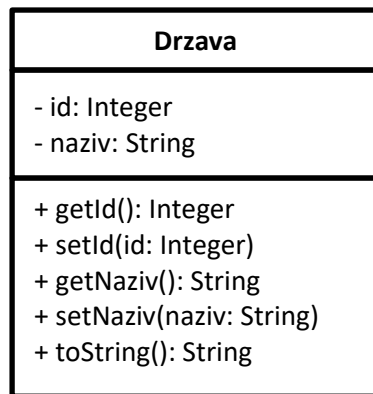
Kao što smo naveli, dijagram klasa prikazuje klase sistema. Klasa se grafički predstavlja kao pravougaonik koji je podeljen u tri dela koji redom prikazuju naziv klase, atribut klase i metode klase. Za atribut i metode se posebnim znakovima označavaju i modifikatori pristupa i to prema notaciji prikazanoj u tabeli. Tabela prikazuje samo modifikatore pristupa koji postoje u programskom jeziku Java.

| Oznaka | Modifikator pristupa |
|--------|----------------------|
| + | public |
| - | private |
| # | protected |

Pored toga, označava se i da li član klase (atribut ili metoda) pripada instanci klase ili samoj klasi. Dakle, za programski jezik Java ova oznaka specificira da li je član klase statički. Ako član klase pripada klasi (tj. ako je statički) njegov naziv se navodi podvučen.







UML modelovanje

Sledeći primer prikazuje ranije pominjanu klasu Drzava u UML notaciji.



Vidimo da notacija predviđa navođenje i tipova podataka za atribute, kao i za parametre i povratne vrednosti metoda. Kao tip se koristi naziv klase koja predstavlja tip ili neki od UML ugrađenih tipova (kao što su Integer i String u prikazanom primeru).

Osim samih klasa, dijagram klasa prikazuje i veze između klasa. Ranije smo pominjali da postoje različiti tipovi veza između klasa. Svaka veza se označava linijom koja povezuje dve klase. Znak kojim linija počinje ili završava ukazuje na tip veze i smer veze. UML koristi sledeće oznake za tipove veza između klasa:

- asocijacija 
- agregacija 
- kompozicija 
- zavisnost 
- generalizacija 
- realizacija 

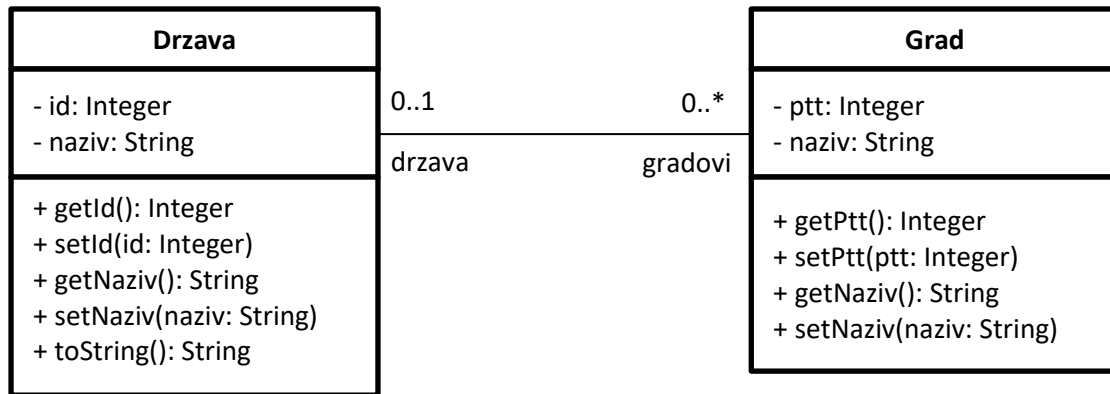
Prikazana notacija prikazuje jednosmernu asocijaciju, agregaciju, odnosno kompoziciju. Ovo je označeno time što linija završava strelicom. Ako je reč o dvosmernoj asocijaciji, agregaciji ili kompoziciji, tada se ne stavlja strelica na kraju linije.

Linija za asocijaciju se obično dodatno označava labelama koje se postavljaju iznad ili ispod linije. Ove labele označavaju sledeće osobine veze:

- naziv veze
- mnogostrukost - označava se navođenjem vrednosti donjeg i gornjeg kardinaliteta koji su razdvojeni sa dve tačke
- uloge – za obe klase koje su u vezi se može označiti naziv uloge u vezi. To je praktično naziv atributa koji u klasi odgovara toj vezi
- modifikatori pristupa – za obe uloge se može označiti modifikator pristupa. To je praktično modifikator pristupa atributa koji u klasi odgovara toj vezi

Pogledajmo primer ranije prikazane dvosmerne veze asocijacije između grada i države.

UML modelovanje



Vidimo da su za prikazanu vezu asocijacije navedene uloge klasa, kao i mnogostrukost.

Pomenimo još da UML omogućuje precizniji opis veze zavisnosti time što ima predefinisani vokabular najčešćih tipova zavisnosti. Neki od vrednosti iz vokabulara su:

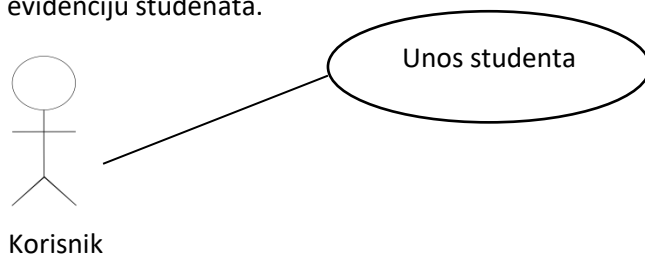
- *call* – klasa poziva drugu klasu
- *use* – klasa na neki način koristi drugu klasu
- *instantiate* – klasa instancira objekte druge klase

Naziv tipa zavisnosti se onda na dijagramu postavlja kao labela kod linije koja označava vezu zavisnosti.

Dijagram slučajeja korišćenja

Dijagram slučajeja korišćenja opisuje akcije koje mogu biti izvršene nad sistemom. Ovaj tip dijagrama se često koristi za formalnu specifikaciju funkcionalnosti sistema obzirom da specificira šta sistem može da uradi.

Dijagrami slučajeja korišćenja su namenjeni razumevanju i od strane korisnika bez posebnog tehničkog znanja. Sadrže dva osnovna elementa. To su učesnik (eng. *actor*) i slučaj korišćenja. Učesnik je bilo koji korisnik sistema. To može da bude čovek, ali i drugi sistem. Takođe, možemo učesnike razvrstati po kategorijama, pa da na primer administratora i običnog korisnika posmatramo kao dva različita učesnika. Slučaj korišćenja opisuje samu akciju koju sistem sprovodi, pri čemu se na dijagramu navodi naziv slučaja korišćenja. Dat je primer jednog slučaja korišćenja kroz koji se vidi grafička notacija koju koriste dijagrami slučajeja korišćenja. Primer ilustruje akciju u sistemu za evidenciju studenata.

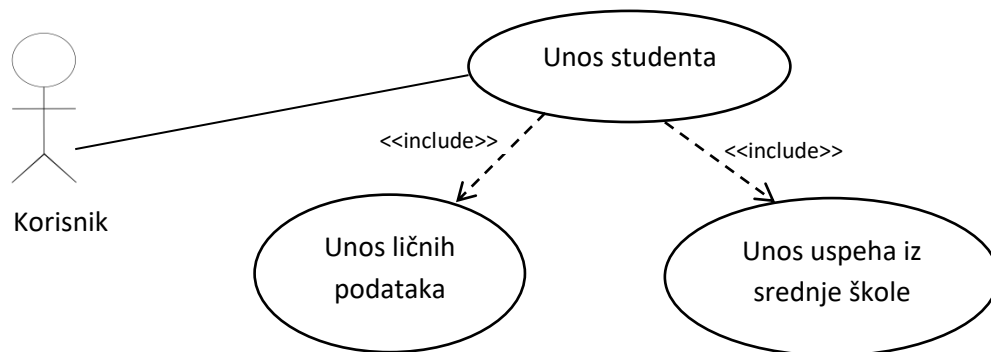


Vidimo da ovaj slučaj koristi učesnik označen nazivom **Korisnik**. Slučaj korišćenja se odnosi na unos studenta.

UML modelovanje

I kod dijagrama slučajeva korišćenja je moguće uspostaviti veze između slučajeva korišćenja. Konkretno, to su veze pod nazivom *uključuje* (eng. *includes*) i *proširuje* (eng. *extends*).

Veza tipa *uključuje* između dva slučaja korišćenja se koristi kada se pri izvršavanju jednog slučaja korišćenja uvek izvršava i drugi slučaj korišćenja. Na primer, unos studenta uključuje unos ličnih podataka i unos uspeha iz srednje škole. Ovaj primer je i ilustrovan narednim dijagramom na kojem se vidi i notacija za definisanje veze tipa *uključuje*.



Veza tipa *proširuje* se koristi kada pri izvršavanju jednog slučaja korišćenja može opciono da nastupi izvršavanje drugog slučaja korišćenja. Na primer, pri unosu studenta može se pojaviti potreba da se u evidenciju gradova doda grad iz kojeg je student ako se ovaj grad trenutno ne nalazi u evidenciji. Ovaj primer zajedno sa notacijom prikaza veze *proširuje* je prikazan na dijagramu. Treba obratiti pažnju na usmerenost veze *proširuje*. Veza je usmerena ka slučaju korišćenja čije se izvršavanje proširuje.

