Razvoj softvera

dr.sc. Emir Mešković

VIII predavanje

Sadržaj predavanja

- UML dijagram objekata i klasa
- UML dijagram sekvenci

Općenito o analizi

- Analiza je temeljna i strateška aktivnost unutar UP. Ona se bavi stvaranjem modela koji ocrtava osnovna svojstva traženog sistema u skladu s postavljenim zahtjevima.
- Glavni rezultati koje analiza treba stvoriti su:
  - Klase na nivou analize (*analysis classes*).
  - Grubo se opisuju klase koje modeliraju ključne koncepte unutar date poslovne domene.
  - Realizacija slučajeva upotrebe. Pokazuje se kako se ponašanje sistema definisano use case-ovima može realizovati interakcijom objekata iz opisanih klasa.
- Daljnji rezultati analize su:
  - Opis veza među klasama.
  - Grupisanje klasa u pakete.
  - Modeliranje složenijih procesa iz date poslovne domene.

Odnos analize i oblikovanja

- UP ne postavlja jasnu granicu između analize i oblikovanja. Teško je odrediti gdje prestaje analiza a počinje oblikovanje.
- Sličnosti između analize i oblikovanja.
  - Obje aktivnosti bave se modeliranjem.
  - Obje aktivnosti služe se istim vrstama UML dijagrama.
  - Model stvoren u analizi dalje se profinjuje tokom oblikovanja.
- Razlike između analize i oblikovanja.
  - Model na nivou analize je jednostavan i daje globalnu sliku sistema.
  - Model na nivou oblikovanja je znatno komplikovaniji jer sadrži sve detalje koji su potrebni za implementaciju.
  - Model na nivou analize prikazuje sam problem (poslovni sistem) u terminima date poslovne domene. Razumljiv je korisnicima.
  - Model na nivou oblikovanja prikazuje rješenje problema (softverski sistem) i služi se informatičkim pojmovima. Namijenjen je softverskim inženjerima.

Svojstva modela na nivou analize

- Izražen je poslovnim jezikom.
- Opisuje predmete, osobe ili pojave koje modeliraju poslovnu domenu.
- Na primjer, ako je riječ o sistemu elektronske trgovine, tada se pojavljuju klase poput Kupac, Narudžba, Korpa, a ne pojavljuju se klasa za pristup bazi podataka ili klasa za mrežnu komunikaciju.
- Ocrtava se globalna slika sistema, na što jednostavniji način.
- Sadrži modele koji “pričaju priču”. Svaki dijagram ističe neki važan aspekt ponašanja sistema.
- Razumljiv je i korisnicima i softverskim inženjerima.

Općenito o klasama

- **Definicija:** Klasa je nacrt (prototip) koji definira zajedničke atribute i operacije svim objektima određene vrste (tipa)
- Klasa je kalup na osnovu kojeg se može napraviti proizvoljan broj objekata sa istim osobinama i ponašanjem
- Klasa uključuje deklaracije svih atributa i operacija koje su pridužene objektima te klase
- Isti skup objekata obično se može klasificirati na razne načine. Pronalaženje dobre klasifikacije je ključ za uspješnu analizu.

Općenito o objektima

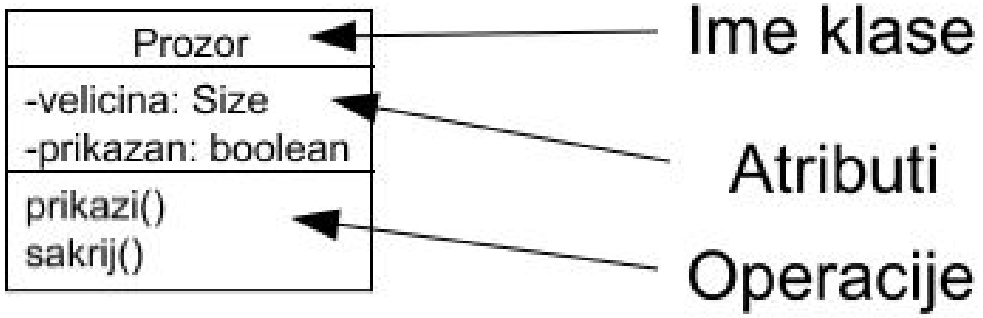
- **Definicija:** Objekat je entitet koji ima stanje i definirani skup operacija koji djeluju na tom stanju
- Stanje je predstavljeno kao skup atributa objekta
- Operacije pružaju usluge drugim objektima (klijentima) koji traže neke usluge obrade (prenos parametara).
- Do vrijednosti atributa obično se može doći jedino tako da se pozove operacija koja vraća tu vrijednost. To se zove *učahurivanje (enkapsulacija).*
- Svaki objekat je instanca klase koja definiše zajednička svojstva (atribute i operacije) za skup sličnih objekata

Općenito o objektima

- Svaki objekat ima sljedeća važna svojstva.
- Identitet.
  - Ono što ga čini drukčijim i jedinstvenim u odnosu na druge objekte.
- Stanje.
  - Određeno je vrijednostima atributa i vezama s drugim objektima. Može se mijenjati od trenutka do trenutka.
- Ponašanje.
  - Određeno je njegovim operacijama. Objekat može raditi ono i samo ono što je sadržano u tim operacijama. Poziv operacije može izazvati promjenu vrijednosti nekog od atributa ili uspostaviti odnosno pokidati veze s drugim objektima. Tako i samo tako objekat mijenja stanje.

Dijagram klasa

- Prikazuje tipove objekata koji se koriste u sistemu skupa sa njihovim atributima i operacijama
- Prikazuje i veze između tipova
  - Ovisnost, asocijacija, generalizacija...
- Koristi se u različitim fazama razvoja sistema sa različitim nivoima detalja
- Klasa se u dijagramima predstavlja kao pravougaonik



Atributi

- **Atribut** je osobina klase**.**
- Atributi se mogu se posmatrati kao članovi klase
- Zapis atributa
  - **vidljivost** ime: tip [**višestrukost**] = **default** {ograničenja}

- Objašnjenje:
  - vidljivost
    - - private; + public; # protected; ~ package(namespace)
  - višestrukost
    - Podrazumjevana je 1; 0..1 nula ili jedna; * proizvoljno
  - default
    - Početna vrijednost za član prilikom kreiranja instace klase

Vidljivost

- Vidljivost se osim na atribute primjenjuje i na operacije. Riječ je o oznaci tko može pristupiti dotičnom atributu odnosno relaciji, prema tabeli:

Oznaka	Vidljivost	Značenje
+	Javna	Bilo koji element koji može pristupiti klasi može pristupiti i bilo kojoj njenoj osobini koja ima javnu vidljivost
-	Privatna	Samo operacije unutar klase mogu pristupiti osobinama koje imaju privatnu vidljivost
#	Zaštićena	Samo operacije unutar klase ili operacije unutar podklasa date klase mogu pristupiti osobinama koje imaju zaštićenu vidljivost
~	Paketska	Bilo koji elemenat koji je u istom paketu kao klasa, ili u ugniježdenom podpaketu, može pristupiti bilo kojoj osobini koja ima paketsku vidljivost

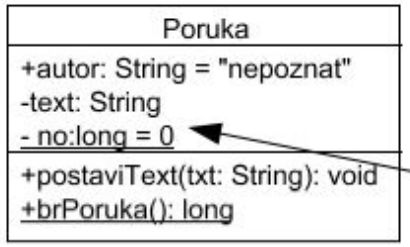
Operacije

- Operacija predstavlja nešto što klasa može raditi.
- Operacija se može posmatrati kao metod klase
- Zapis operacije
  - **vidljivost** ime(smjer naziv_parametara : tip_parametra = podrazumjevana vrijednost): povratni_tip
- Smjer određuje da li je riječ o ulaznom ili izlaznom parametru, može biti *in*, *out*, *inout*, *return*. Default je *in*. Smjer *return* služi za operacije s više povratnih vrijednosti.
- Default vrijednost parametra se podrazumijeva ako u pozivu operacije nije bila zadana nikakva vrijednost.

Doseg na nivou objekta ili klase

- Uobičajeno je da objekti imaju svoje kopije atributa. Također, operacije obično djeluju na pojedinim objektima. Kažemo da ti atributi odnosno operacije imaju *doseg na nivou objekta (instance scope).*
- No ponekad nam je potreban atribut koji ima jedinstvenu (zajedničku) vrijednost za sve objekte iz klase. Također, postoje operacije (na primjer instancijacija) koje se primjenjuju na samu klasu a ne na njene objekte. Tada kažemo da ti atributi odnosno operacije imaju *doseg na nivou klase (class scope).*
- Vrijede sljedeća pravila za pristup.
  - Operacija s dosegom na nivou objekta može pristupiti bilo kojoj drugoj operaciji ili atributu.
  - Operacija s dosegom na nivou klase može pristupiti samo operacijama i atributima koje su također s dosegom na nivou klase.

Primjer

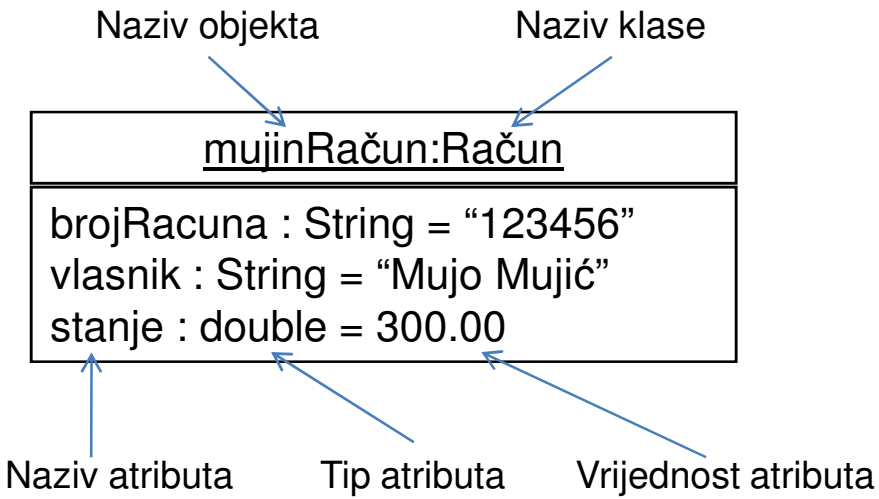


Statički metodi i članovi označavaju se podvučenom linijom

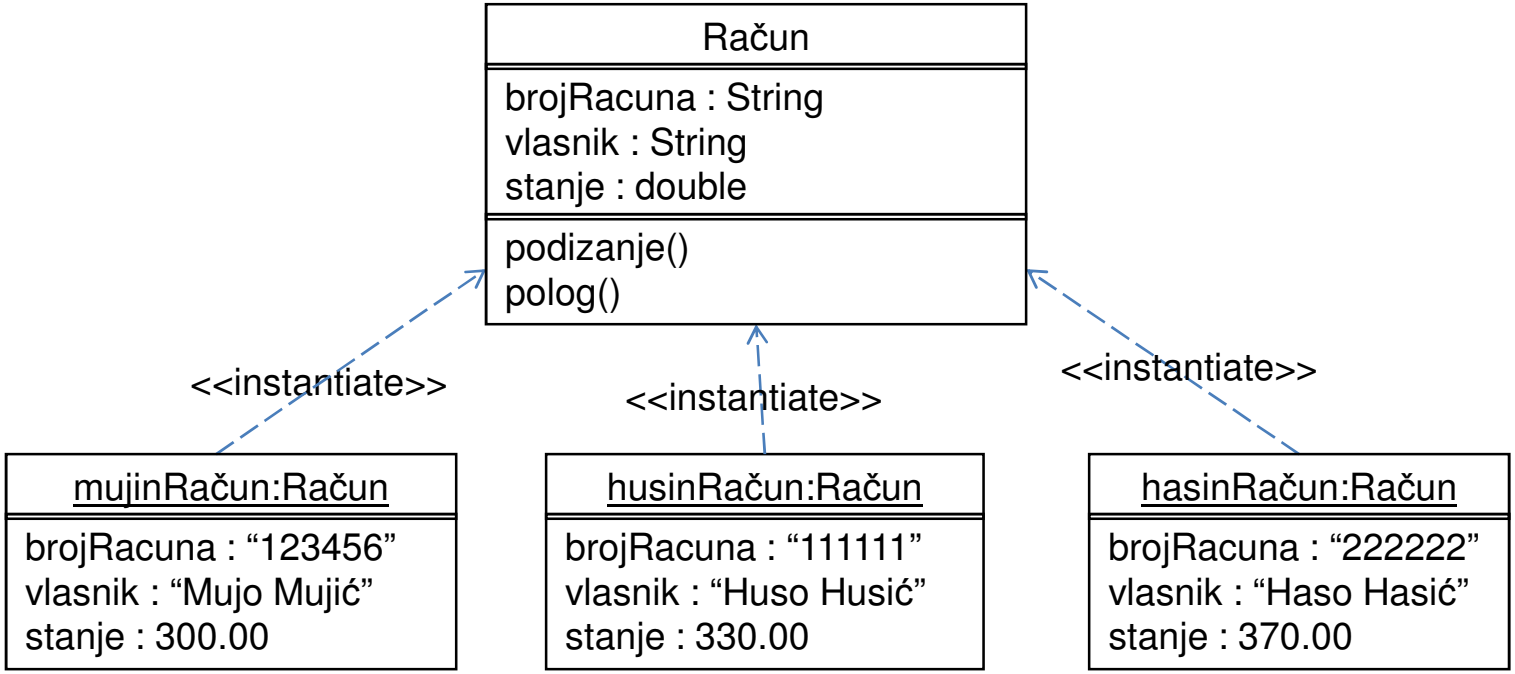
```
class Poruka
{
    public String autor = "nepoznat";
    private String text;
    private static long no = 0;
    public void postaviText(String txt)
    {
        text = txt;
    }
    public static long brPoruka()
    {
        return no;
    }
}
```

Prikaz objekata u UML-u

- Objekt se crta kao pravougaonik s dva odjeljka.
  - Gornji odjeljak sadrži ime objekta i/ili ime pripadne klase (podvlačenje je važno jer se na taj način vidi da pravougaonik prikazuje objekt, a ne klasu).
  - Donji odjeljak sadrži imena i vrijednosti atributa. Dozvoljeno je da neki ili čak svi atributi budu ispušteni. Također se mogu ispustiti tipovi



Odnos objekata i klasa



- Odnos se vidi na slici. Strelice znače da su objekti dobiveni instancijacijom (stvaranjem) iz klase.
- Općenito, crtkana strelica u UML-u označava *ovisnost (dependency).* Element iz kojeg strelica kreće zove se *klijent (client),* a element s druge strane zove se *dobavljač (supplier).* Strelica znači da klijent ovisi o dobavljaču.

Odnos objekata i klasa

- U našem primjeru, ovisnost je dobila posebno značenje zbog korištenja stereotipa <<instantiate>>. Taj stereotip pretvara običnu ovisnost u ovisnost *instancijacije.*
- Stereotipi predstavljaju općeniti mehanizam kojim se sintaksa i semantika UML-a može prilagoditi potrebama. Stereotip se primjenjuje na neki od postojećih elemenata za modeliranje i njime se stvara nova varijanta tog elementa s novom semantikom. Stereotip prepoznajemo po <<…>>.
- U programskim jezicima instancijacija objekta iz klase ostvaruje se *konstruktorom.* Riječ je o posebnoj operaciji koja djeluje na klasu a ne na objekat. Obično postoje i obratne operacije *destruktori.*

Kako treba izgledati klasa na nivou analize?

- Ona modelira jedan specifični element poslovne domene.
- Njeno *ime* jasno odražava njen smisao.
- Ona sadrži svega *nekoliko najvažnijih atributa.* Tipovi tih atributa ne moraju biti određeni.
- Ona sadrži svega nekoliko *odgovornosti (responsibilities).* Odgovornost je skup srodnih operacija, bilježimo je kao da je riječ o jednoj operaciji, ne navodimo parametre.
- Između njenih atributa i odgovornosti postoji *jaka kohezija (high cohesion).*
- Između nje i drugih klasa postoji slaba *povezanost (low coupling).*

Kako pronaći klase na nivou analize

- Analiza imenica i glagola.
  - Čitaju se zahtjevi, *use case-ovi, projektni pojmovnik...*
  - U tim tekstovima se pronalaze imenice i glagoli.
  - Imenice su kandidati za klase ili atribute.
  - Glagoli su kandidati za odgovornosti .
  - Analiziramo skupljene kandidate te odlučujemo koji od njih će zaista postati klase, atributi, odnosno odgovornosti.
  - Također odlučujemo kako ćemo atribute i odgovornosti podijeliti po klasama (tako da postignemo visoku koheziju unutar klasa, te slabu povezanost između klasa).

Kako pronaći klase na nivou analize

- Razmatranje drugih izvora klasa.
  - Još jednom posmatramo stvarni svijet. Uočavamo fizičke predmete, osobe, dokumente, interfejse.
  - Proučavamo objavljena rješenja drugih analitičara koja se odnose na standardne poslovne sisteme (*archetype patterns*).

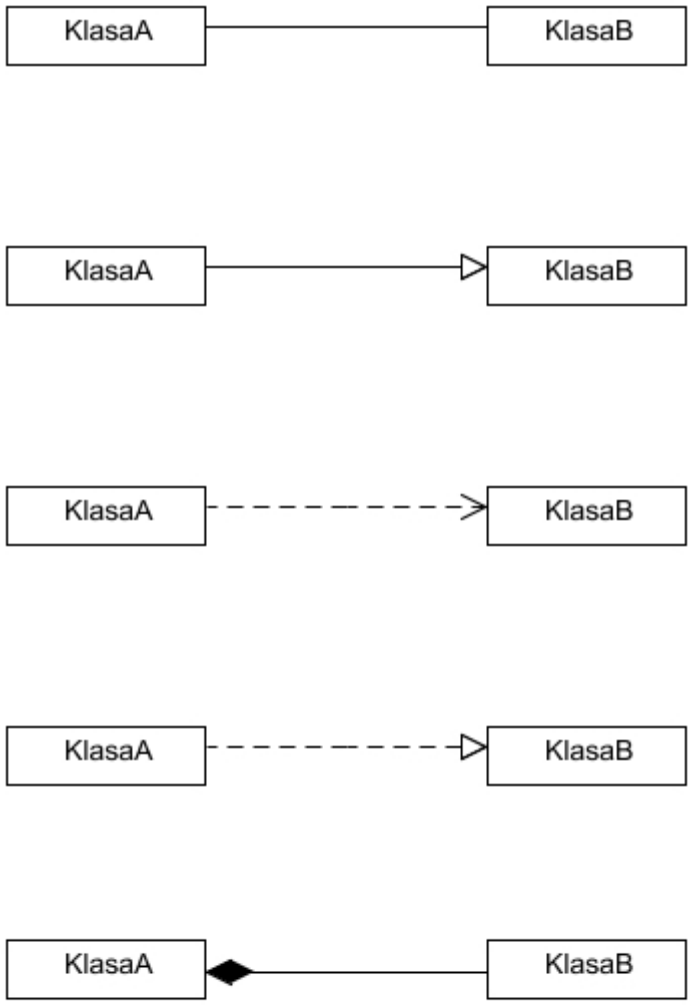
Kako pronaći klase na nivou analize

- CRC analiza
  - Timski rad, odvija se na sastanku, tim se sastoji od analitičara i korisnika.
  - Stvari koje su važne za problemsku domenu zapisuju se na ljepljivim papirićima.
  - Svaki papirić ima tri odjeljka (C-R-C):
    - *Class* – upisuje se ime klase
    - *Responsibilities* – upisuje se lista odgovornosti za klase
    - *Collaborators* – upisuje se lista drugih klasa s kojima dotična klasa sarađuje.

Kako pronaći klase na nivou analize

- Ako smo koristili više metoda za pronalaženje klasa, tada je na kraju potrebna *konsolidacija rezultata.*
  - Uspoređujemo rezultate dobivene pomoću različitih metoda.
  - Razrješavamo sinonime i homonime.
  - Posebnu pažnju posvećujemo mjestima gdje postoje razlike u rezultatima.
  - Odlučujemo se za najbolju ili kombinovanu verziju klasa, te tako dobivamo početni popis klasa na nivou analize.
  - Taj popis će se vjerojatno i dalje dotjerivati.

Relacije među klasama i objektima



- Asocijacija
  - relacija "ima"
- Generalizacija
  - relacija "je"
- Ovisnost
  - relacija "koristi"
- Implementacija
  - relacija "implementira"
- Kompozicija
  - relacija "sastoji se od"

Vrste veza

- *Poveznica (link)* je veza između objekata koja nastaje kada jedan objekat ima referencu (pointer) na drugi objekat
- *Asocijacija (association)* je veza između klasa koja kaže da između objekata dotičnih klasa postoje poveznice
- *Ovisnost (dependency)* između klijenta i dobavljača znači da klijent na neki način ovisi o dobavljaču.
- Ge*neralizacija (*g*eneralization)* je vrsta ovisnosti koja se uspostavlja između podklase i nadklase.
- *Agregacija (aggregation)* i *kompozicija (composition)* su vrste asocijacije koje kažu da objekti jedne klase predstavljaju cjelinu, a objekti druge klase se pojavljuju kao dijelovi u toj cjelini.

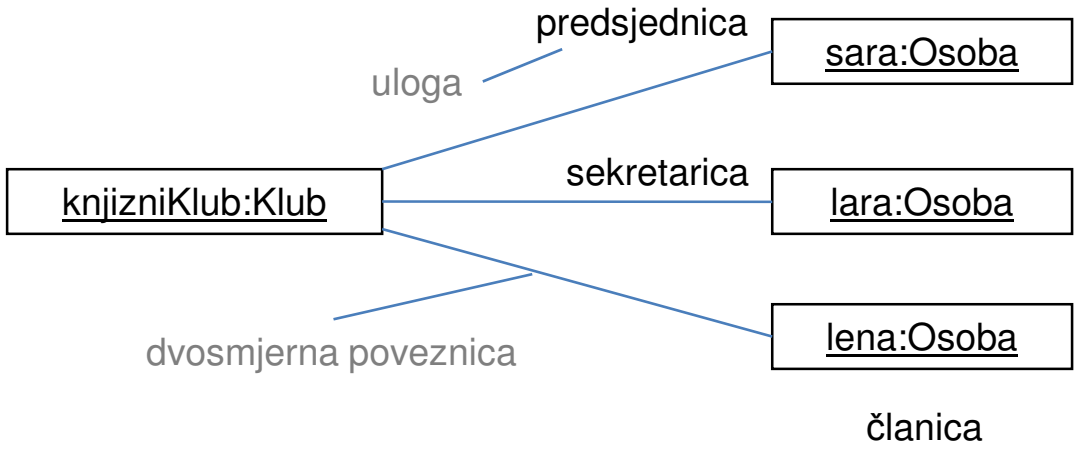
Poveznice među objektima

- *Poveznica(link)* nastaje onda kad jedan objekat ima referencu (*pointer*) na drugi objekat.
- Poveznica omogućuje saradnju objekata:
  - jedan objekat šalje poruku drugom objektu,
  - taj drugi objekat reaguje na poruku tako da izvrši odgovarajuću operaciju.
- Poveznica može biti dvosmjerna i jednosmjerna, ovisno o tome da li oba objekta mogu slati poruku drugom, ili to može raditi samo jedan od njih.
- Jedna poveznica obično povezuje tačno dva objekta. Postoje (ali rijetko se koriste) višestruke poveznice koje povezuju tri ili više objekata.

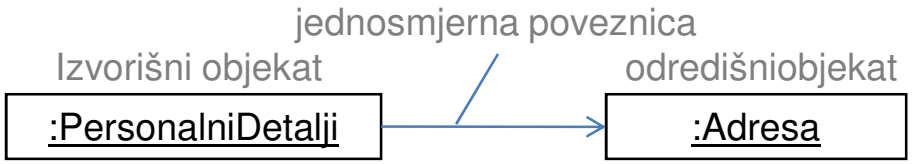
Poveznice među objektima

- Jednostruka dvosmjerna poveznica se na dijagramu se crta kao spojnica, a višestruka kao romb sa zrakastim spojnicama. Jednostruka jednosmjerna spojnica crta se kao strelica.
- Uz spojnicu mogu biti označene uloge koje objekti igraju jedan prema drugom.
- U objektnom sistemu objekti mogu nastajati i nestajati, a isto tako i poveznice među njima
- Zato dijagram treba shvaćati kao “fotografiju” koja bilježi stanje sistema u jednom trenutku.

Poveznice među objektima

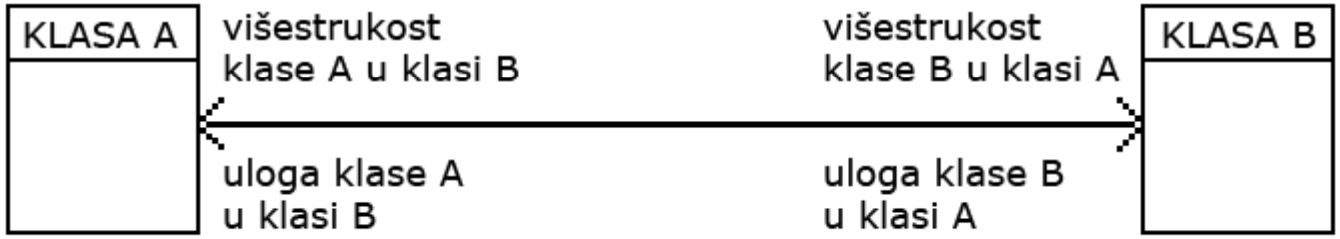


- Na dijagramu je prikazan klub i njegovi članovi. Trenutno sara igra ulogu predsjednice, lara je sekretarica itd. Ako se kasnije te uloge promijene, morali bi crtati novi dijagram.
- Na sljedećem dijagramu komunikacija između objekata dozvoljena je samo u jednom smjeru



Asocijacija

- Asocijacija predstavlja drugi način zapisivanja osobina klasa
- Omogućava predstavljanje uloga koje klasa ima u drugim klasama
- Asocijacija je veza između klasa koja kaže da između objekata tih klasa mogu postojati poveznice



- Uloge klasa mogu biti izostavljene sa dijagrama
- Višestrukost označava broj objekata klase koji učestvuju u vezi
- Strelice na liniji nam pokazuju smijer asocijacije
  - Jednosmjerna asocijacija
  - Dvosmjerna asocijacija
- Refleksivna asocijacija kaže da objekti iz dotične klase mogu imati poveznice prema drugim objektima iz iste klase.

Višestrukost veze

- *Višestrukost* se odnosi na jednu od klasa u asocijaciji. On određuje broj objekata te klase koji mogu na osnovu asocijacije biti povezani sa jednim te istim objektom druge klase.

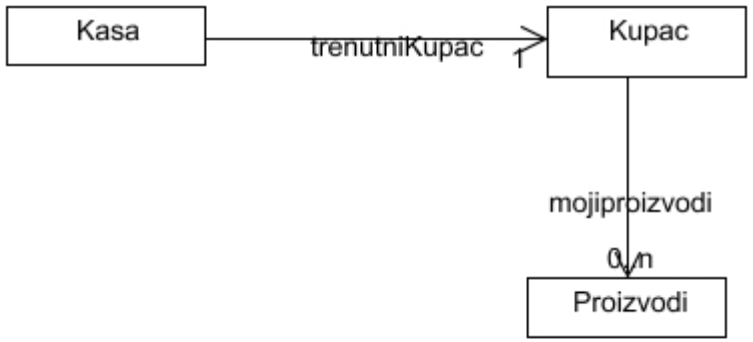
Indikator	Značenje
0..1	Niti jedan ili jedan
1	Samo jedan
0..* ili *	Niti jedan ili više
1..*	Jedan ili više
X,Y,Z	X, Y ili Z objekata
N	Samo N objekata gdje je N>1
0..N	Od nula do N objekata gdje je N>1
1..N	Od jednog do N objekata gdje je N>1

Smjer asocijacije

- *Smjer asocijacije (navigabilnost)* određuje da li se od objekta jedne klase može pristupiti povezanim objektima iz druge klase.
- *Navigabilnost* zapravo znači da objekat prve klase sadrži reference na povezane objekte iz druge klase. Ovisno o višestrukosti, to može biti jedna referenca ili cijela kolekcija.
- Postoji više dogovora kako da se navigabilnost pročita s dijagrama. Uobičajeni način:
  - asocijacija u obliku spojnice bez strelica ili sa strelicom na oba kraja se smatra navigabilnom u oba smjera,
  - asocijacija sa strelicom je navigabilna samo u smjeru strelice.

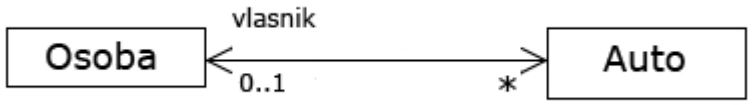
Smijer asocijacije

- Jednosmjerna asocijacija




```
class Kasa{
   private Kupac trenutnikupac;
   ...
}
class Kupac{
   private Proizvod[] mojiproizvodi;
   ...
}
class Proizvodi{
   ...
}
```

- Dvosmjerna asocijacija




```
class Osoba{
   private Auto [] automobili;
   ...
}
class Auto{
   private Osoba vlasnik ;
   ...
}
```

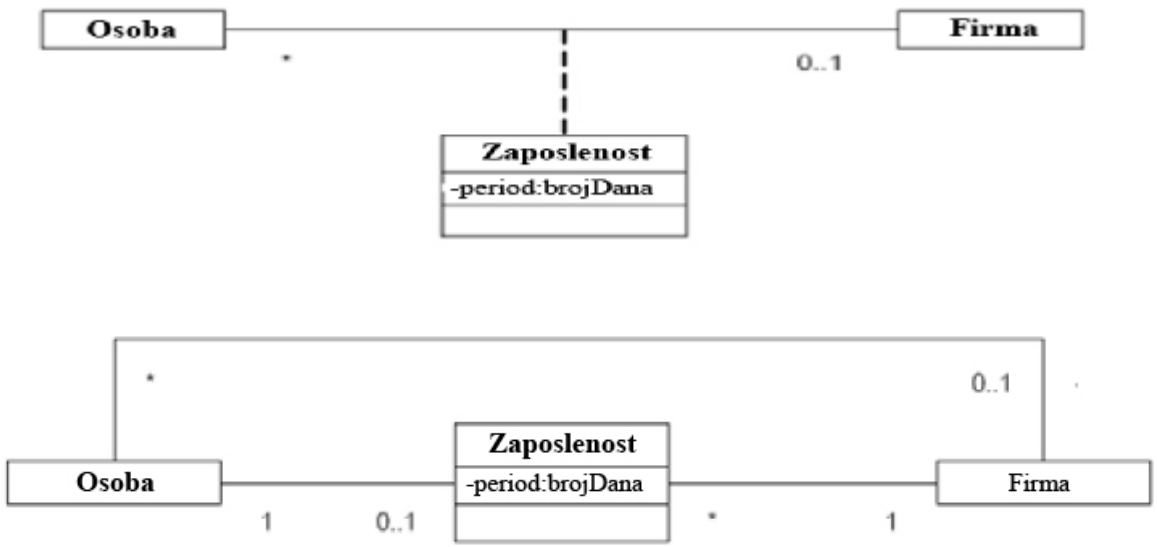
Asocijativna klasa

- Pridruživanjem atributa i operacija relaciji asocijacije dobivamo asocijativne klase
  - Omogućava nam definisanje dodatnih pravila za relaciju
  - **ograničenje:** smije postojati samo jedna instanca asocijativne klase za par povezanih objekata, inače postaje prava klasa

- Pažljivo koristiti jer vrlo lako dovodi do zabune i generalno je bolje modelirati uvijek u vidu punih klasa

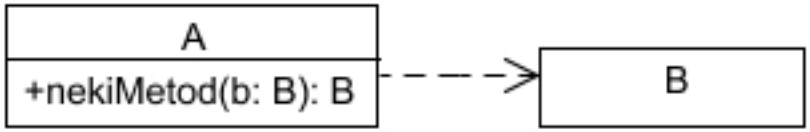
Asocijativna klasa - primjer

- Sljedeći primjer prikazuje dijagram za vezu između uposlenika i kompanije (jedna osoba radi za jednu kompaniju).
- Za svaku osobu je potrebno čuvati informacije o periodu uposlenja u kompaniji.



Ovisnost

- Ako klasa A u nekoj operaciji koristi klasu B kao podatak, ulazni ili povratni parametar operacije tada je A ovisna od B




```
class A{
        ...
        public B nekiMetod(B b){
                B y = new B();
                 ...
                return y;
         }
}
```

Ovisnost

- Budući da je ovisnost vrlo općenita tvorevina koja može različita značenja, njena semantika se nastoji pobliže odrediti korištenjem stereotipa

<<call>>	klijentova operacija poziva dobavljačevu operaciju
<<parameter>>	dobavljač je parametar ili povratna vrijednost u klijentovoj operaciji
<<use>>	klijent na neki način koristi dobavljača, nismo odredili kako. Ovo je default ako ništa ne piše
<<substitute>>	klijent može u run-time-u služiti kao zamjena za dobavljača
<<instantiate>>	klijent je instanca dobavljača
<<derive>>	klijent je izveden iz dobavljača
<<refine>>	klijent je profinjena verzija dobavljača

Agregacija

- Označava povezanost cjeline s nekim njenim dijelom (relacija “je dio”)
- Komponente su dijelovi objekta takvi da
  - dio može nastati i postojati nezavisno od cjeline
  - uništenje "cjeline" ne uništava njene dijelove
- Objekt može pripadati u više cjelina
  - istu instancu nekog objekta može dijeliti više agregata
- UML puno bolje definiše relaciju kompozicije
- Generalno treba izbjegavati agregaciju jer ne daje jasnu predstavu

Kompozicija

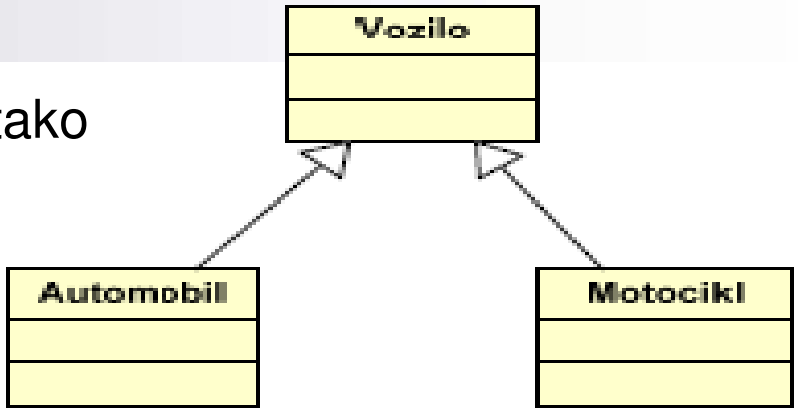
- Kompozicija povezuje dio s cjelinom s time da se dio ne može izostaviti iz cjeline
- Objekt može pripadati samo jednoj cjelini
- Elementi postoje tako dugo dok postoji cjelina
  - kaskadno brisanje, tj. uništenje sastavnih objekata
- Kardinalnost na kraju cijeline kompozicije mora biti 1 ili 0..1 – a dio ne može biti dio više od jedne cjeline.

Kompozicija - primjer

- Tačka može biti element poligona ili centar kruga, ali ne može biti jedno i drugo.
- U kompoziciji važi opšte pravilo da, iako klasa može biti komponenta mnogih drugih klasa, svaka instanca mora pripadati samo jednom vlasniku.
- Osnovno pravilo kompozicije je da nema dijeljenja.

Generalizacija

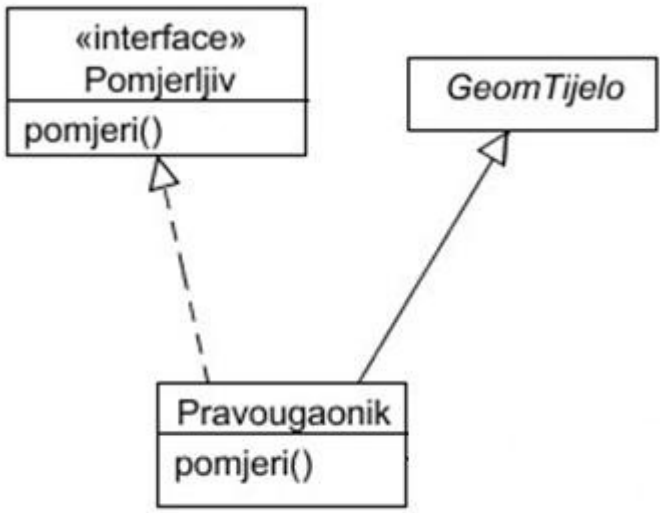
- Odnos između bazne(općenite) i izvedene klase
  - nadređena klasa ili nadtip (*superclass*, *supertype*)
  - podređena klasa ili podtip (*subclass*, *subtype*)
- Sve što vrijedi za baznu klasu vrijedi i za izvedenu klasu ali ne i obrnuto (nasljeđivanje)
- Pod-klasa nasljeđuje sljedeće osobine od nad-klase:
  - atribute,operacije, veze, ograničenja.
- Pod-klasa može:
  - Dodati nove osobine,
  - Redefinisati naslijeđenu operaciju tako da definiše novu operaciju s istom signaturom.



Implementacija

- Relacija “implementira” veže se za pojmove apstraktna klasa i interface
- Pokazuje koje apstraktne klase nasljeđuje klasa odnosno koje interface implementira
- Apstraktna klasa je klasa koja ima barem jedan apstraktni metod tj. deklarisani metod koji nema implementaciju unutar klase
- Nije moguće kreirati objekat te klase
- U UML notaciji označava se *kosim* pismom
- Interface je apstraktna klasa koja nema implementiran niti jedan od deklarisanih metoda i ne sadrži niti jedan član
- Označava se sa <<interface>>

Generalizacija i implementacija



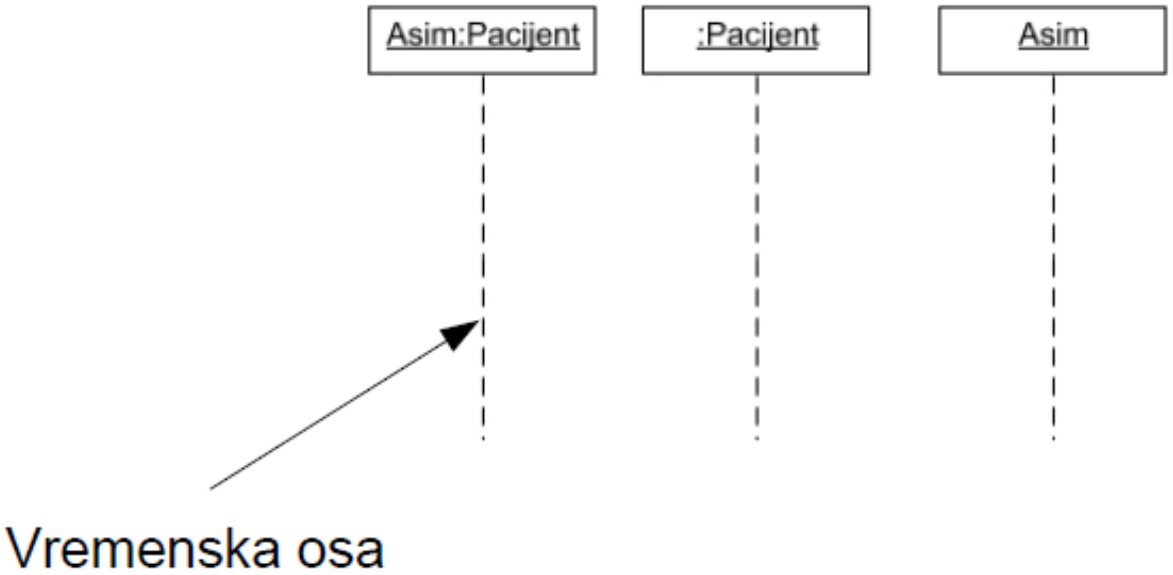

```
interface Pomjerljiv {
        void pomjeri(int x, int y);
}

abstract class GeomTijelo {
....
}

class Pravougaonik : GeomTijelo,
Pomjerljiv
{
...
}
```

Modeliranje ponašanja

- Dijagrami sekvenci – najčešće korišteni UML dijagrami za prikaz dinamičkog modela sistema
- Koriste se za vizuelno objašnjenje interakcije između objekata u modelu
- Elementi dijagrama sekvenci
  - Akter (*participant*)
    - Objekat (ili entitet) koji učestvuje u sekvenci
  - Poruka (*message*)
    - Komunikacija između aktera

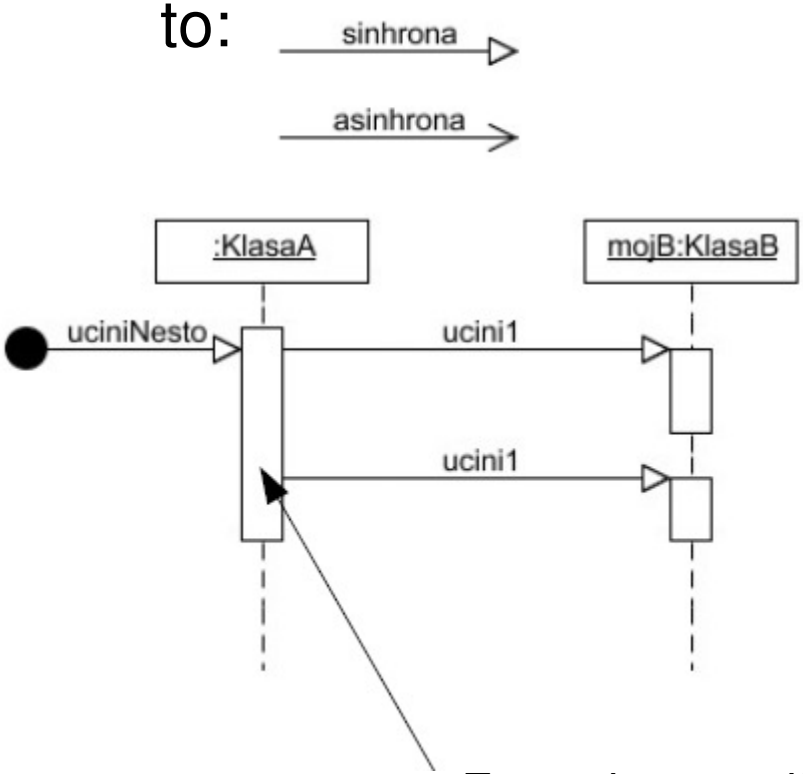
Objekti u dijagramima sekvenci

- Prikaz se vrši na dvije ose
  - Horizontalna osa prikazuje aktere u obliku pravougaonika
  - Vertikalna osa prikazuje vrijeme izvršenja
- Pravougaonici predstavljaju objekte
  - Ime ili tip objekta nije obavezan
  - Generalni format je: `<ime_objekta>:<tip_objekta>`



Poruke

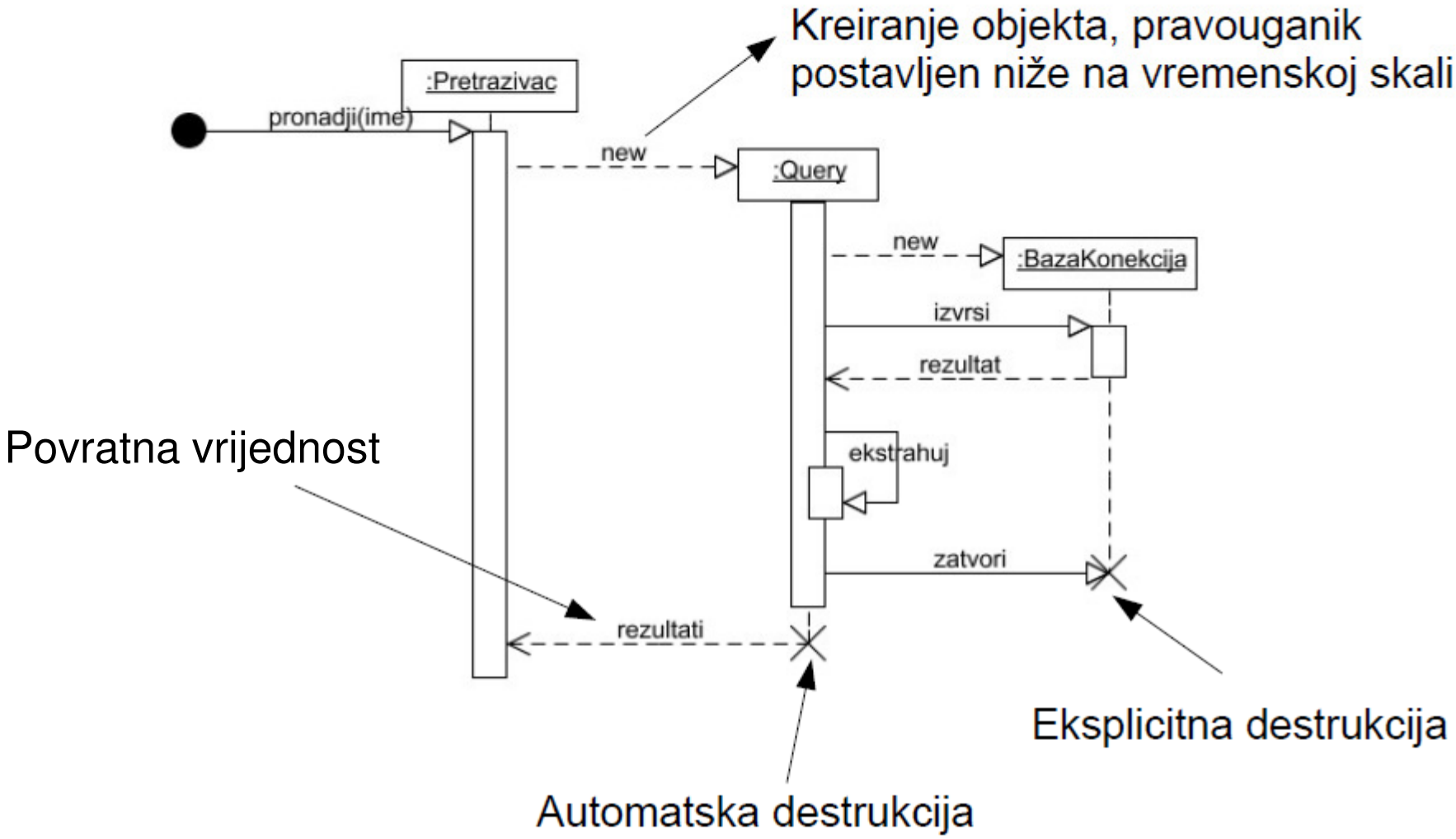
- Poruke se prikazuju horizontalnom linijom između objakata sa naznačenom strelicom
  - Različiti tipovi strelica označavaju različite tipove poziva i to:




```
public class KlasaA
{
    private KlasaB mojB = new KlasaB();

    public void UciniNesto()
    {
        mojB.ucini1();
        mojB.ucini2();
    }
    //...
}
```

Execution specification (ili *activation*) *bar* prikazuje dužinu izvršenja operacije na stacku

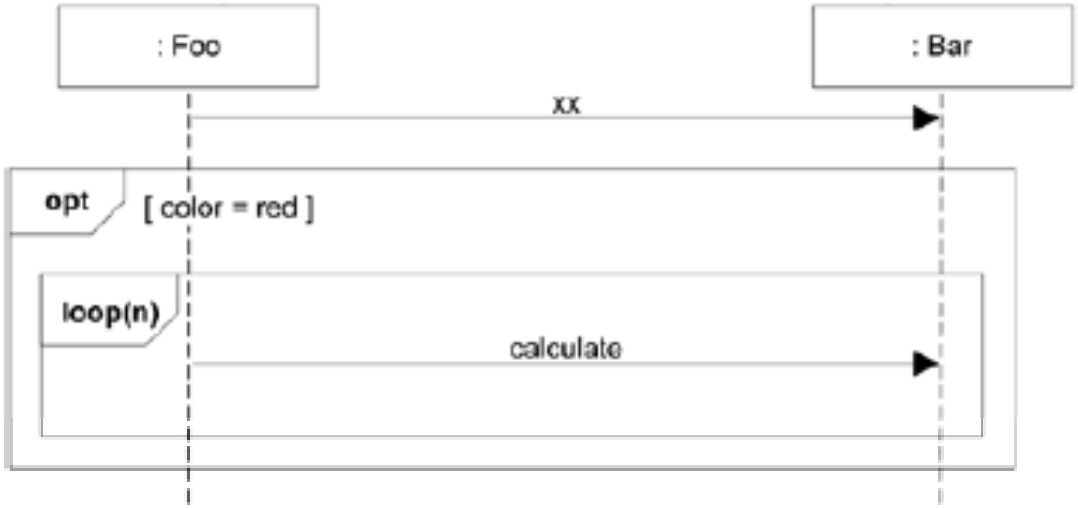
Još o porukama

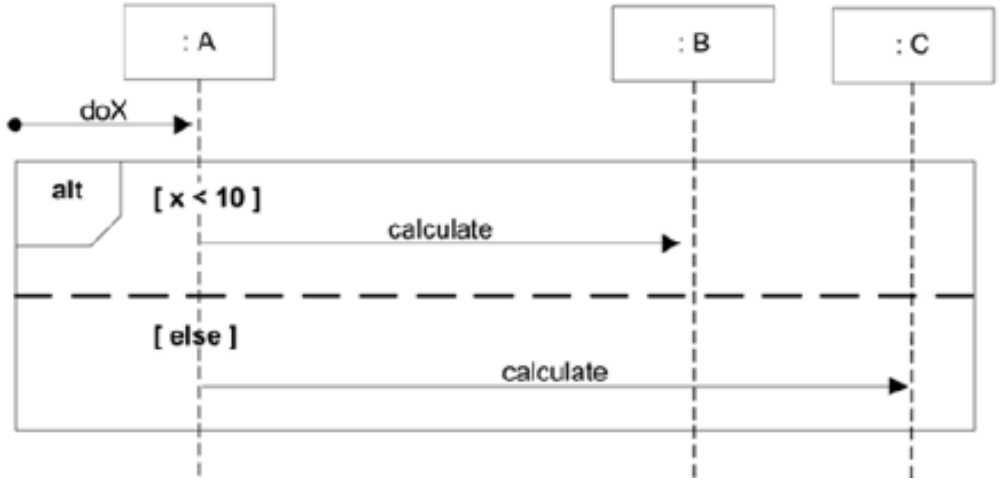


Okviri

- Za prikaz petlji i grananja u metodama koriste se okviri. Okviri sadrže
  - Kontrolnu varijablu (ili logički izraz)
  - Operator
    - alt – fragmenti međusobno isključive logike u ovisnosti od kontrolne varijable
    - loop – ponavlja fragment dok je kontrolna varijabla tačna
    - opt – opcioni fragment u ovisnosti od vrijednosti kontrolne varijable

Okviri


: Foo
: Bar
xx
opt
[color = red]
loop(n)
calculate


: A
: B
: C
doX
alt
[x < 10]
calculate
[else]
calculate