UML:
Unified Modeling Language
- osnove -

OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- **OO karakteristike**
  - **Sve je objekat.**
  - **Objekat je definisan podacima i skupom metoda koji opisuju njegovo ponašanje.**
  - **Objekti komuniciraju međusobno slanjem poruka.**
  - **Svaki objekat pripada svojoj klasi (ima svoj tip).**
  - **Klasa opisuje strukturu objekta i njegovo "ponašanje".**
  - **Svi objekti iste klase mogu da odgovore na iste poruke.**

OBJEKTNO ORIJENTISANO PROGRAMIRANJE

Osnovni koncepti

- **Klasa**
  - Definiše strukturu objekata (podatke) i njihovu funkcionalnost (ponašanje). Klasa može da se posmatra kao šablon objekata, (template) kojim se opisuje model po kome će se kreirati novi objekat.

- **Objekat**
  - Primerci (istance) klasa. pojedinačni,.
  - Svi objekti jedne klase imaju strukturu definisanu klasom i nad njima se mogu izvršavati samo operacije definisane klasom kojoj pripadaju.

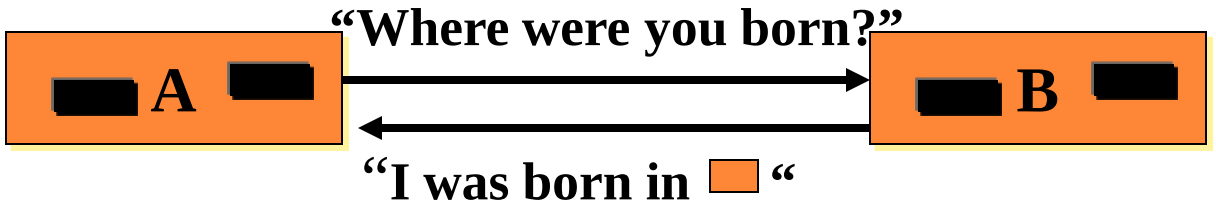
- **Poruka**
  - Objekat odgovara na poruku tako što se izvršava odgovarajući metod.
  - Kažemo da na objekte delujemo porukama, pri čemu svaka poruka pretstavlja poziv metoda (funkcije ili procedure) definisanog u klasi kojoj objekat pripada.

INTERAKCIJA IZMEDJU OBJEKATA

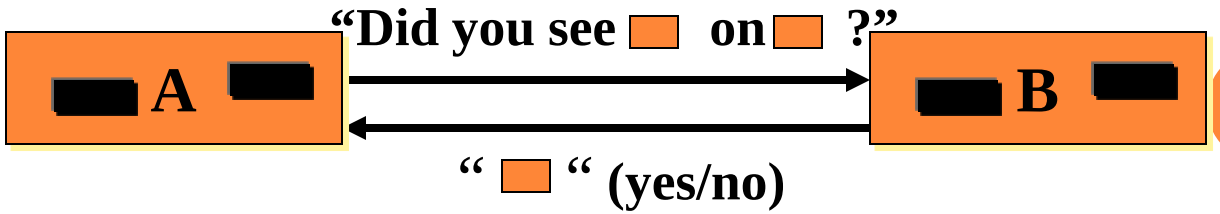
Komande

Komande sa argumentima

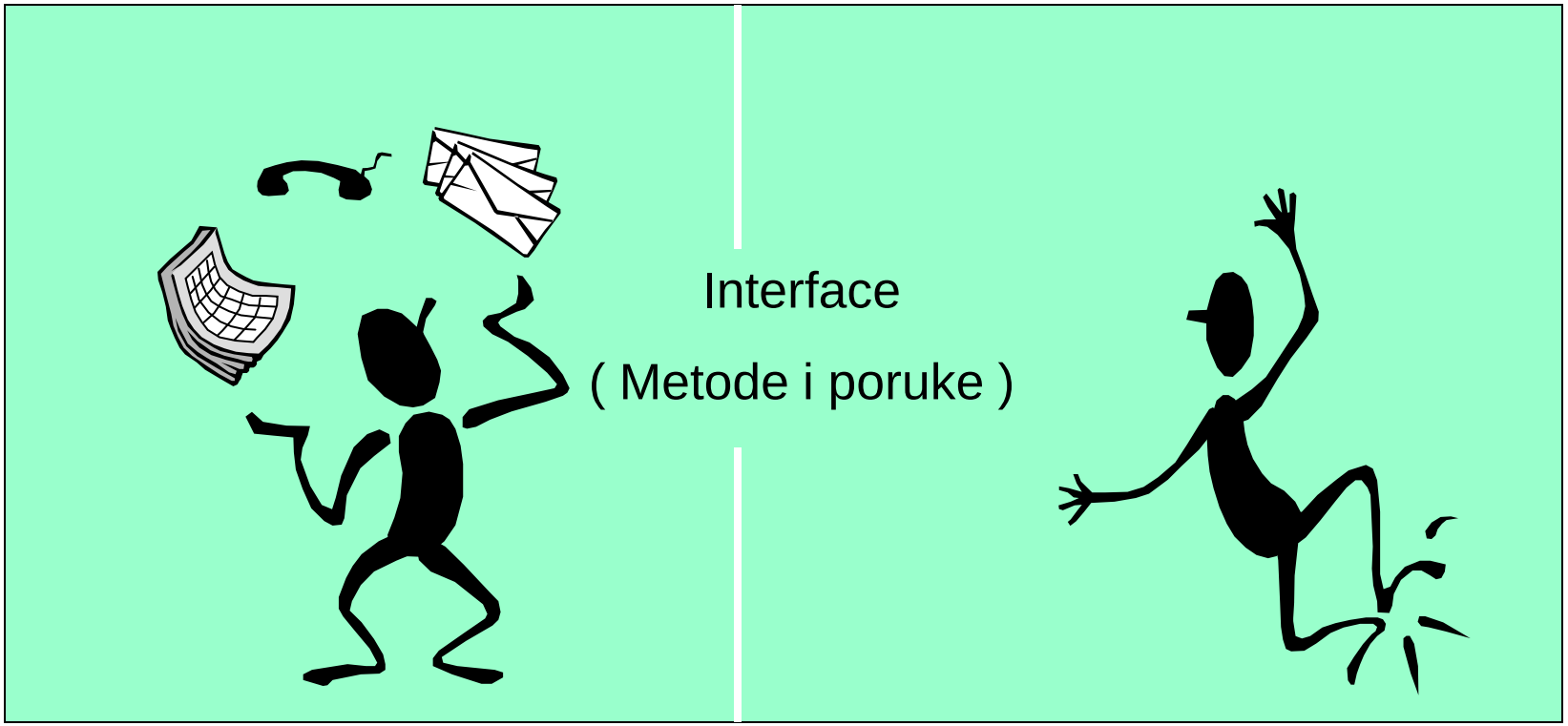
Pitanja



Pitanja sa argumentima



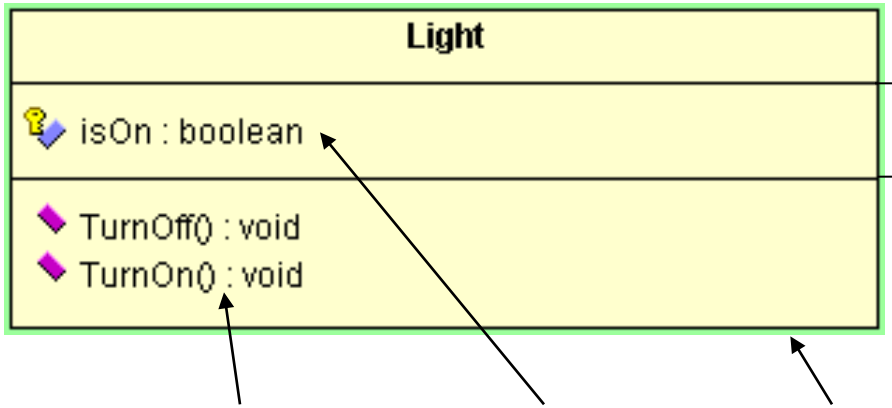
Enkapsulacija



programer, kreator klase

Korisnik klase

Primer



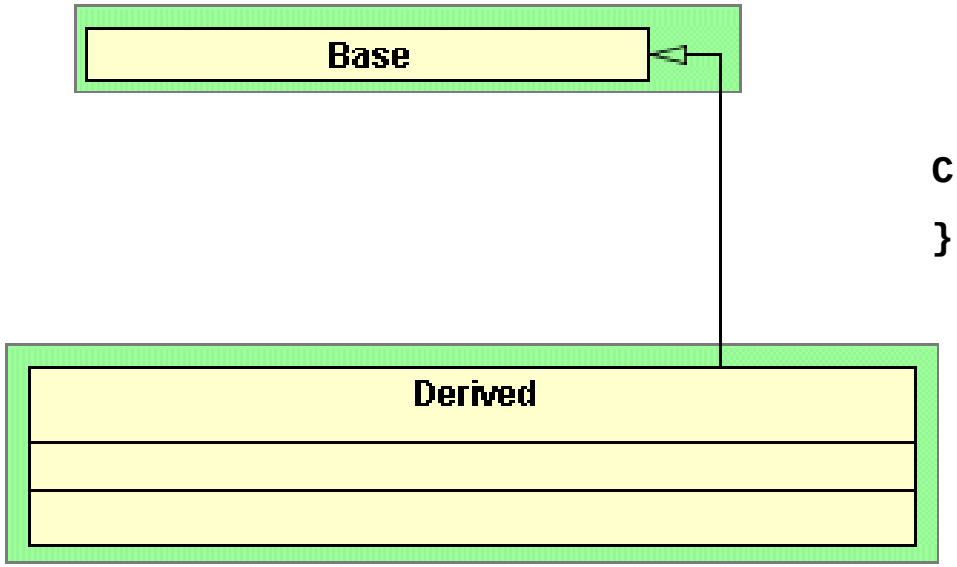

```
public class Light {

  protected boolean isOn;

 public void TurnOff()
{/*...*/}
 public void TurnOn()
{/*...*/}
}
```

Kreiranje objekta:

```
Light kitchenLight = new Light();
kitchenLight.TurnOn();
```

NASLEĐIVANJE

- Nasleđuju se sva svojstva postojeće klase i njima se dodaju nova.
- Uspostavlja se “is-like-a” relacija




```
Class Derived extends Base {
}
```

NASLEĐIVANJE

- Nasleđivanje omogućava da se definišu nove klase na osnovu postojećih.
- Nova klasa se definiše kao specijalizacija ili proširenje neke klase koja već postoji.
- Nova klasa inicijalno ima sva svojstva klase koju nasleđuje.
- Novoj klasi mogu da budu pridodata nova svojstva.
- Kada jedna klasa, u OO terminologiji podklasa ("subclass", "child class", "derived class"), nasleđuje neku klasu , u OO terminologiji superklasa ("superclass", "parent class", "base class"), onda podklasa nasleđuje sva svojstva superklase.

NASLEĐIVANJE

- Podklasi mogu biti pridodata nova svojstva (podaci, metodi, poruke), a neka svojstva (prvenstveno metodi) mogu biti izmenjena sa ciljem da se nova klasa prilagodi specifičnim potrebama.
- Sva dodata svojstva i izmene odnose se na poklasu (subclass) dok originalna klasa (superclass) ostaje nepromenjena.

- Svojstva koja nisu definisana u podklasi automatski se preuzimaju iz njene superklase.

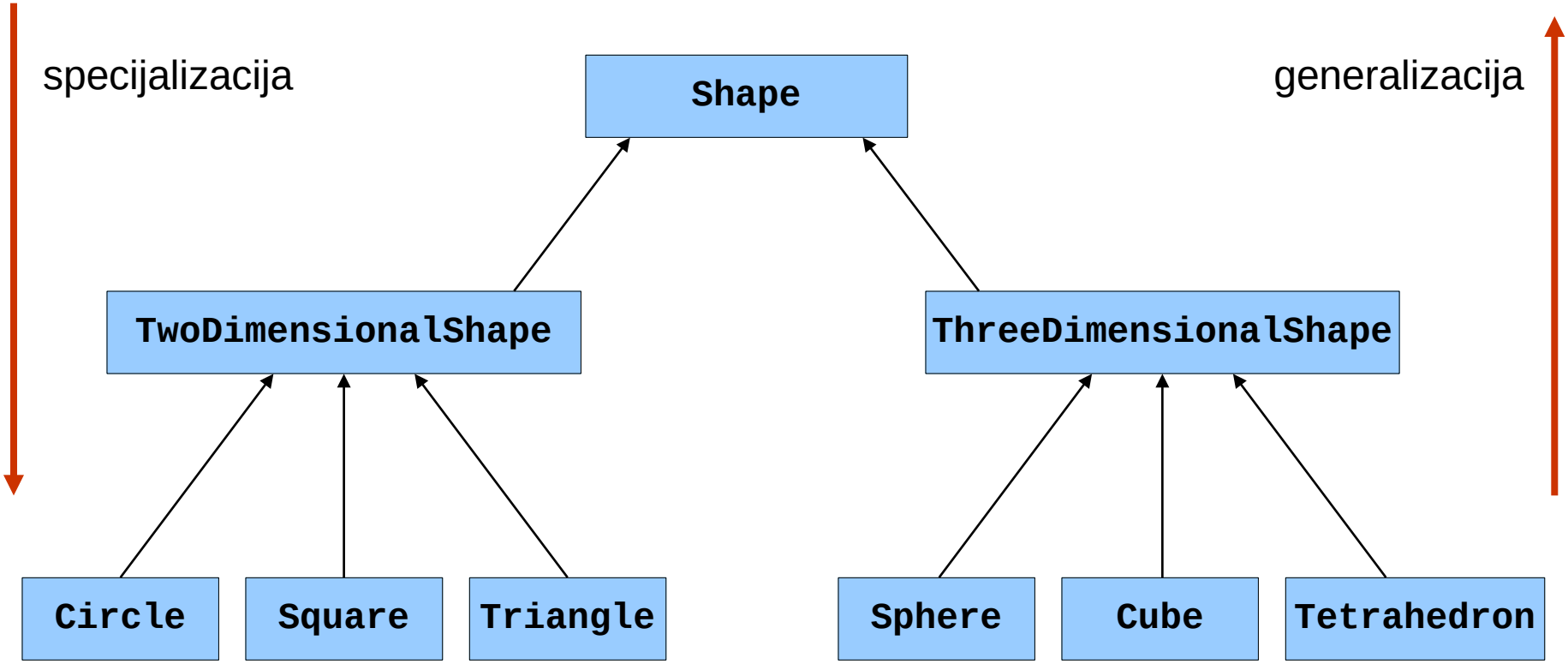
ŠTA SE MOŽE MENJATI?

- ***Nove poruke***, time se proširuje interfejs nove klase. Za svaku novu poruku treba da bude definisan odgovarajući metod.

- ***Novi podaci (atributi) instanci,*** time se proširuje struktura objekata. Posmatrano sa strane apstrakcije objekti podklasa su specijalizovani (detaljnije opisani) od objekata superklasa. Subklase ne samo da imaju veću funkcionalnost već imaju i više svojstava.

- ***Postojeći metodi mogu biti predefinisani***, na taj način se modifikuje ponašanje objekta. U ovom slučaju govorimo o predefinisanju metoda.

PRIMER NASLEĐIVANJA

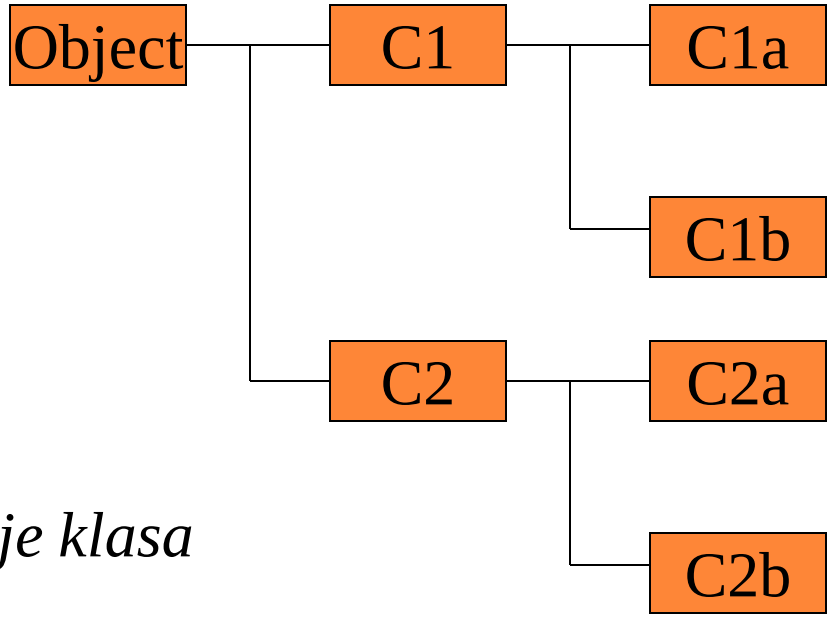


IS A RELACIJA

- Nasleđivanjem se definiše IS A relacija između originalne klase i nove, izvedene klase.
- Kako objekti podklase mogu da prime sve poruke kao i objekti superklase oni se mogu koristiti svuda gde se mogu koristiti i objekti superklase.
- Svaki objekat jedne klase je i (is an) objekat njene superklase.

HIJERARHIJA KLASA

- Kada se nasleđivanje primeni višestruko dobija se struktura tipa stabla koja se obično naziva hijerarhija klasa.

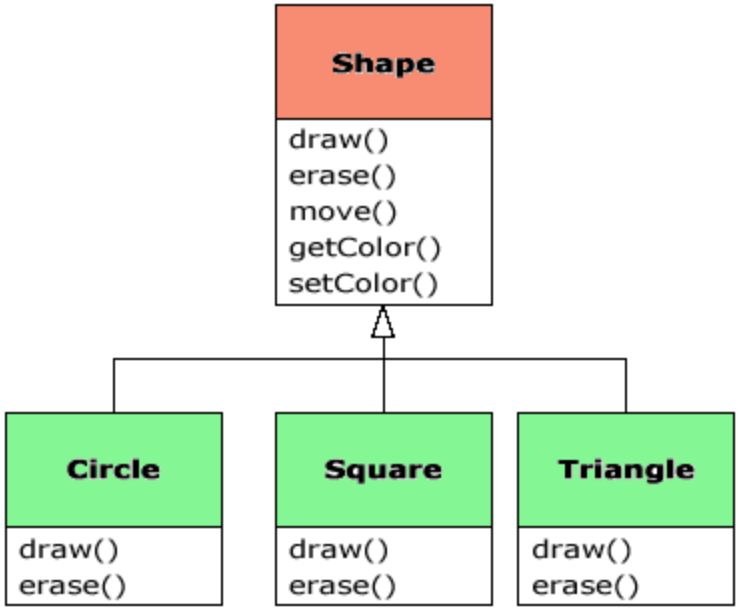


Primer hijerarhije klasa

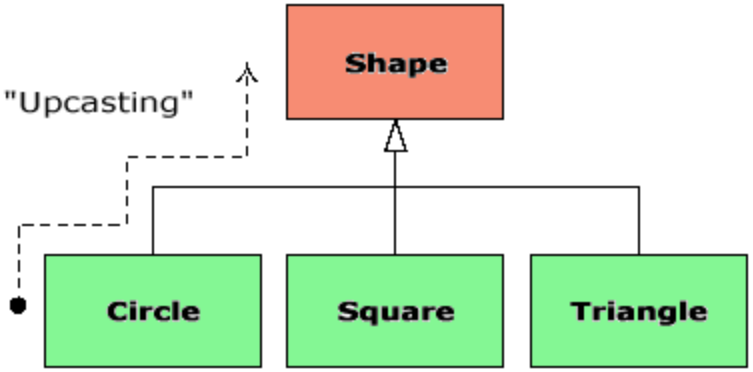
POLIMORFIZAM

- Polimorfizam se često navodi kao najjače svojstvo OO jezika.
- Termin označava “više formi” - i u kontekstu objektnih jezika ukazuje na mogućnost da se metodi više objekata pozivaju preko istog interfejsa.
- Sposobnost da različiti objekti odgovore na iste poruke (na sebi svojstven način).
- Koristi se isti interfejs za različite objekte.

PRIMER POLIMORFIZMA

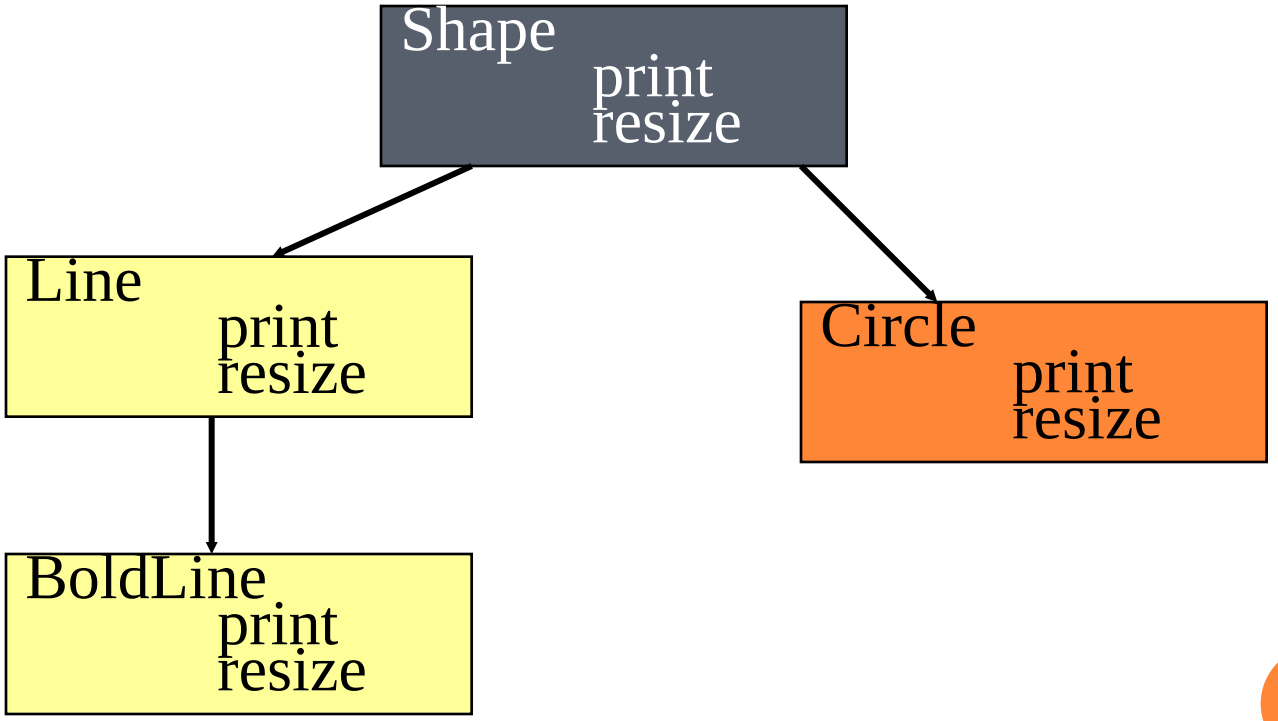



```
Shape c = new Circle();
Shape t = new Triangle();
Shape s = new Square();
// ...
c.draw();
t.draw();
s.draw();
```



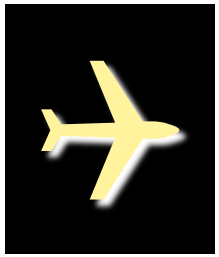
APSTRAKTNE KLASE I INTERFEJSI

- Osnovne klase koje služe kao osnova za generisanje novih klasa.



APSTRAKCIJA

Mentalni proces izdvajanja nekih karakteristika i svojstava i isključivanja preostalih koja nisu relevantna



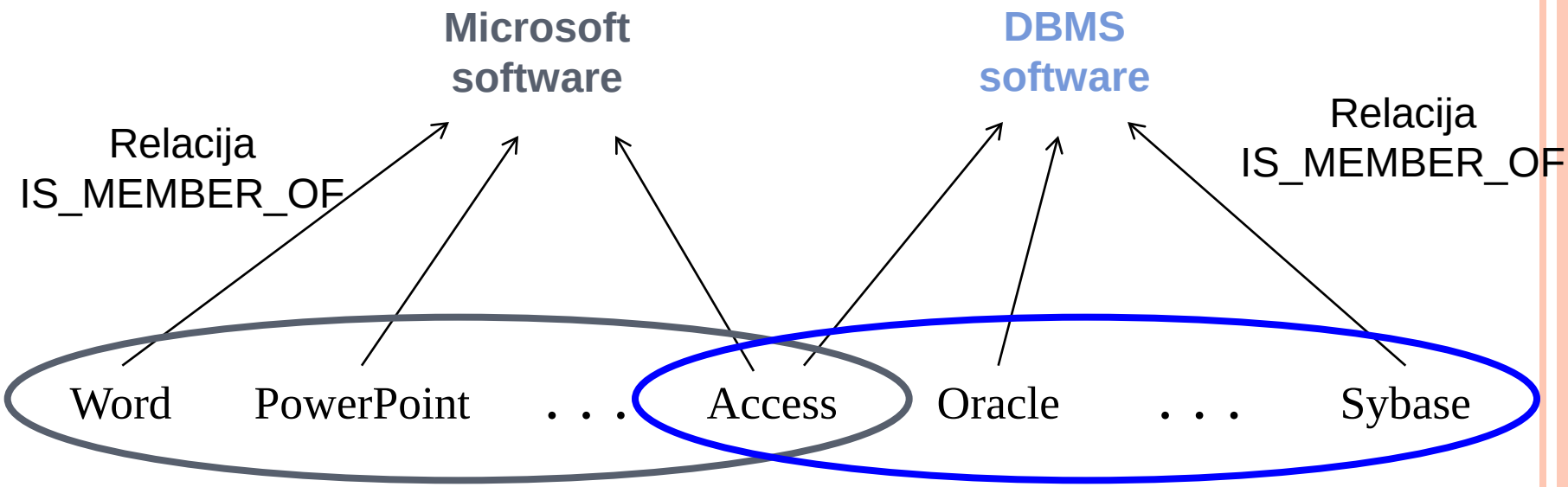
- ima krila, leti

- Apstrakcija je uvek sa nekim ciljem, namenom
- Iste stvari moguće je apstrahovati na više različitih načina
- Svaka apstrakcija je nepotpuna slika realnosti

➔ *Ne želimo potpunost, već samo adekvatno modeliranje!*

TIPOVI APSTRAKCIJE

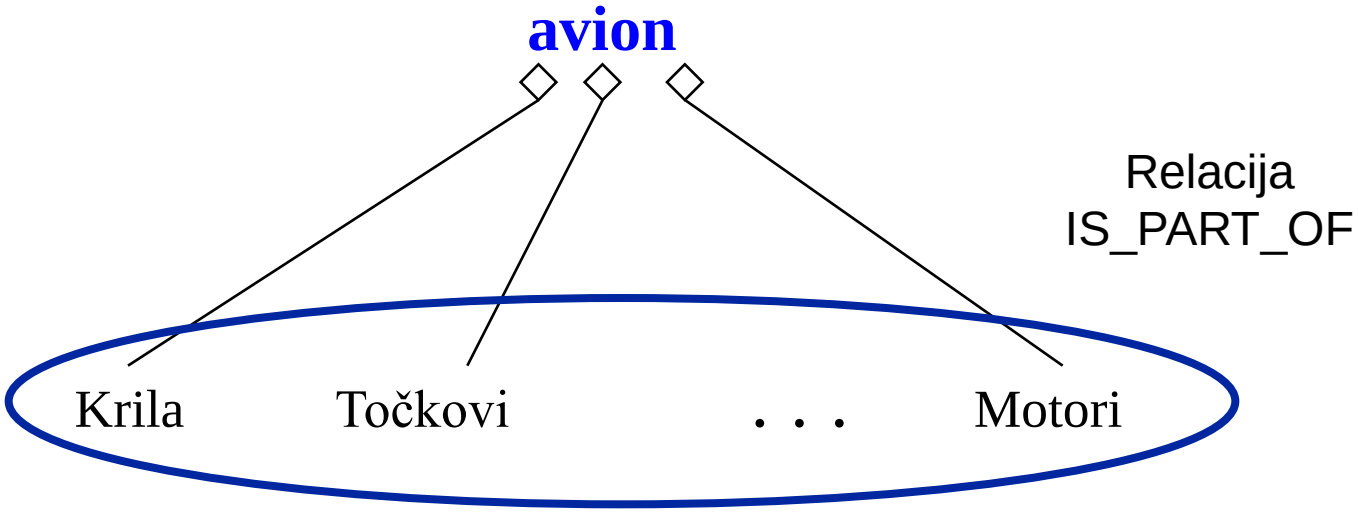
- Klasifikacija — grupa sličnih instanci objekta



➔ Ističu se zajednička svojstva i ignorišu posebna

TIPOVI APSTRAKCIJE

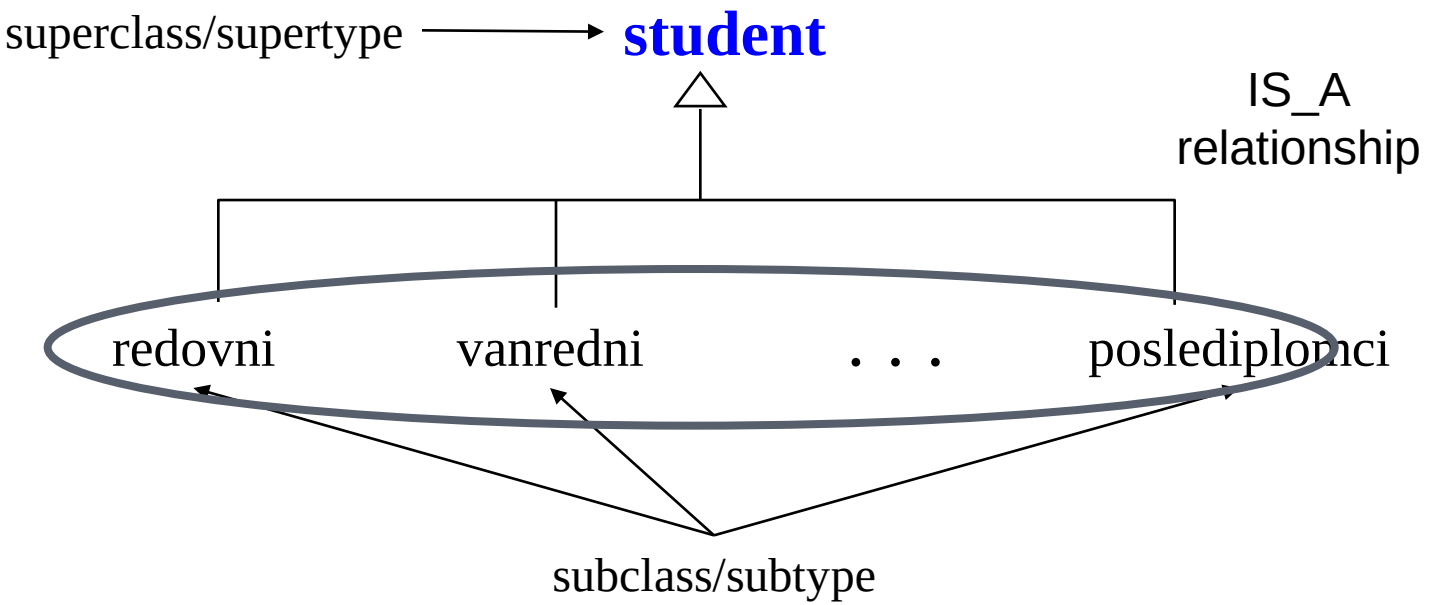
- Agregacija — grupa različitih skupova objekata



➔ ignorišu se razlike između delova - Koncentrišemo se na činjenicu da oni čine celinu

TIPOVI APSTRAKCIJE

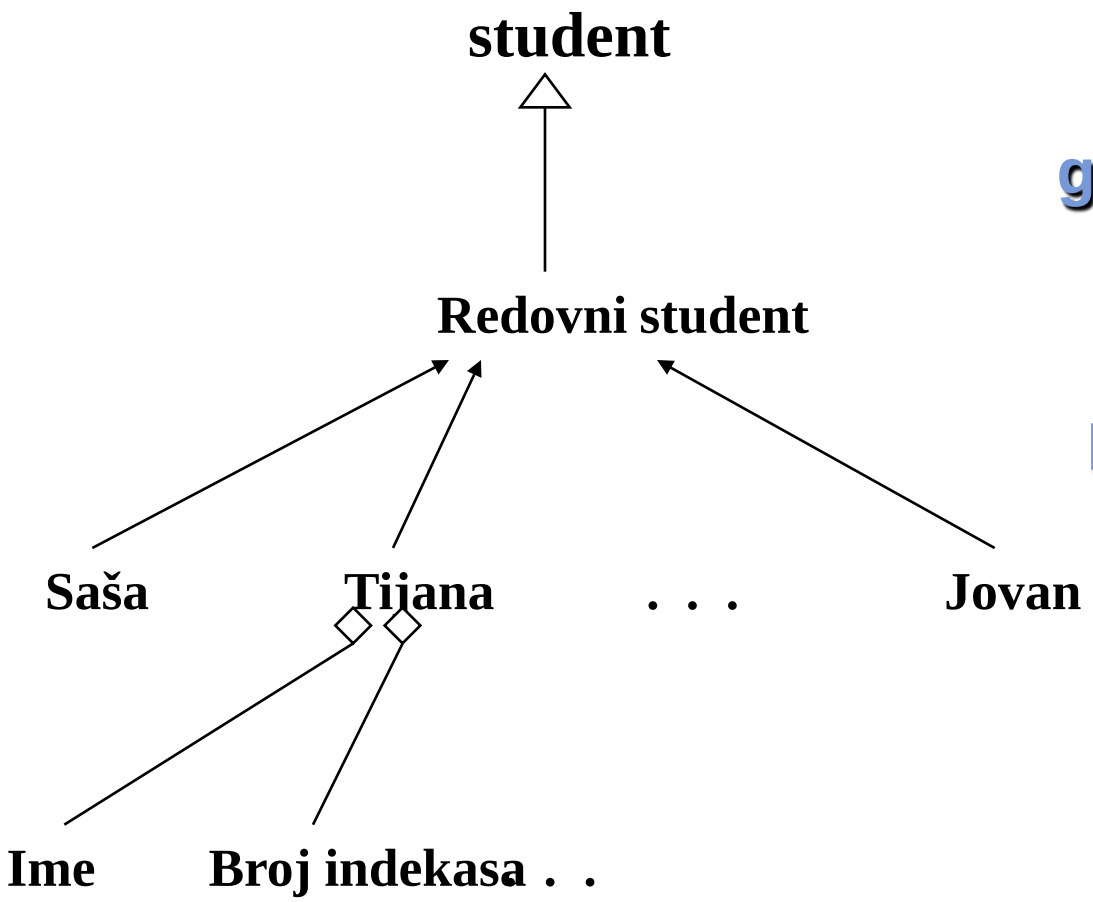
- Generalizacija — grupa sličnih skupova objekata



➔ Uočimo razliku između klasifikacije i generalizacije

- klasifikacija – primenjuje se na individualne instance objekata
- generalizacija – na **skup** objekata (klase)

TIPOVI APSTRAKCIJE



generalizacija

klasifikacija

agregacija

Objektno-orijentisano programiranje (OOP)

- nedovoljno efikasno
- nedostaci:

  - ne pruza dovoljno efikasna sredstva za specifikaciju i dokumentovanje
  - koncepti OOP nedovoljno apstraktni za složene softvere
  - tekstualni način specifikacije je manje efikasan od vizuelnog, grafičkog načina
  - ne pruža dovoljno efikasna sredstva za dobru dokumentaciju

Objektno-orijentisano modelovanje (OOM)

- Razvoj modela softvera na višem nivou apstrakcije

  (koncepti apstraktniji od OOP konc.)
- Specifikacija modela pomoću vizuelnih grafičkih notacija
- Transformacija OO modela softvera u implementacione forme (C++ kod)

MODELOVANJE U SOFTVERU

- Softver - primaran
- Dobar softver = dobra softverska praksa
- Softverski proizvod je kao i svaki drugi
- Model je pojednostavljenje stvarnosti
- Model = razumevanje
  - struktura
  - ponašanje
  - plan
  - dokumentacija

MODELOVANJE

- Modeli razbijaju kompleksnost
- Modelovanje direktno utiče na rešenje
- Modelovati prirodno!
- Svaki model ima nivoe detaljnosti
- Najbolji su modeli koji stoje na zemlji
- Mora biti više modela

U M L

- Standard u softverskom modelovanju
  - vizuelizuje (vizuelni grafički jezik)
  - definiše (precizni, nedvosmisleni modeli)
  - konstruiše (softverski sistem koji se posle može implementirati)
  - dokumentuje (zahtevi, arhitektura, projekat, izvorni kod, itd)
- Širok opseg primene: svuda
- UML sredstvo, a ne metod!!!

UML koncepti

- UML se može koristiti za:
  - Prikazivanje granica sistema i njegovih glavnih funkcija koristeći slučajeve korišćenja i aktere.
  - Ilustrovanje realizacije slučajeva korišćenja dijagramima interakcije (sekvencijalni i kolaboracioni dijagrami).
  - Predstavljanje statičke strukture nekog sistema korišćenjem klasnih dijagrama.
  - Modeliranje ponašanja objekata dijagramima promene stanja (dijagrami aktivnosti i dijagrami stanja).
  - Otkriva fizičku implementaciju arhitekture preko komponentnih dijagrama i dijagrama razvoja
  - Proširivanje naše funkcionalnosti sa stereotipovima

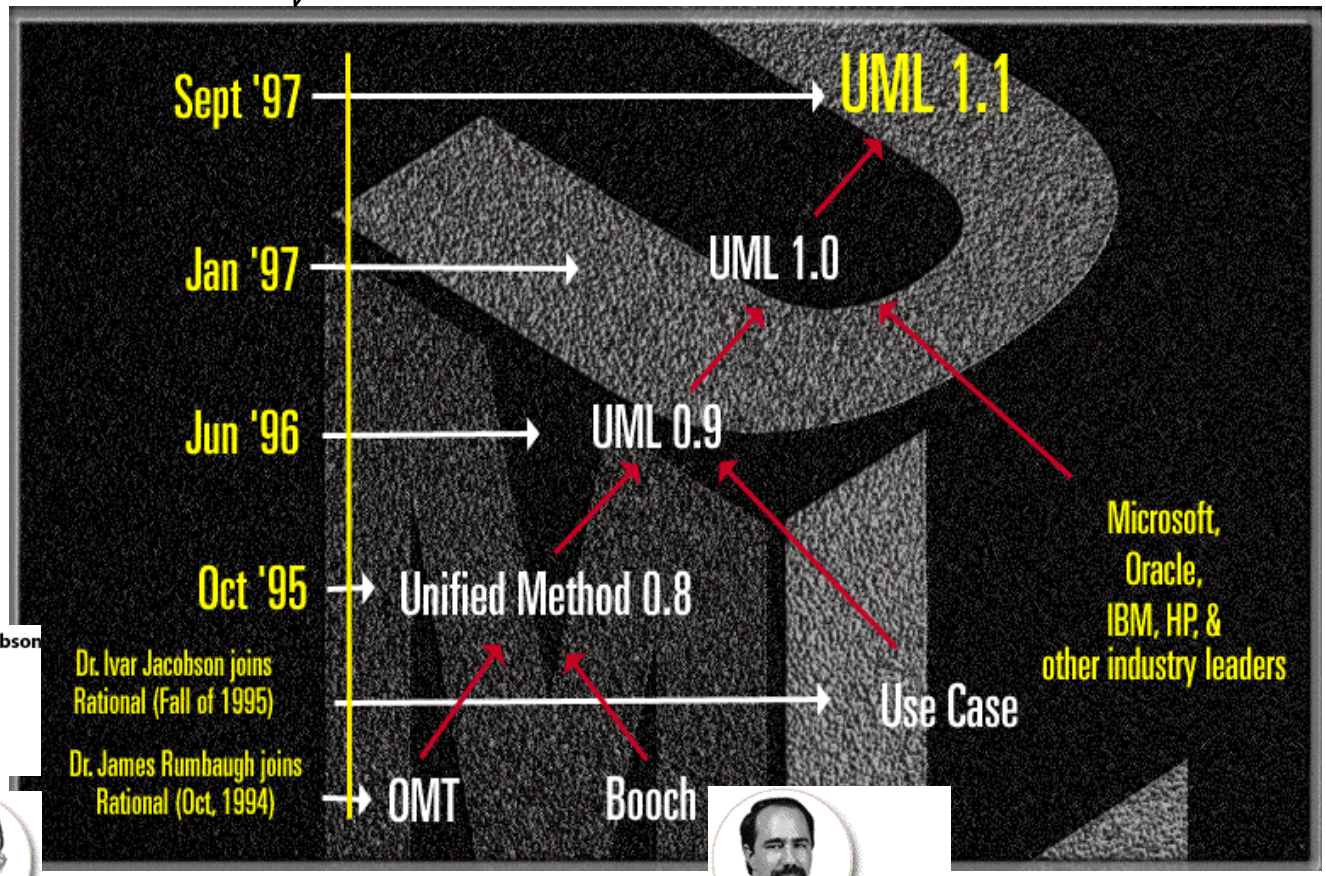
Istorija UML-a

- Jezici za OO modeliranje se pojavljuju još od sredine 70ih
- Posledica pojava nove generacije OO jezika i povećane složenosti softverskih sistema
- U periodu 1989-1994 broj OO metoda je porastao sa manje od 10 na više od 50
- Metode koje su ostvarile najveći uticaj na oblast OO modeliranja su:
  - Booch metoda
  - OMT (*Object Modeling Technique*, Rumbaugh)
  - OOSE (*Object Oriented Software Engineering*, Jacobson)
  - Fusion
  - Coad-Yourdon
- 1994: početak rada na UML-u - Rumbaugh se pridružio Booch-u u firmi Rational
- Oktobar 1995: pojavila se verzija 0.8 drafta UM-a (*Unified Method*)
- Jesen 1995: i Jacobson se pridružio Rational-u - počeli objedinjenje UM sa OOSE
- Jun 1996: pojavila se verzija 0.9 UML-a
- Važniji partneri (za verziju 1.0): DEC, HP, IBM, Microsoft, Oracle, Rational, TI, ...
- U januaru 1997: OMG-u (*Object Management Group*) predlog standarda UML 1.0
- Grupa partnera je proširena drugim podnosiocima predloga (ObjecTime, Ericsson,...)
- Jul 1997: podnet predlog UML 1.1 koji je prihvaćen od OMG 14.11.1997.
- Jun 1998: pojavila se verzija UML 1.2
- Jesen 1998: pojavila se verzija UML 1.3

Istorija UML-a

Nov ‘97

UML odobren od OMG-a



Ivar Jacobson

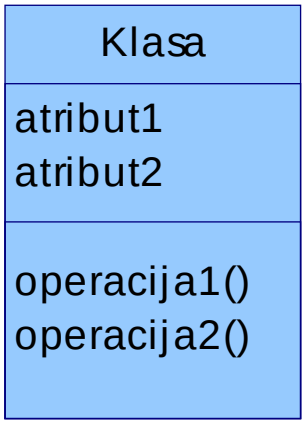
Jim Rumbaugh

Grady Booch

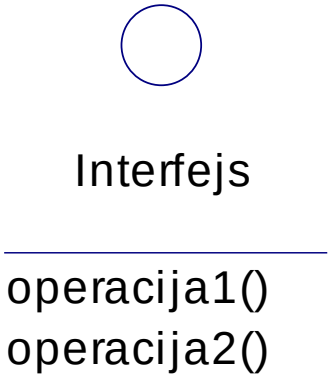
GRADIVNI ELEMENTI UML-A:

- Stvari (things) (sredstva)
  - Strukturne stvari (structural things)
  - Stvari ponašanja (behavioral things)
  - Stvari grupisanja (grouping things)
  - Stvari označavanja (annotational things)
- Relacije (relationships)
- Dijagrami (diagrams)

STRUKTURNE STVARI

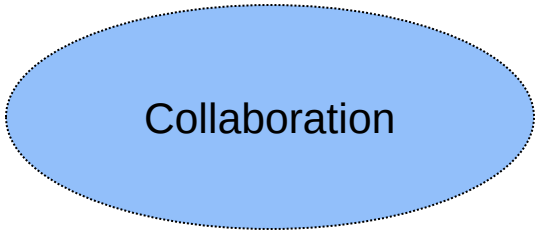


Klasa – opis skupa objekata koji imaju iste atribute, operacije, relacije i semantiku

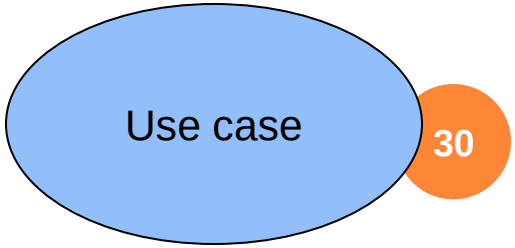


Interfejs – skup operacija koje definišu usluge klase ili komponente (deklaracije ne i implementacije)

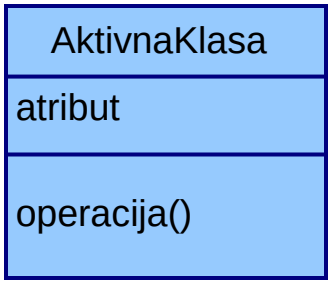
Kolaboracija – skup uloga i drugih elemenata koji saradjuju da bi ispunili kooperativno ponašanje složenije od proste sume ponašanja elemenata



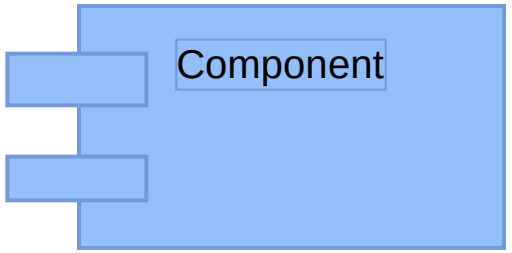
Slučaj upotrebe – opis skupa sekvenci akcija koje sistem izvršava da bi proizveo spolja uočljivo ponašanje bitno za nekog aktera



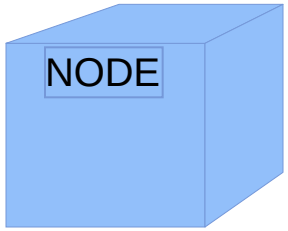
Aktivna klasa – klasa čiji objekti poseduju jedan ili više procesa ili niti kontrole



Komponenta – fizički i zamenljiv deo sistema koji zadovoljava i realizuje skup interfejsa



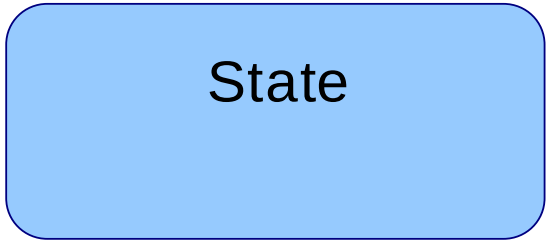
Čvor – fizički element koji postoji u vreme izvršavanja i predstavlja računarski resurs koji u principu poseduje memoriju i najčešće mogućnost obrade



Stvari ponašanja

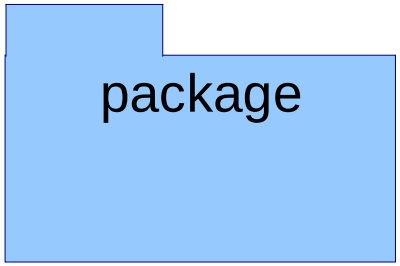
Interakcija – uključuje skup poruka koje se razmenjuju izmedju objekata u odredjenom kontekstu da bi se ostvarila odredjena svrha

Mašina stanja – definiše sekvencu stanja kroz koje objekat ili interakcija prolazi tokom svog života, kao posledica dogadjaja, zajedno sa reakcijama na te dogadjaje



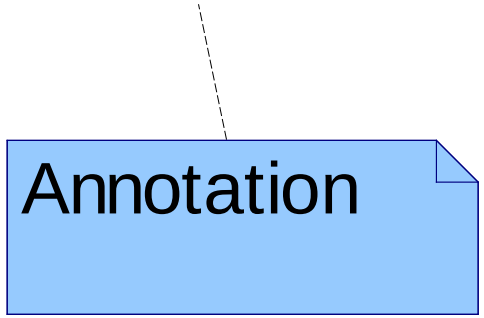
Stvari grupisanja

Paket (package) – Čisto konceptualna stvar u koju se grupišu svi ostali elementi jezika



Stvari označavanja

Komentari (note) – komentari su jedine stvari označavanja



RELACIJE

- **Zavisnost (Dependancy)**: promena jedne stvari utiče na semantiku druge

- **Asocijacija (Association)**: strukturna relacija, veza među objektima

- **Agregacija (Aggregation)** = Asocijacija u varijanti "celina i delovi"

- **Generalizacija (Generalization)**:

  Objekte generalizovanog elementa (roditelja) mogu zameniti objekti specijalizovanog elementa (potomak)

- **Realizacija (Realization)**: semantička veza u kojoj je jedan element ugovor koji drugi element ispunjava

DIJAGRAMI (DIAGRAMS)

- grafička prezentacija skupa elemenata koji predstavlja samo jedan pogled na jedan deo modela
- dijagram ne nosi semantiku i ne sadrži elemente modela, on je samo prikaz nekih elemenata modela
- elementi modela grupisani u pakete (hijerarhijska organizacija)
- paket može sadržati i dijagrame koji prikazuju elemente jednog ili više paketa

Osnovne vrste dijagrama:

Dijagrami za prikaz statičkih aspekata sistema:

–Dijagram klasa (class diagram)

–Dijagram objekata (object diagram)

–Dijagram komponenata (component diagram)

–Dijagram rasporedjivanja (deployment diagram)

Dijagrami za prikaz dinamičkih aspekata sistema:

–Dijagram slučajeva upotrebe (use case diagram)

–Dijagram interakcije (interaction diagram)

- Dijagram sekvence (sequence diagram)
- Dijagram kolaboracije (collaboration diagram)

–Dijagram stanja (statechart diagram)

–Dijagram aktivnosti (activity diagram)

Dozvoljeno pravljenje i ostalih dijagrama, po potrebi !!!

UML PRAVILA (ZA DOBRO FORMIRAN MODEL)

- **Korektan (well-formed) model:**
  - **semantički samokonzistentan**
  - **u harmoniji sa povezanim modelima**
- **UML ima semantička pravila za**
  - **Imena** - kako se nazivaju stvari, relacije i dijagrami
  - **Scope** - koji kontekst daje specifično značenje imenu
  - **Vidljivost** - gde se imena mogu videti i koristiti od strane drugih
  - **Integritet** - kako se stvari propisno i konzistentno dovode u relaciju prema drugim stvarima
  - **Izvršenje** - šta to znači za izvršenje ili simulaciju dinamičkog modela
- **Modeli:**
  - **Skraćeni** - izvesni elementi su sakriveni da se pojednostavi izgled
  - **Nekompletni** - izvesni elementi nedostaju
  - **Nekonzistentni** - integritet modela nije garantovan

OPŠTI MEHANIZMI UML-A

- **Specifikacije** – precizan opis sadržaja svakog elementa. Vizuelni prikaz ne mora sadržati ceo opis. Različiti dijagrami - različito prikazan element

- **Ukrasi** (Adornments) – pridružuju se vizuelnom prikazu elementa da bi se potpunije prikazao njegov sadržaj

- **Mehanizmi proširenja** (extensibility):
  - **Stereotip** - proširuje rečnik UML-a dopuštajući kreiranje novih vrsta gradivnih blokova specifičnih za problem (nove apstrakcije jezika << >>)
  - **Označena vrednost** (tagged values) - proširuju osobine UML gradivnog bloka dopuštajući dodavanje nove informacije { }
  - **Ograničenja** - proširuju semantiku UML gradivnog bloka dopuštajući da se dodaju nova pravila ili promene postojeća { }

Opšte podele

Dve osnovne podele:

- klase (apstrakcija) i objekti (konkretna manifestacija te apstrakcije)
- interfejsi i implementacije

U UML-u se razlika između apstrakcije i instance pravi tako što se imena instanci podvlače

Primeri prve podele:

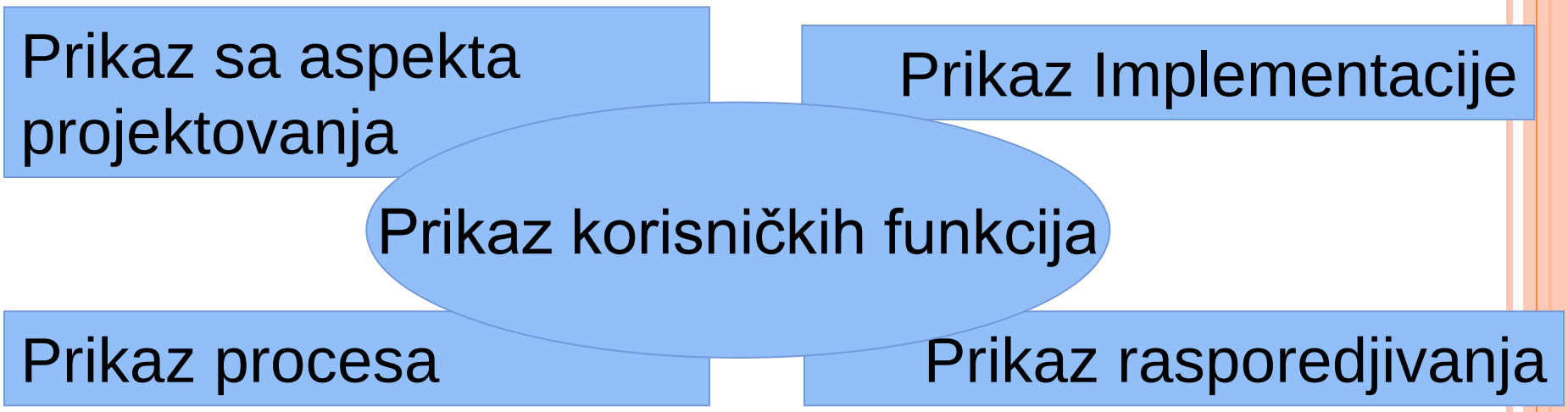
- klase/objekti
- slučajevi korišćenja/instance slučajeva korišćenja
- komponente/instance komponenti

Primeri druge podele:

- interfejsi/komponente
- slučajevi korišćenja/kolaboracije
- operacije/metodi

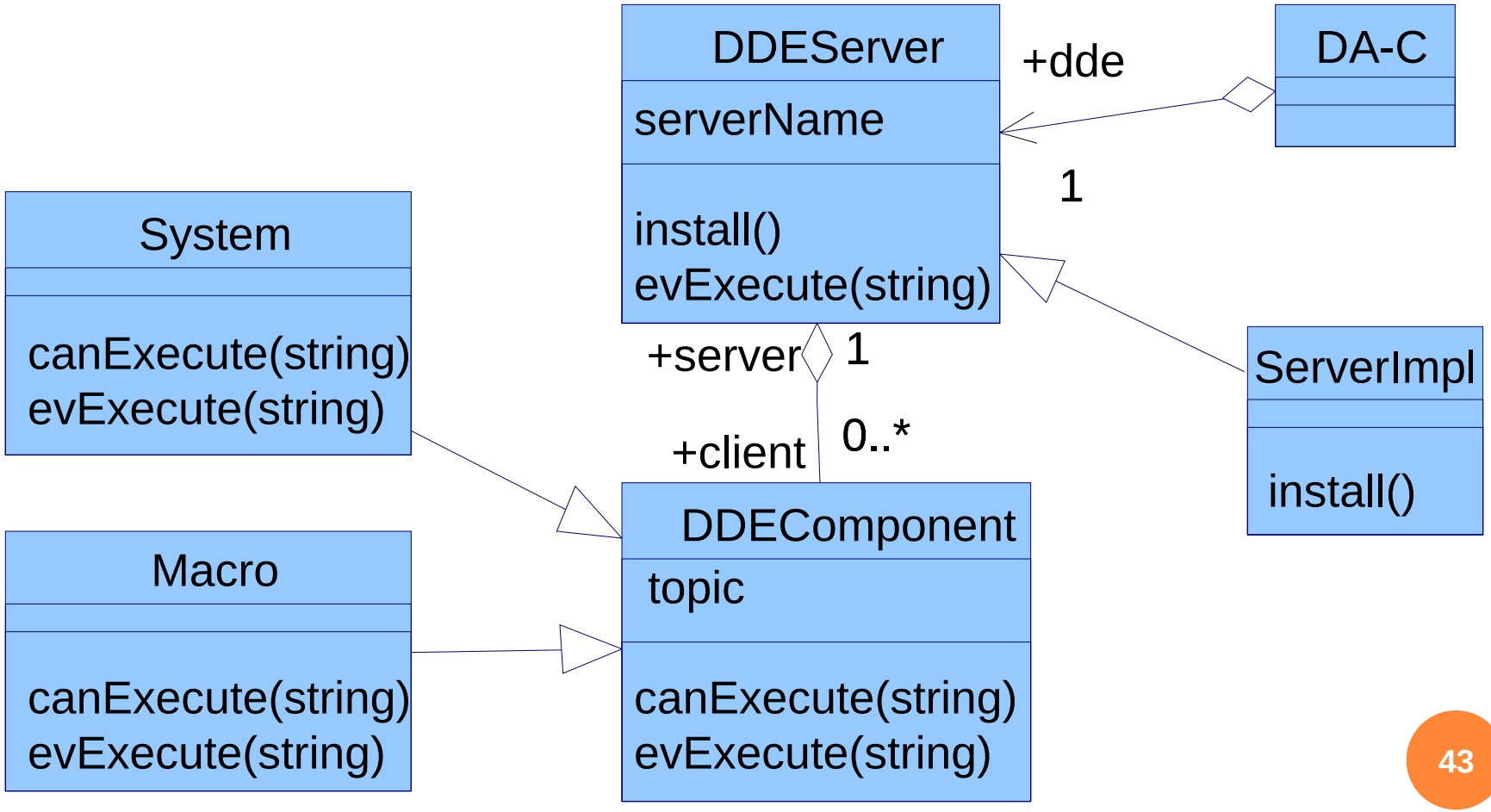
Interfejs deklariše ugovor, a implementacija reprezentuje jednu konkretnu realizaciju ugovora

Integrisani pogled na sistem
(modelovanje arhitekture sistema)



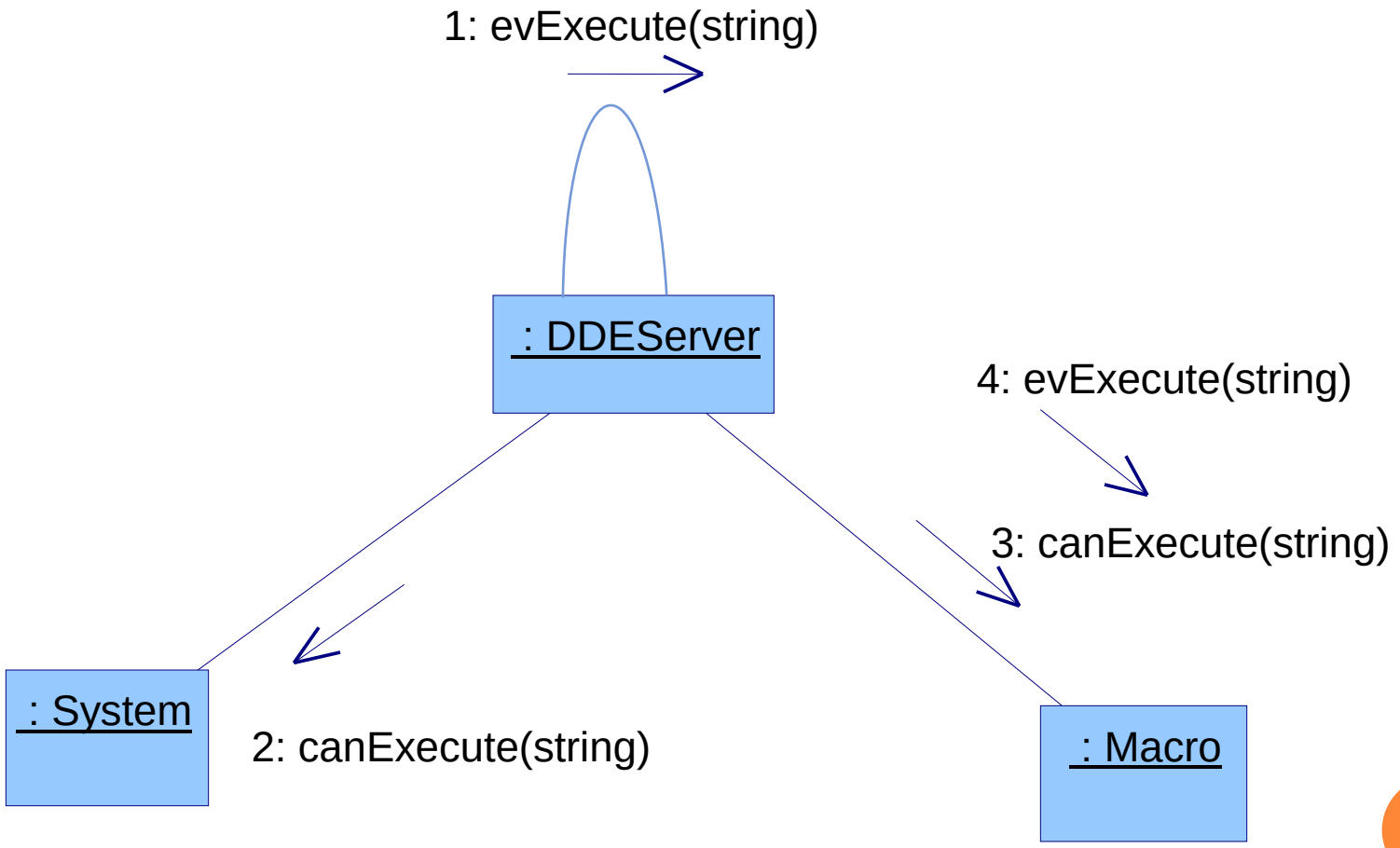
UML PRIMER: DA-C DDE

Dijagram klasa - Strukturni pristup



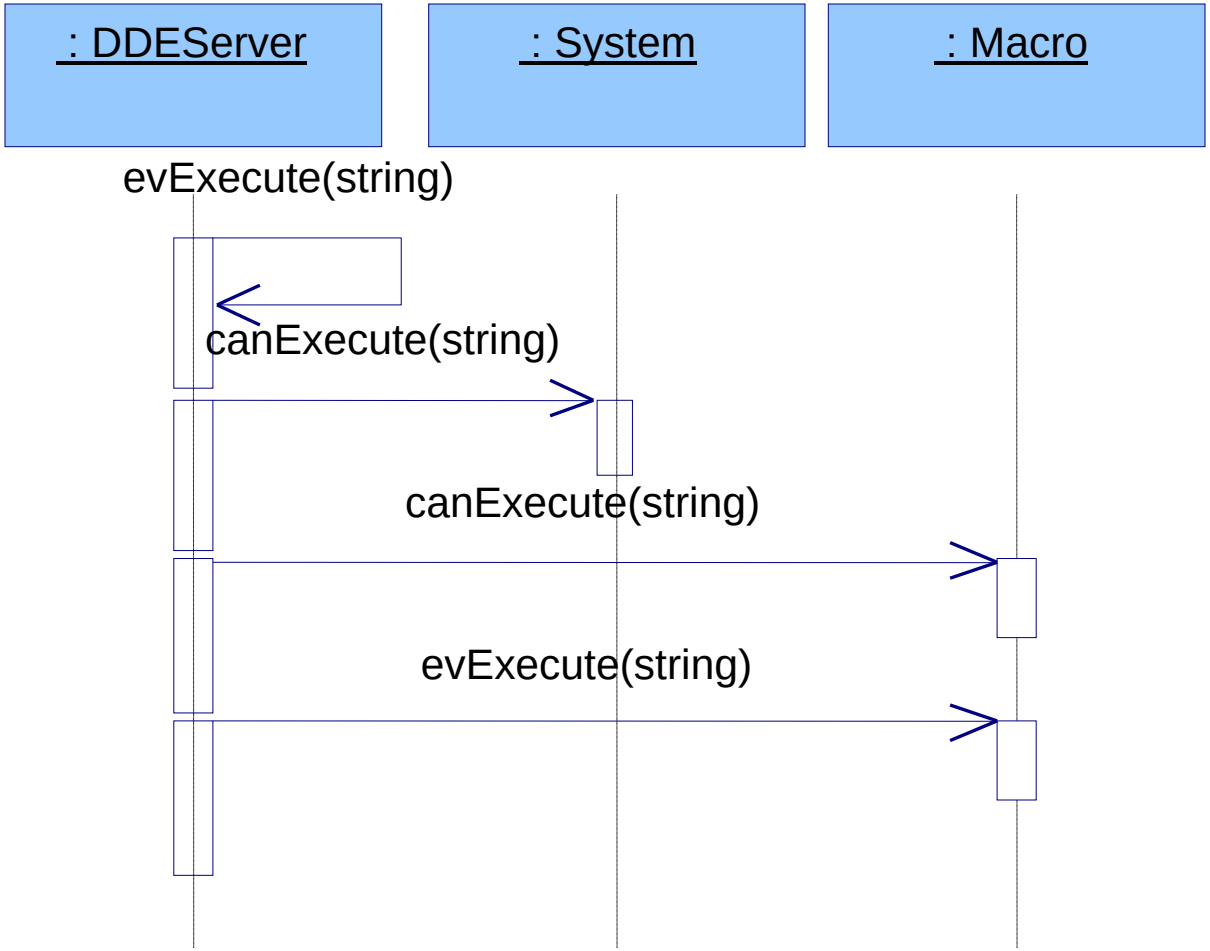
UML PRIMER: DA-C DDE

Collaboration



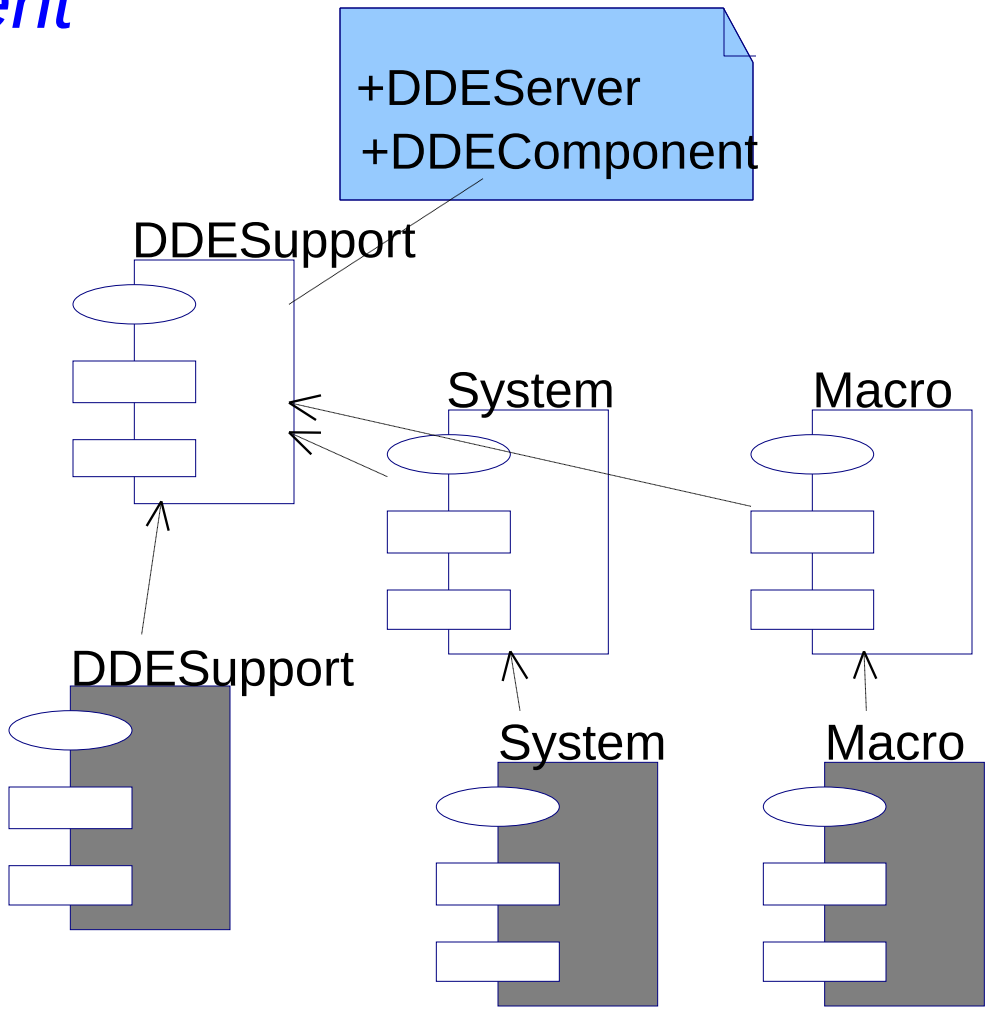
UML PRIMER: DA-C DDE

Sequence



UML primer: DA-C DDE

Component



CASE alati za modeliranje

- Rational Rose
http://www.rational.com/products/rose/index.jtmpl
- Computer Associates Platinum Paradigm Plus
http://www.cai.com/products/platinum/appdev/pplus_ps.htm
- Telelogic Tau UML Suite (COOL:Jex)
http://www.telelogic.com/solution/tools/uml.asp
- TogetherSoft Together
http://www.togethersoft.com
- Advanced Sofware Technologies GDPro
http://www.advancedsw.com/evaluate/evaluate.html
- Rhapsody Modeler
http://www.ilogix.com/modeler

PITANJA