OSNOVE GEOINFORMATIKE

Modelovanje podataka

Modelovanje podataka

- Model sistema je prikaz bitnih funkcija nekog stvarnog (ili zamišljenog) sistema koji predstavlja obeležja sistema u upotrebljivom obliku.
- Modeli se izrađuju u cilju boljeg razumevanja sistema jer pomažu u vizuelizaciji stvarnog ili zamišljenog sistema, opisuju njegovu strukturu i ponašanje, predstavljaju šablon po kojem se sistem može implementirati i dokumentuju sve korake izgradnje sistema.
- Izbor vrste modela utiče na pristup problemu i na način oblikovanja rešenja.
- Ni jedan model sam po sebi nije dovoljan, već je potrebno strukturu i ponašanje sistema posmatrati i modelovati sa više raznih gledišta.

Šta je UML?

- OMG (*Object Management Group*) – organizacija zadužena za brigu o standardizaciji UMLa
- UML (*Unified Modeling Language*) je standardni vizuelni jezik za:
  - Specifikaciju zahteva, analizu, projektovanje i implementaciju softverskih sistema
- Koriste ga poslovni analitičari, softver arhitekte i developeri za:
  - opisivanje, specifikaciju, razvoj i dokumentovanje postojećih ili novih poslovnih procesa, strukture i ponašanja softverskih sistema

Modelovanje podataka

- Za potrebe modelovanja sistema razvijen je jezik za modelovanje sistema UML (Unified Modeling Language).
- U geoprostornom domenu definisan je standard ISO 19103 koji preporučuje upotrebu UML jezika za konceptualne šeme u kontekstu geoprostornih informacija i definiše profil UML-a koji se koristi kod svih konceptualnih šema.
- UML predstavlja alat za vizuelizaciju, opis, izgradnju i dokumentovanje softverske podrške kod analize i izrade prvenstveno softverskog rešenja.
- UML predstavlja rečnik i pravila za izražavanje znanja o modelovanom sistemu
- UML je formalni jezik namenjen za formalno specificiranje sistema (mogućih stanja i ponašanja) , modelovanje sistema, analizu sistema, dokumentovanje sistema, vizuelizaciju sistema i razvoj (analizu i projektovanje) softverskih sistema.

Modelovanje podataka

- Modeli pomažu projektnom timu u prikazu sistema koji se formira i omogućavaju evidentiranje raznovrsnih ograničenja na sistemu
- UML olakšava razmenu informacija među učesnicima projekta budući da je usmeren na višestruke komunikacije, npr. između korisnika i razvojnog tima, konstruktora sistema i projektanata baze podataka, članova razvojnog tima koji rade na različitim delovima sistema
- Za ove potrebe UML nudi skup dobro određenih grafičkih prikaza i dijagrama, razumljivih i projektantima sistema i korisnicima sistema
- Tehnički opis UML-om je precizan, nedvosmislen i potpun, pa na taj način omogućava razumevanje strukture i ponašanja sistema uz smanjeni rizik pogrešnog tumačenja
- Modele je moguće definisati na osnovu postojećih sistema u cilju održavanja i proširivanja funkcionalnosti sistema

Modelovanje podataka

- Cilj modelovanja nekog stvarnog sistema je izrada programskog kôda
- Mnogobrojni koncepti UML-a odgovaraju direktno i indirektrno konceptima programskih jezika (Java, C++, C#), kao i konceptima u modelovanju baza podataka
- Izrada sistema se može vršiti tako da se iz modela generiše programski kôd (unapred - forward engineering)
- Dijagrami za modelovanje se razvijaju alatima za vizualno modelovanje kao što su Enterprise Architect, Power Designer, IBM Rational Rose i drugi.

Modelovanje podataka

- Drugi način izrade sistema se odnosi na izradu (ili obnovu) modela iz programskog kôda (unazad - reverse engineering).
- U slučaju unapređivanja postojećih sistema najčešće se koristi postupak kombinacije pomenuta dva načina unapred i unazad (round-trip engineering).
- Postupkom unazad se dobijaju osnovni modeli sistema koji se potom dorađuju da obuhvate željene dodatne funkcije, a zatim se na osnovu novokreiranih modela postupkom unapred generiše programski kôd

Modelovanje podataka

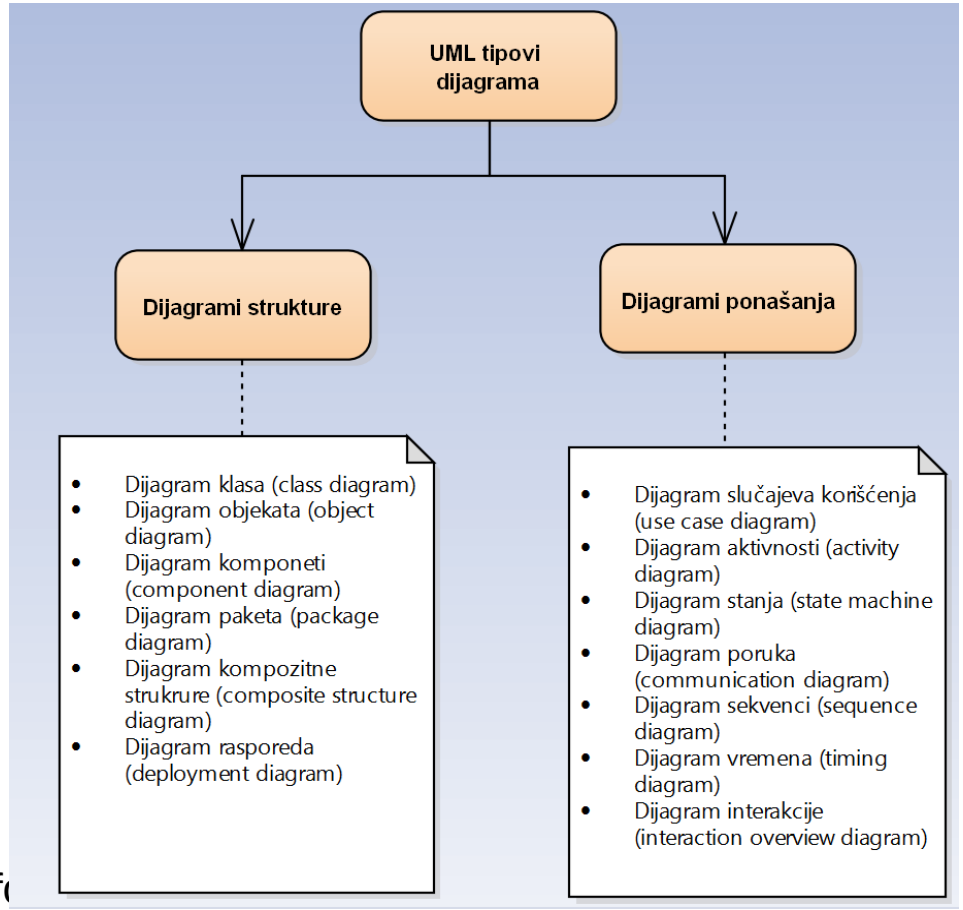
- UML je zasnovan na principima objektne orijentacije
- Polazni koncepti UML-a su klasa, objekat, veza i poruka
  - **Klasa** predstavlja skup objekata koji imaju iste osobine (atibute) i funkcionalnosti (operacije), istu semantiku i zajedničke veze sa drugim objektima. Klasa predstavlja model skupa entiteta realnog sistema koji imaju zajedničku osobinu.
  - **Objekat** je pojava posmatrane klase i ima određenu ulogu u sistemu (skup svih studenata, profesora). On opisuje pojedinačan predmet, entitet, bilo stvaran ili apstraktan, sa dobro definisanom ulogom u domenu problema. On je određen svojim stanjem, ponašanjem i identitetom (student Petar Petrović, fakultet tehničkih nauka...). Objekat predstavlja model konkretnog entiteta realnog sistema.

Modelovanje podataka

- Polazni koncepti UML-a su klasa, objekat, veza i poruka
  - **Veza** predstavlja model mogućih odnosa između klasa objekata i/ili samih objekata. Osnovne vrste veza su:
    - *asocijacija* - semantička zavisnost između klasa (student-profesor, predmet-profesor). Specijalne vrste asocijacije su *agregacija* - predstavija vezu celina-deo izmedu pojava datih klasa ili objekata (fakultet, odsek, smer) i *kompozicija* – modeluje odnos celina-deo tako da je celina isključivi i jedini vlasnik njenih delova (država-grad).
    - *generalizacija* - veza između klasa u kojoj jedna klasa ili objekat deli strukturu i/ili ponašanje definisano u jednoj ili više klasa ili objekata (student, studentNaBudzetu, studentSamofinansiranje).
    - *zavisnost* - modelira takav odnos između klasa objekata, u kojem koncepti jedne klase zahtevaju prisustvo druge klase, tj. koncepti jedne klase referenciraju koncepte druge klase (klijent-server).
    - *realizacija* - modeluje takav odnos između klasa objekata, u kojem koncepti jedne klase služe za implementaciju koncepata druge klase, pa tako postoji klasa implementacije koja implementira koncepte druge klase koja se naziva tipska klasa odnosno klasa, čiji su koncepti implementirani putem druge klase.
  - **Poruka** predstavlja oblik moguće interakcije između objekata. Predstavlja poziv operacije klase, nad jednim objektom te klase, iz neke druge metode. Pozvana metoda operacije "opslužuje" poruku.

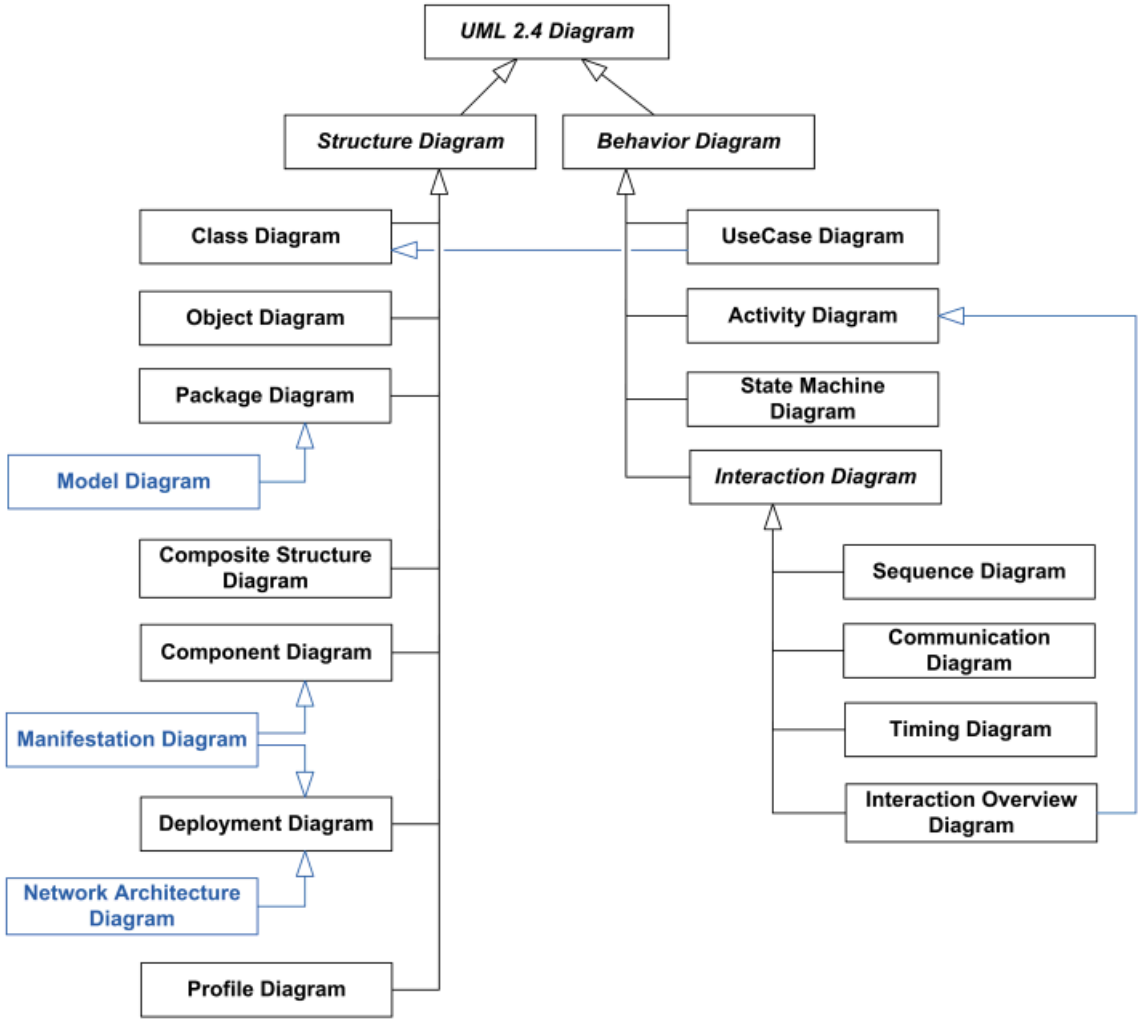
Tipovi dijagrama

- Osnovna podela UML dijagrama se vrši na osnovu toga da li opisuju statičku strukturu sistema ili ponašanje sistema



Objektno-orijentisano modelovanje pomoću UML jezika

- UML sadrži skup dijagrama za opis strukture i dinamike svih vrsta sistema



Tipovi dijagrama

- Dijagrami strukture su:
  - Dijagram klasa (*class diagram*) opisuje statički pogled na model kroz klase i veze između njih.
  - Dijagram objekata (*object diagram*) je specijalni slučaj dijagrama klasa i naglašava vezu između instaci klasa u nekom trenutku vremena.
  - Dijagram komponenti (*component diagram*) pokazuje komponete sistema, osnovne gradivne elemente sistema.
  - Dijagram paketa (*package diagram*) se koristi za podelu modela u logičke celine i prikaz veza između tih celina.
  - Dijagram rasporeda (*deployment diagram*) modeluje fizički izgled sistemskog mapiranja softverskih elemenata na hardver u kojem se oni izvršavaju i njihovu komunikaciju.
  - Dijagram kompozitne strukture (*composite structure diagram*) pokazuje unutrašnju strukturu klase i saradnje koje je omogućavaju i opisuju.

Tipovi dijagrama

- Dijagrami ponašanja su:
  - Dijagram slučajeva korišćenja (*use case diagram*) služi za specificiranje poslovne funkcije sistema. Slučaj korišćenja predstavlja opis (deklaraciju) niza akcija koje sistem izvršava prilikom realizacije poslovne funkcije, a koja se pokreće na zahtev korisnika.
  - Dijagram aktivnosti (*activity diagram*) služi da opiše redosled izvršavanja aktivnosti u okviru jednog slučaja korišćenja.
  - Dijagram stanja (*state machine diagram*) pokazuje ponašanje sistema kao odgovor na neki spoljni stimulans. Prikazuje stanja sistema i način kako sistem prelazi iz jednog u drugo stanje.
  - Dijagram poruka (*communication diagram*) modeluje interakcije između objekata ili delova u smislu sekvenciranih poruka.
  - Dijagram sekvenci (*sequence diagram*) pruža grafički prikaz interakcija objekata tokom vremena. Jedan dijagram sekvence obično predstavlja jedan scenario slučaja korišćenja ili tok događaja.
  - Dijagram vremena (*timing diagram*) se koristi za prikaz promene stanja ili vrednosti jednog ili više elemenata modela tokom vremena.
  - Dijagram interakcije (*interaction overview diagram*) koristi kontrole iz dijagrama aktivnosti kako bi se kontrolna logika postavila oko dijagrama nižih nivoa.

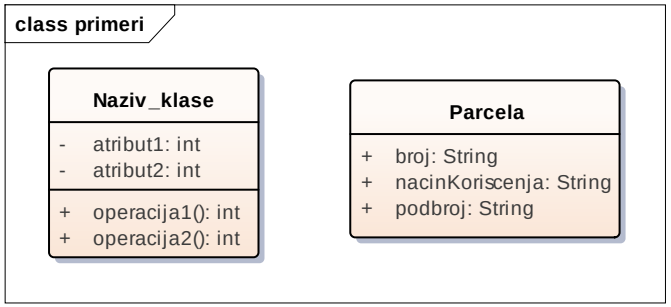
Dijagram klasa

- Dijagram klasa opisuje tipove objekata nekog sistema i različite vrste statičkih odnosa koji postoje između tih objekata. Dijagram klasa predstavlja grafičku reprezentaciju statičkog pogleda statičkih elemenata. Tehnika modelovanja dijagrama klasa se bazira na objektno-orijentisanim principima. Poseduje najbogatiju notaciju u UML-u.

- Najvažniji elementi UML dijagrama klasa su:
  - Klase
  - Atributi
  - Operacije
  - Odnosi
  - Asocijacije
  - Generalizacije
  - Zavisnosti
  - Realizacija
  - Pravila ograničenja i zapisi

Klasa

- Klasa predstavlja opis skupa objekata koji imaju slične atribute, operacije, odnose i ponašanje.
- Klasa ima jedinstveni naziv i definiše svoje ponašanje korišćenjem operacija.
- Klasa je apstraktni prikaz nekog predmeta, osobe ili događaja, dok je objekat je instanca klase, odnosno stvarni prikaz apstrakcije.
- Grafički se klasa predstavlja pravougaonikom sa određenim delovima koji su namenjeni za smeštanje imena klase, atributa i operacija
- Naziv klase piše se velikim slovom.
- Naziv atributa je kratka imenica ili fraza
- Naziv operacije je glagol ili glagolska fraza.



Klasa

- Vidljivost atributa i operacija predstavlja pravo pristupa, čiji se znak piše ispred atributa/operacije.

- Vidljivost atributa se odnosi na javni, privatni, zaštićeni ili implementacioni domen.
  - Ako stoji znak + za javni domen u tom slučaju pristup nije ograničen.
  - Ako je znak – za privatni domen, to znači da je pristup ograničen na članove iste klase i podklase.
  - Zaštićeni domen se označava sa # i odnosi se na prošireno pravo pristupa i na klase izvedene iz date klase.
  - Implementacioni domen se označava sa ? i ukazuje da opseg vidljivosti tog trenutka nije definisan, već da će se definisati u okviru implementacije.
  - Jedinstvena identifikacija klase je realizovana označavanjem atributa za ključ uvođenjem znaka * ispred atributa. Kod nekih alata ovo se vizuelno prikazuje sa {id} iza definicije atributa.

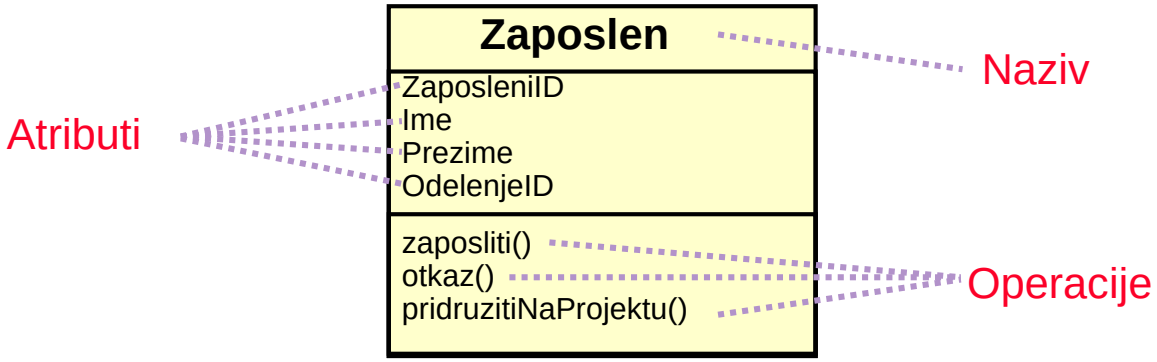
- Za atribut se definiše i tip atributa. Tip atributa može biti ceo broj (integer), realni broj (real), karakter (string), datum (date)…

- Operacija klase se definiše nazivom operacije, vidljivošću i povratnim tipom. Povratni tip je tip rezultata koji operacija vraća. Primer operacije: +*operacija(parametar): povratni tip*.

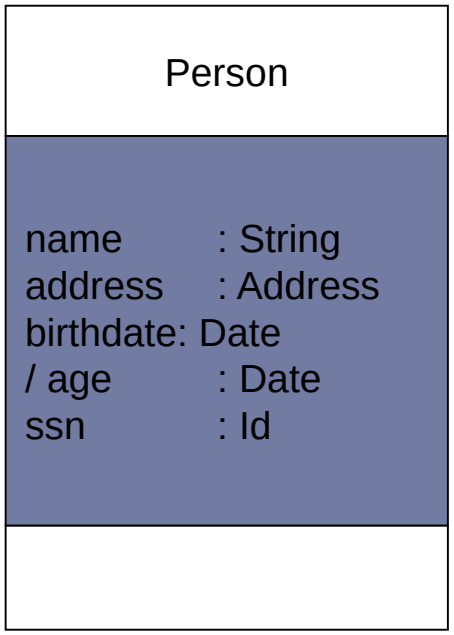
Šta je klasa?

- Klasa opisuje grupu objekata sa
  - sličnim osobinama (atributima),
  - zajedničkim ponašanjima (operacijama/metodama),
  - zajedničkim relacijama ka drugim objektima,
  - jedinstveno značenje ("semantika").

- Primer:
  - Zaposleni ima ID, ime i prezime, odelenjeID gde radi; jedan zaposleni se može zaposliti, dobiti otkaz ili može raditi na projektima.



Atributi klasa



- Atributi su obično navedeni u formi:

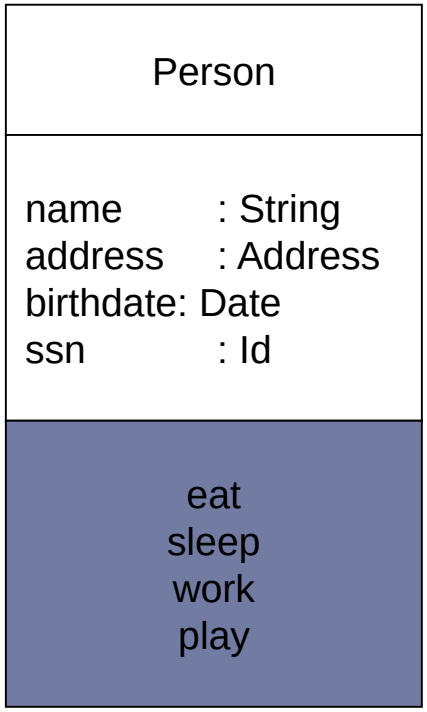
  attributeName : Type

- Izvedeni (*derived*)atribut je onaj koji se može izračunati iz drugih atributa, ali on u stvari ne postoji.
  - Na primer, čovekova starost može se izračunati od datuma njegovog rođenja.

- Izvedeni atribut označen je sa '/':

  / age : Date

Operacije (metode) klasa



Operacije opisuju ponašanje klase i pojavljuju se u trećem delu klase.

Operacije (metode) klasa

PhoneBook
newEntry (n : Name, a : Address, p : PhoneNumber, d : Description) getPhone (n : Name, a : Address) : PhoneNumber

Možete odrediti operaciju navođenjem njenog potpisa: navođenje imena, tipa i difoltne vrednosti svih parametara, a u slučaju funkcija, tip povratka.

Veze između klasa

- Klase same po sebi nisu dovoljne da opišu neki sistem, već je neophodno definisati relacije između klasa jer one zapravo čine osnovu strukture modela.

- Mogu se razlikovati relacije
  - **asocijacije** (relacije udruživanja),
  - **agregacije** i **kompozicije** (relacije sastavljanja)
  - **zavisnosti,**
  - **generalizacije** (relacije uopštavanja).

Asocijacija

- specificira da je objekat jedne klase u vezi sa objektom druge (možda iste) klase
- Vizuelno se predstavlja linijom između dve klase
- objekti na kraju jedne asocijacije mogu da "prepoznaju" objekte na kraju neke druge asocijacije
- Primer: "nosilac prava ima određeno pravo nad parcelom"

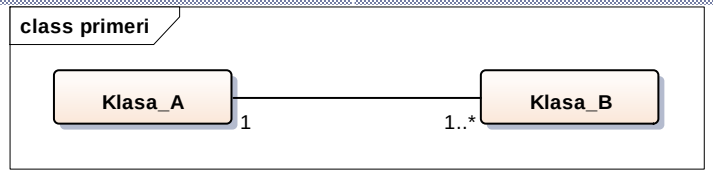
Asocijacija

- Da bi se objasnilo značenje, asocijacija može da dobije svoje ime
- ime se postavlja na sredinu linije koja predstavlja asocijaciju. Obično je u pitanju glagol ili glagolska fraza
- Uloga je na kraju asocijacije, na mestu gde se spaja sa klasom; ukazuje na ulogu klase koja se nalazi na kraju putanje asocijacije; obično je u pitanju imenica.
- Naziv uloge je obavezan za rekurzivne asocijacije.

Asocijacija

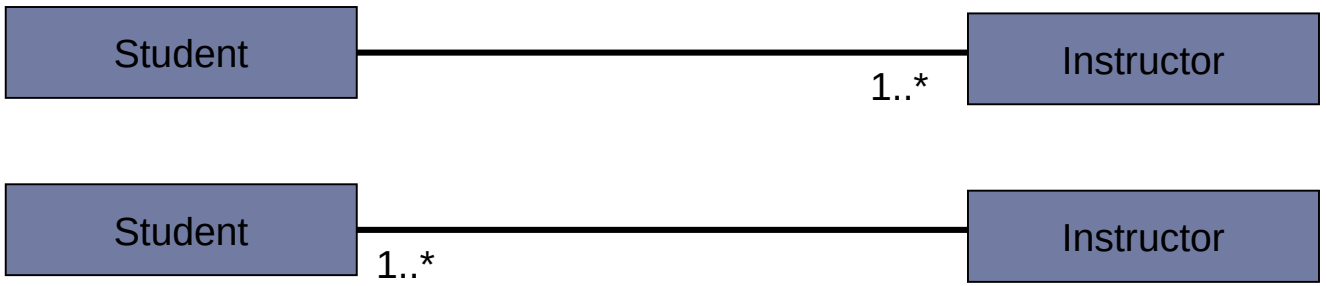
- Na oba kraja asocijacije se definiše višestrukost ili multiplikativnost
- Višestrukost označava broj instanci klase i ukazuje na to da li je asocijacija obavezna ili ne
- obezbeđuje donju i gornju granicu broja instanci

Pojava klase A vezana je za jednu i samo jednu pojavu klase B	**1**
Pojava klase A vezana je za nijednu ili jednu pojavu klase B	0..1
Pojava klase A vezana je za nijednu ili više pojava klase B	*
Pojava klase A vezana je za jednu ili više pojava klase B	1..*
Pojava klase A vezana je za tri do pet, sedam ili petnaest pojava klase B	3..5, 7, 15
Pojava klase A vezana je za tri ili više pojava klase B	3..*



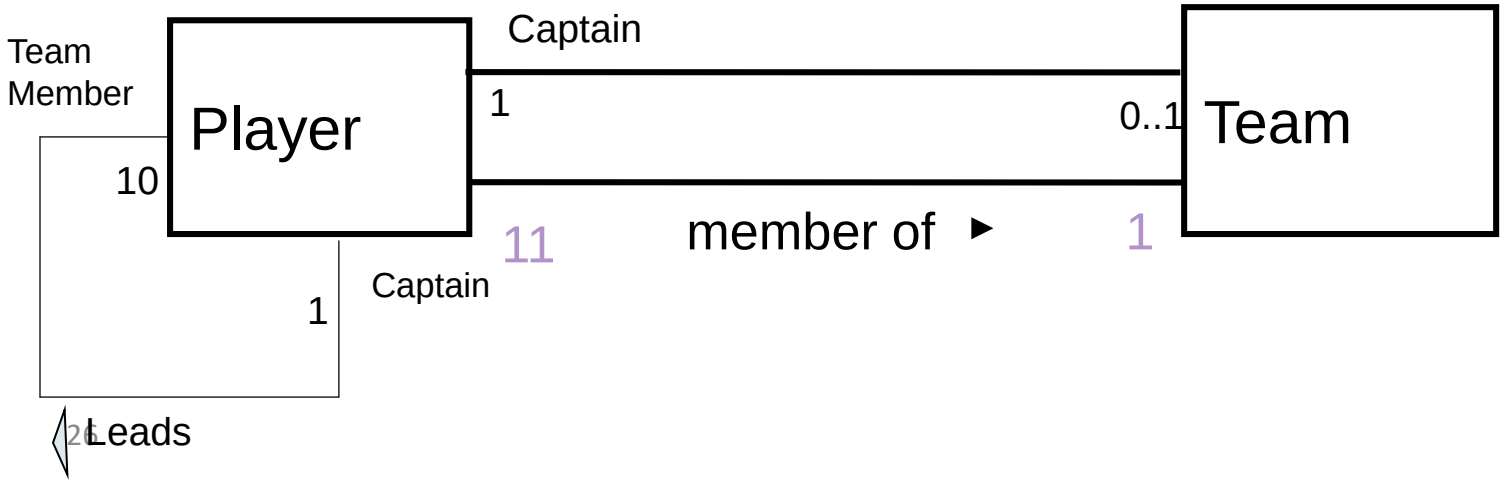
Multiplikativnost

- Multiplikativnost je broj objekata klase koji se može pridružiti jednom objektu druge klase
- Beleži se na oba kraja relacije
- Multiplikativnost odgovara na dva pitanja:
  - Da li je asocijacija obavezna ili opciona? – ako je nula onda je takva asocijacija opciona
  - Koji je minimalni, a koji maksimalni broj instanci koje mogu biti povezane ka drugoj instanci?



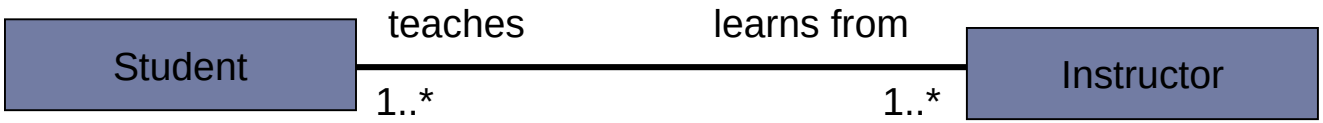
Association - Multiplicity

- A cricket team has 11 players. One of them is the captain.
- A player can play only for one Team.
- The captain leads the team members.

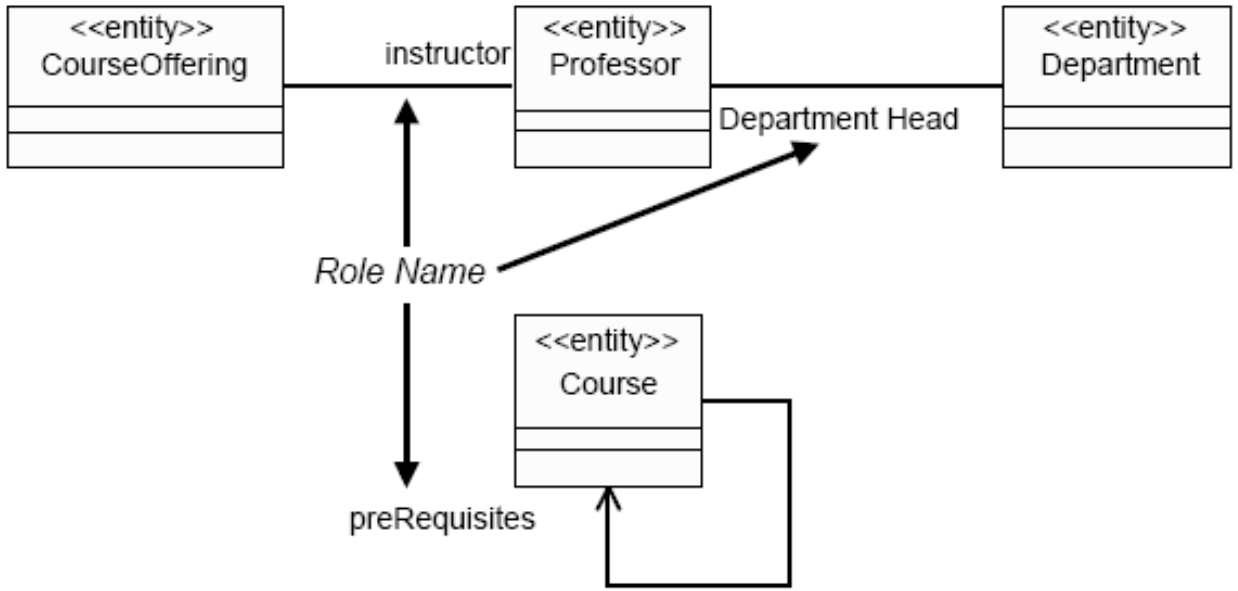


Šta su role (uloge)?

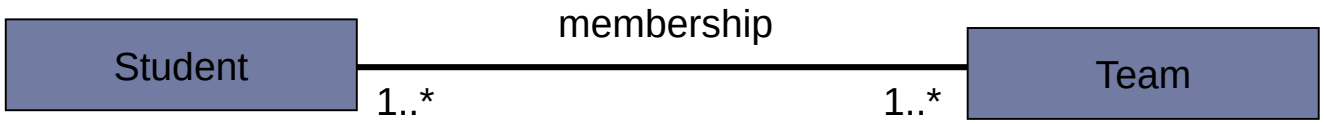
- Asocijacije sadrže neku ulogu (rolu) u relaciji između klasa
  - Uloga ili rola se ispisuje na krajevima linije asocijacije
  - Rola mora imati naziv (obično imenica)



Primer uloga

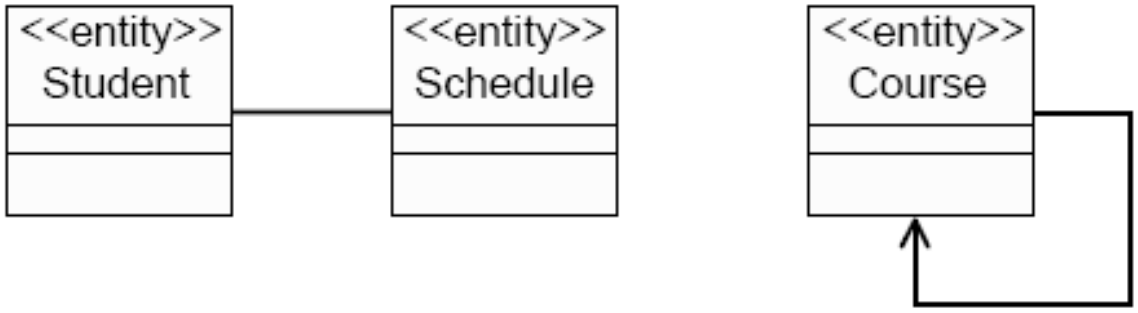


Naziv asocijacija



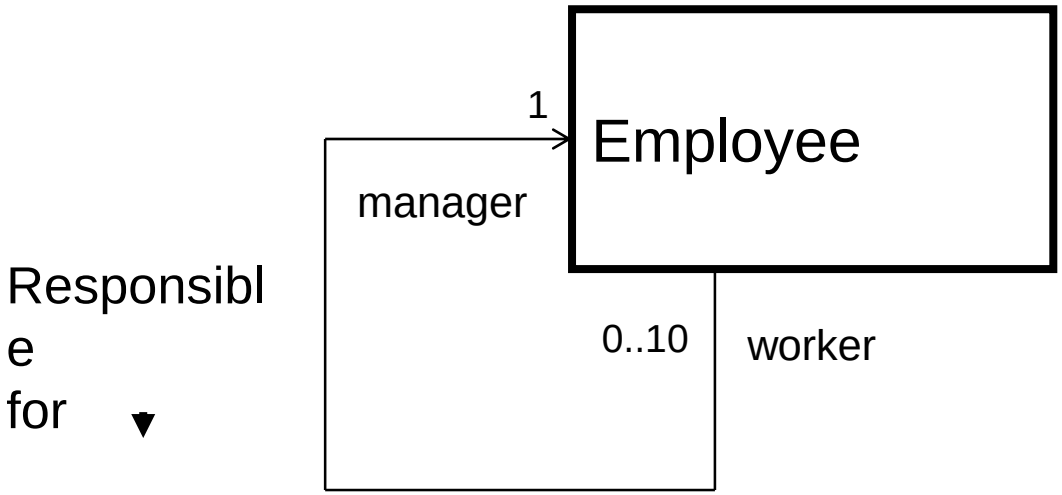
Rekurzivna asocijacija

- Ponekad klasa ima asocijaciju na samu sebe - uglavnom označava da jedna instanca klase ima asocijaciju ka drugoj instanci iste klase

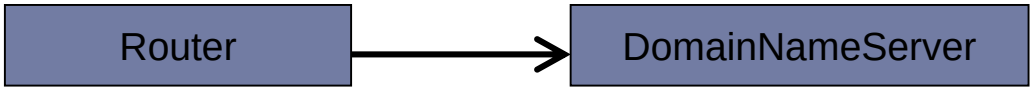


Primer rekurzivne asocijacije

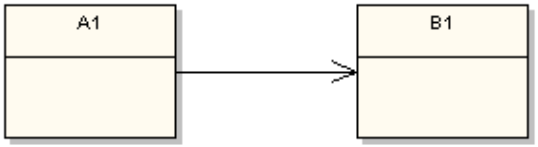
- A Company has Employees.
- A single manager is responsible for up to 10 workers.



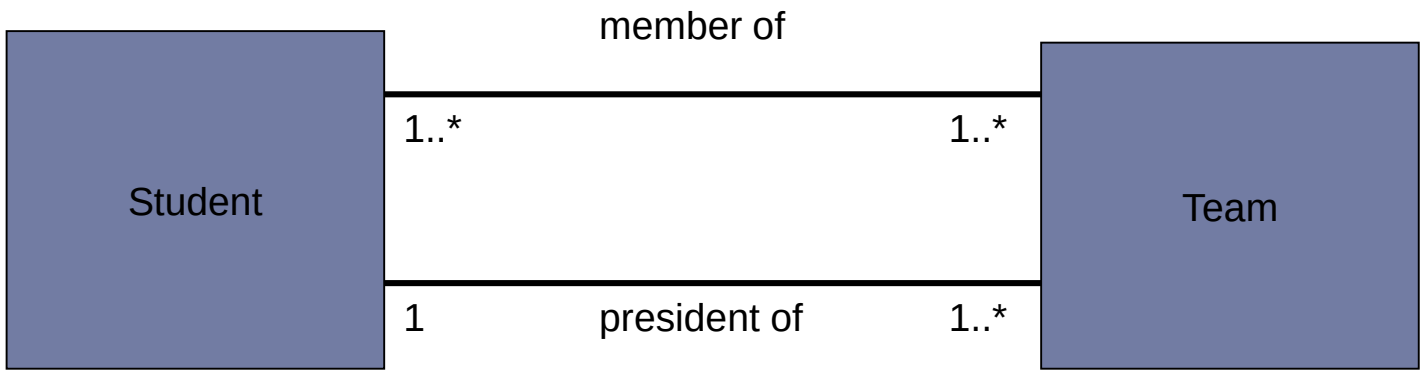
Usmerena asocijacija
(*Directed Association*)



- Usmerena asocijacija označava da se komunikacija između dva objekta odvija samo u jednom pravcu.
- Strelica ukazuje na relaciju u jednom pravcu
- Klasa A1 koristi i sadrži instancu klase B1, ali B1 ne sadrži instance klase A1.

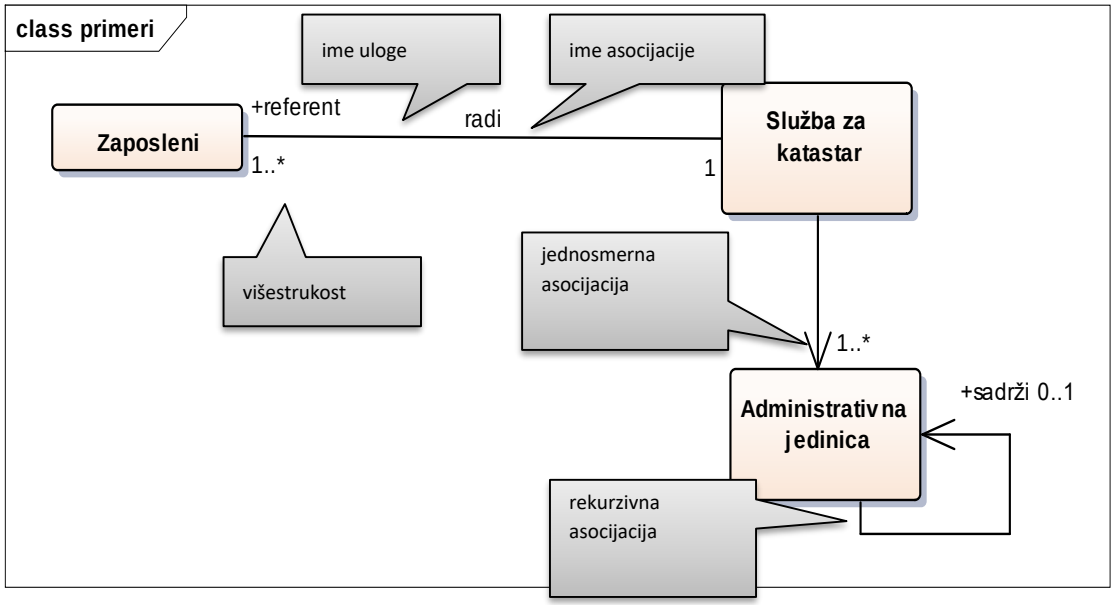


Dvostruke (*dual*) asocijacije



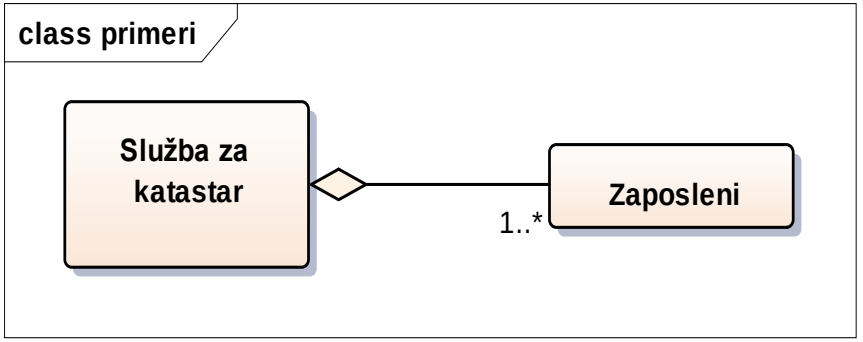
Asocijacija

- Relacija udruživanja pretpostavlja dvosmernost odnosa, odnosno da klase znaju jedna o drugoj.
- Ukoliko je neophodno omogućiti da samo jedna klasa zna o postojanju druge klase može se koristiti jednosmerna asocijacija. Usmerenost odnosa se označava strelicom.
- klasa *Služba za katastar* vidi instance klase *Administrativna jedinica*, dok instanca klase *Administrativna jedinica* ne zna za instance klase *Služba za katastar*
- Poseban slučaj asocijacije je rekurzivna asocijacija. To je asocijacija koja povezuje klasu samu sa sobom.
- jedna administrativna jedinca može hijerarhiski biti povezana sa nijednom ili jednom administrativnom jedinicom.



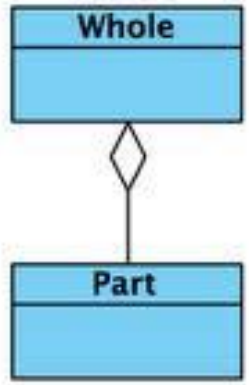
Agregacija

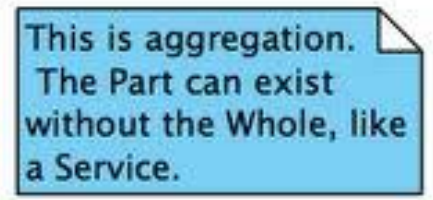
- Agregacija je specijalan oblik asocijacije koji modeluje odnos “celina-deo” između celine i njenih delova.
- Modeluje “je deo-deo od” odnos.
- Kod agregacije delovi postoje nezavisno od celine
- Agregacija se predstavlja linijom sa belim rombom kod klase koja predstavlja celinu.

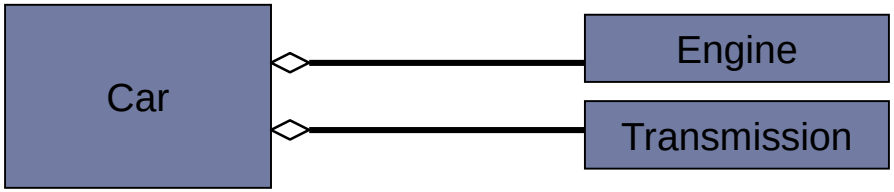


Šta je agregacija?

- Specijalni oblik asocijacije koja modeluje relacije između celine i njenih delova
  - Prazan romb je na strani celine ukazuje na relaciju agregacije
  - Kada je klasa u relaciji agregacije sa samom sobom, to znači da jedna instanca klase se sastoji od drugih instanci iste klase

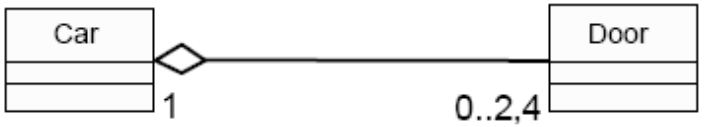




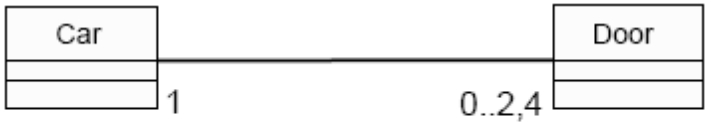


Asocijacija ili agregacija?

- Ukoliko su dva objekta **usko povezana relacijom celina-deo,** onda je relacija agregacija
  - Ukoliko modelujete **prodavnice kola**, onda relacija između kola i vrata treba da bude modelovana kao agregacija, jer kola uvek dolaze sa vratima, a vrata se nikad samostalno ne prodaju



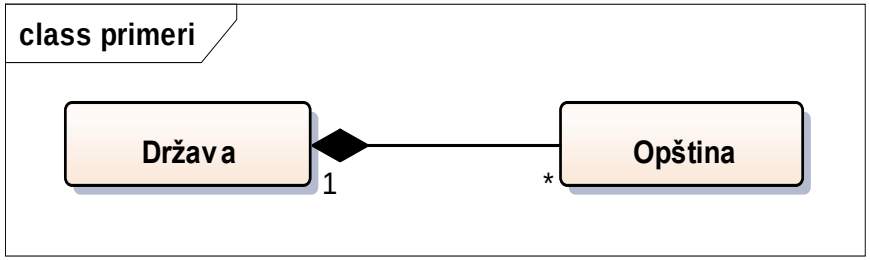
- Ukoliko su dva **objekta nezavisna** onda je relacija asocijacija
  - Ukoliko modelujete **prodavnicu auto delova**, onda relacija između kola i vrata može da bude asocijacija, jer onda telo kola može da se pojavi nezavisno od vrata



When in doubt use association

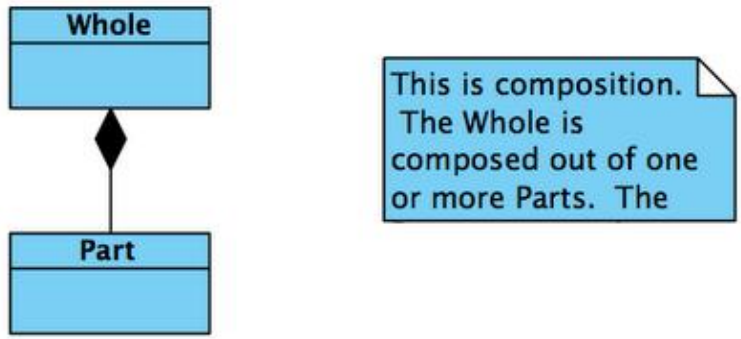
Kompozicija

- Kompozicija predstavlja jači oblik agregacije.
- Kompozicija takođe modeluje odnos "celina-deo" s tim da je celina isključivi i jedini vlasnik njenih delova
- Deo objekta može da pripada jednoj celini.
- Višestrukost sa strane celine mora biti nula ili jedan.
- Životni vek dela isključivo zavisi od celine.
- Kompozicija se predstavlja linijom sa crnim rombom kod klase koja predstavlja celinu.

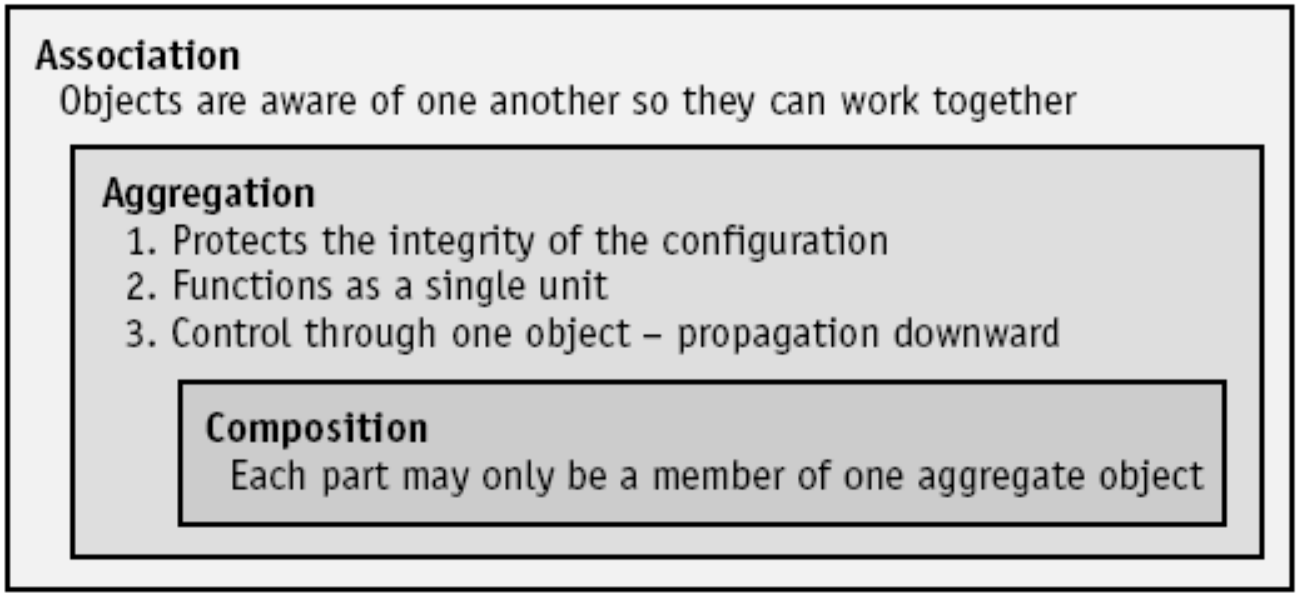


Kompozicija

- Relacija kompozicije je slična relaciji agregacije samo što se kod uništavanja objekta uništava i klasa koja je deo tog objekta



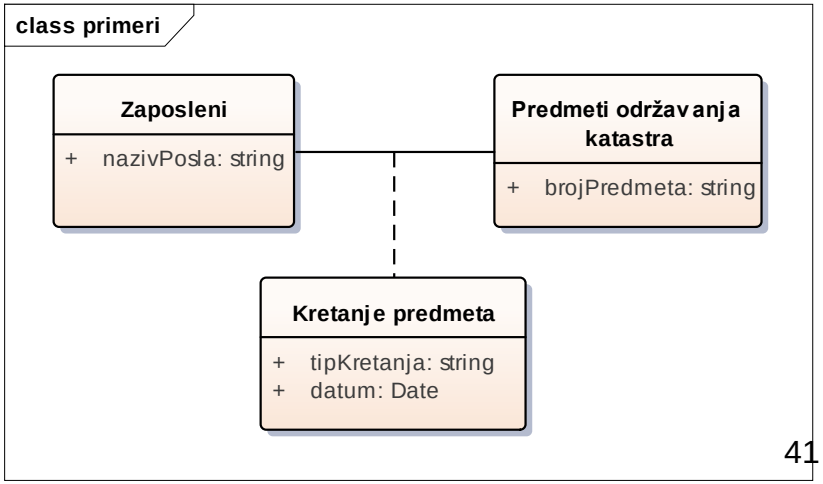
Odnos relacija



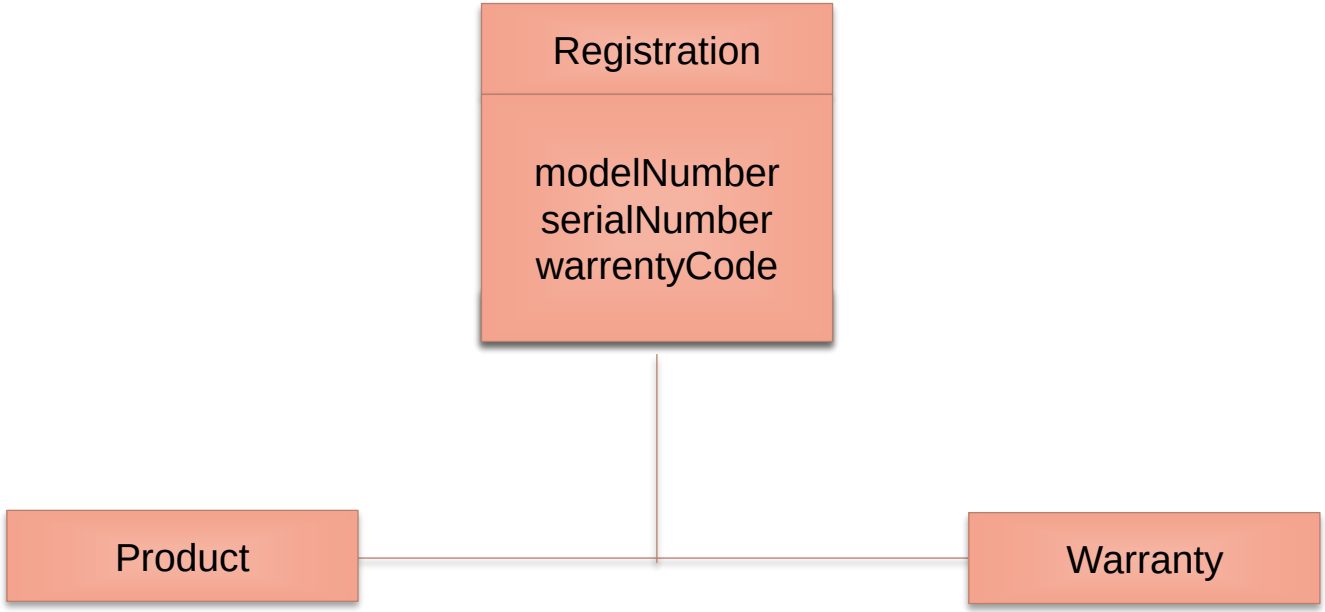
Klasa asocijacije

- Klasa asocijacije je klasa koja dozvoljava da asocijacija između dve klase ima operacije i attribute

- primer dodeljivanja predmeta za provođenje promena u katastru nepokretnosti zaposlenom: jednostavna asocijacija između dve klase nije dovoljna jer zaposleni vrši konkretan posao na predmetu koji je definisan tipom kretanja i datumom. Kako je ovo složen entitet jer sadrži detalje koji ne pripadaju ni klasi *Zaposleni* ni klasi *Predmeti održavanja katastra*, definše se nova klasa *Kretanje predmeta* koja ima ulogu asocijacije

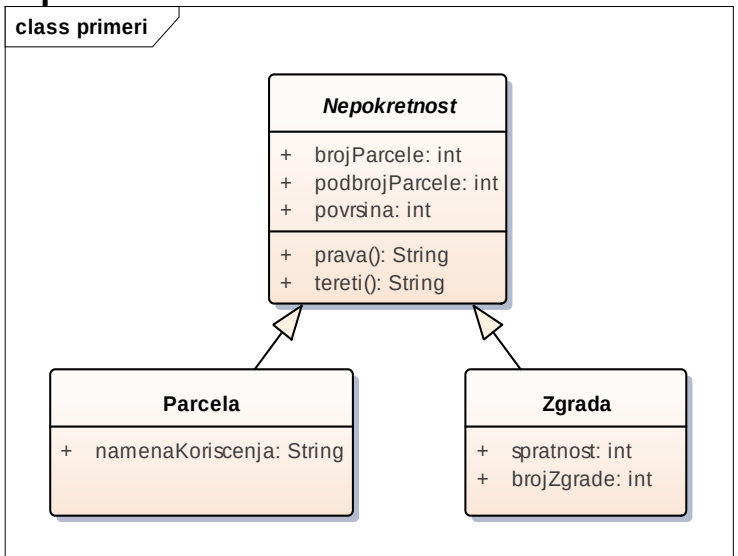


Asocijativne klase (Association Classes)



Generalizacija

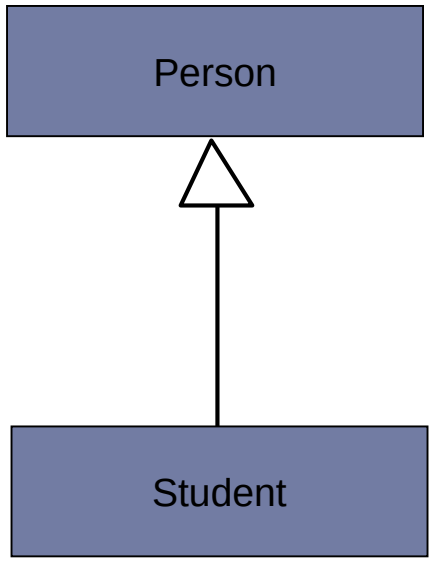
- Relacija generalizacije se odnosi na nasleđivanje i koristi se za prikazivanje odnosa roditelj/dete.
- Podklase (dete) nasleđuju sve atribute, operacije i asocijacije od superklase (roditelja).
- Podklasa može da dodaje atribute i operacije, dodaje relacije, prerađuje (override) nasleđene operacije.
- Grafička predstava generalizacije je puna linija sa velikom strelicom u obliku trougla koja pokazuje na roditelja



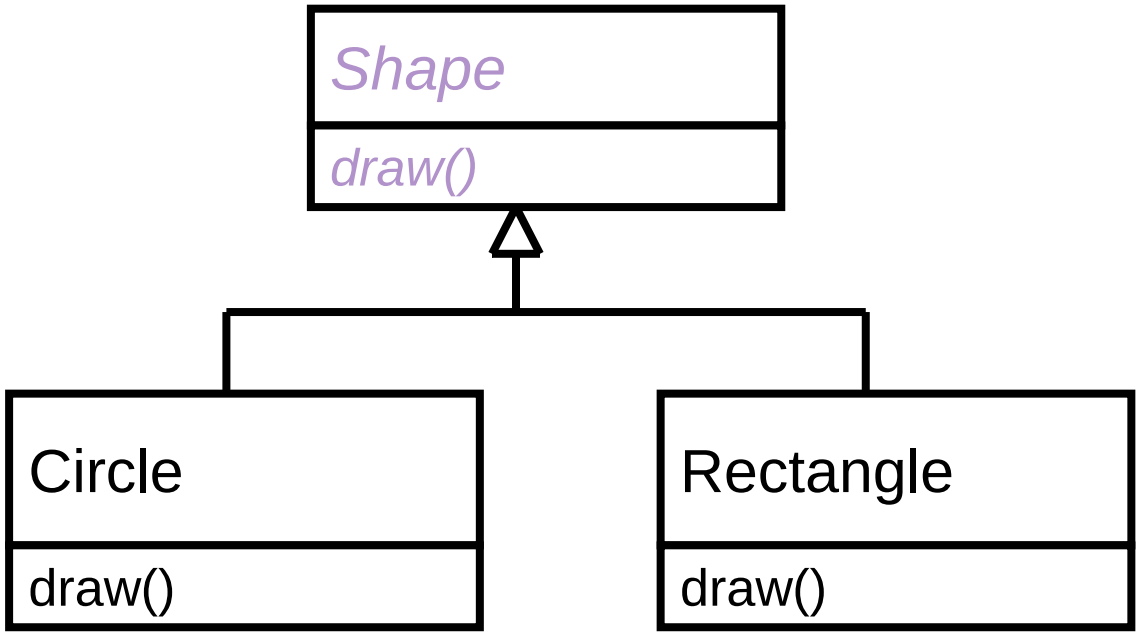
Generalizacija (*Generalization*)

- Generalizacija povezuje nadklasu sa njenim potklasama.

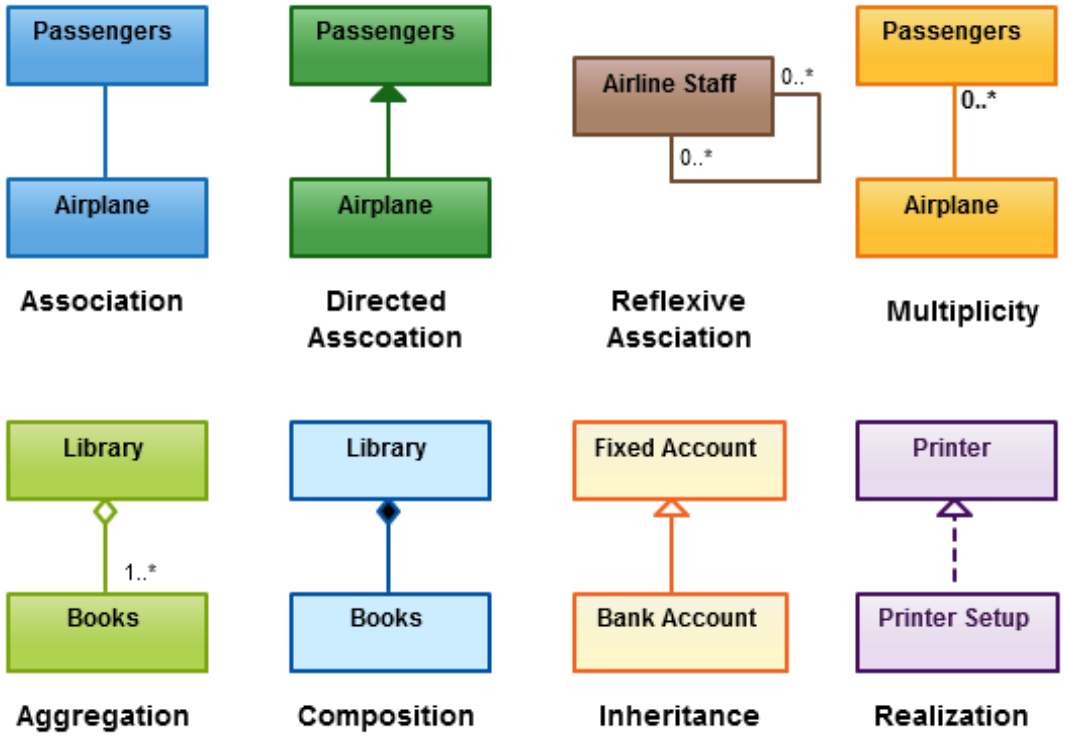
- Potklase nasleđuju atribute i metode nadklase.



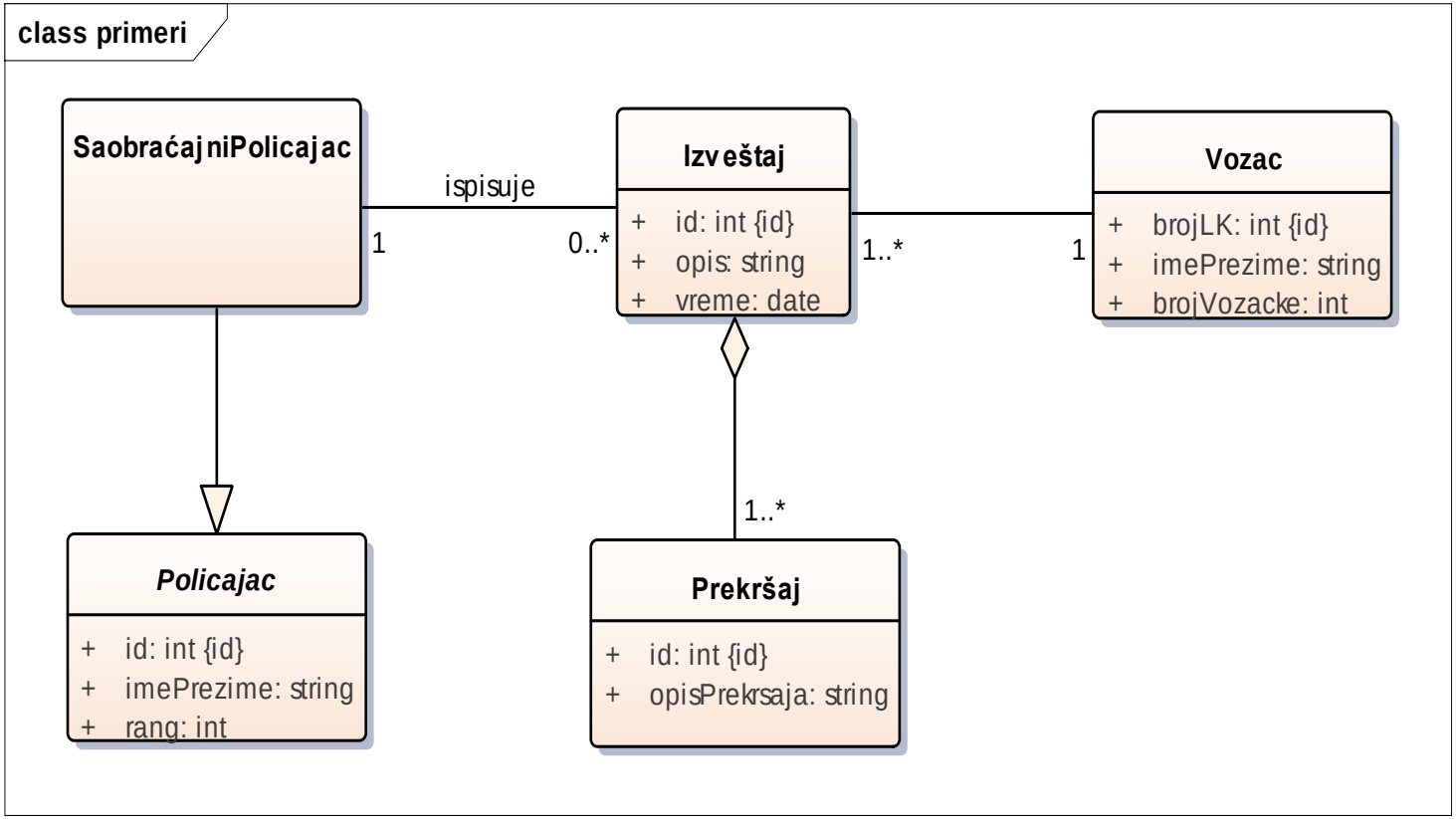
Apstrakne metode (operacije)



Moguće relacije na dijagramu klasa



Primer


class primeri
SaobraćajniPolicajac
ispisuje
1
0..*
Izveštaj
+ id: int {id}
+ opis: string
+ vreme: date
1..*
1
Vozac
+ brojLK: int {id}
+ imePrezime: string
+ brojVozacke: int
Policajac
+ id: int {id}
+ imePrezime: string
+ rang: int
1..*
Prekršaj
+ id: int {id}
+ opisPrekrsaja: string

Primer zadatka

- Nacrtati dijagram klasa za informacioni sistem škole.
  - Škola ima jedan ili više departmana.
  - Departman nudi jedan ili više predmeta.
  - Jedan predmet nudi samo jedan departman.
  - Departman ima instruktore i instruktori mogu raditi za jedan ili više departmana.
  - Student može upisati do 5 predmeta u školi.
  - Instruktori mogu predavati do 3 predmeta.
  - Isti predmet mogu predavati različiti instruktori.
  - Studenti mogu biti upisani u više škola.

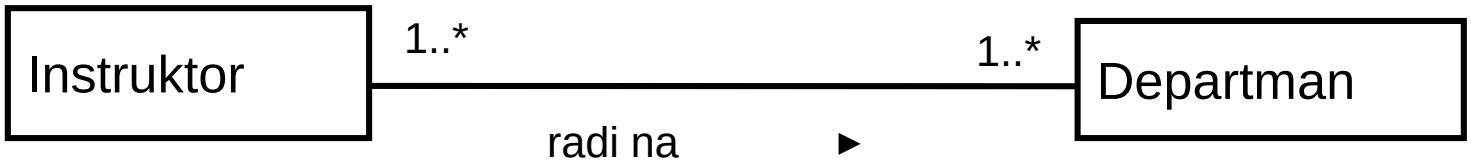
Primer zadatka

- Škola ima jedan ili više departmana.

Departman nudi jedan ili više predmeta.
Jedan predmet nudi samo jedan departman.

Primer zadatka

- Departman ima instruktore i instruktori mogu raditi za jedan ili više departmana.



- Student može upisati do 5 predmeta u školi.

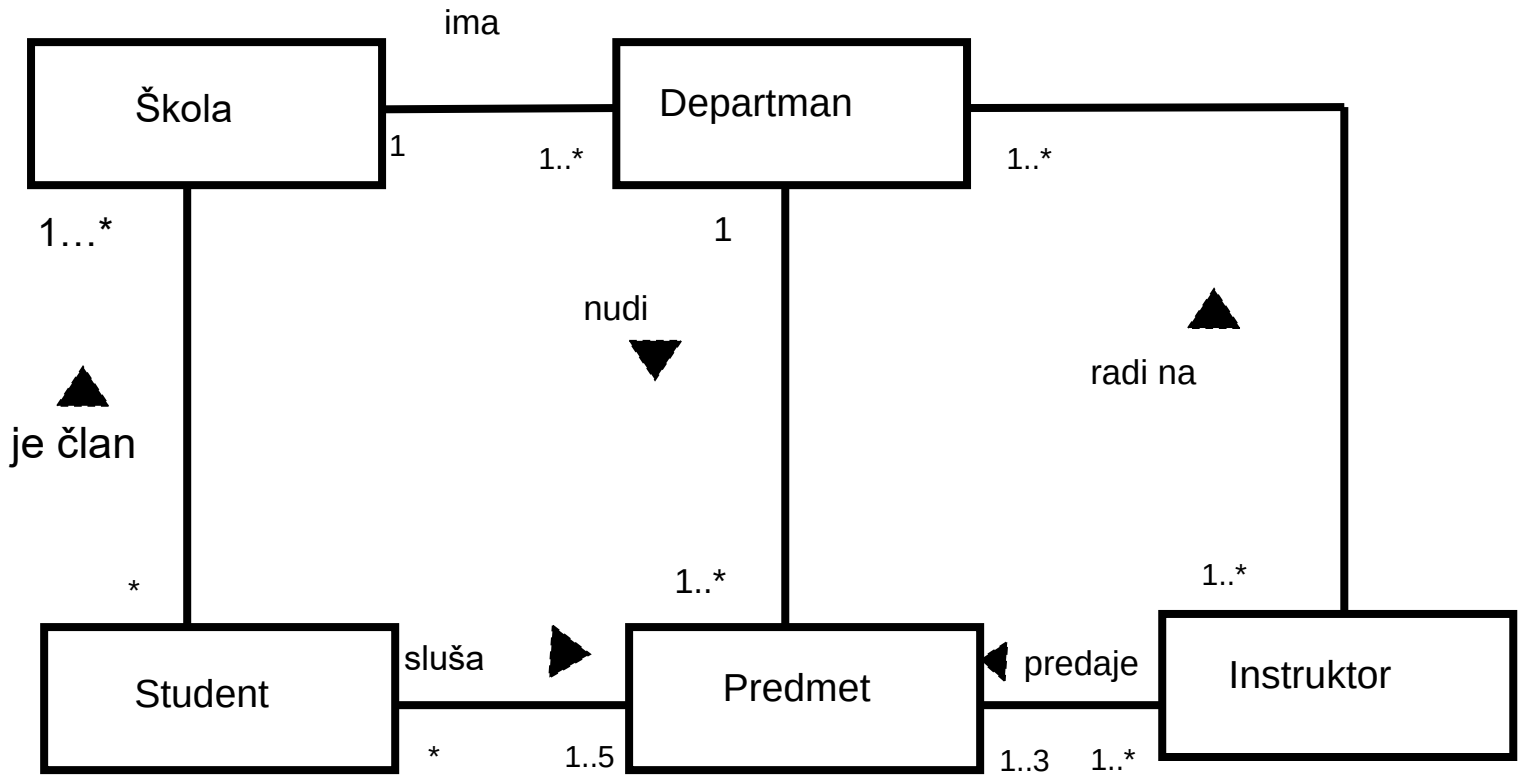
Primer zadatka

- Instruktori mogu predavati do 3 predmeta.
- Isti predmet mogu predavati različiti instruktori.

Primer zadatka

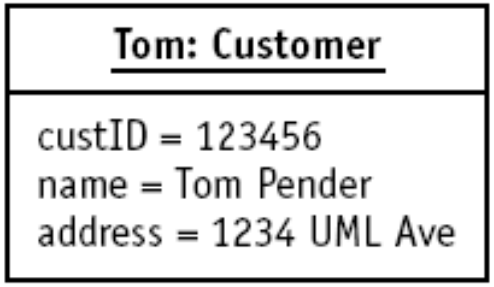
- Studenti mogu biti upisani u više škola.

Primer zadatka

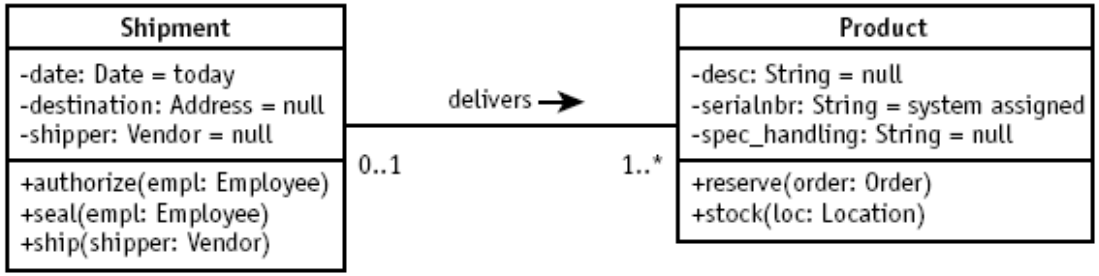


Objektni dijagram

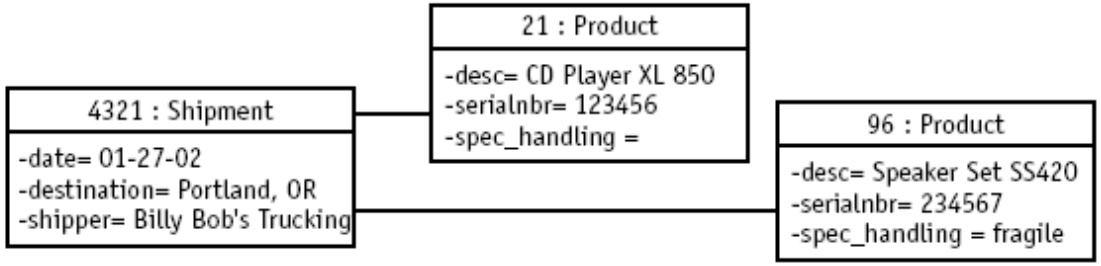
- Objektni dijagram je primarno alat za testiranje i verifikovanje tačnosti dijagrama klasa
- UML notacija objekta:



Poređenje objektnog dijagrama i dijagrama klasa



UML Class notation for the Shipment and Product



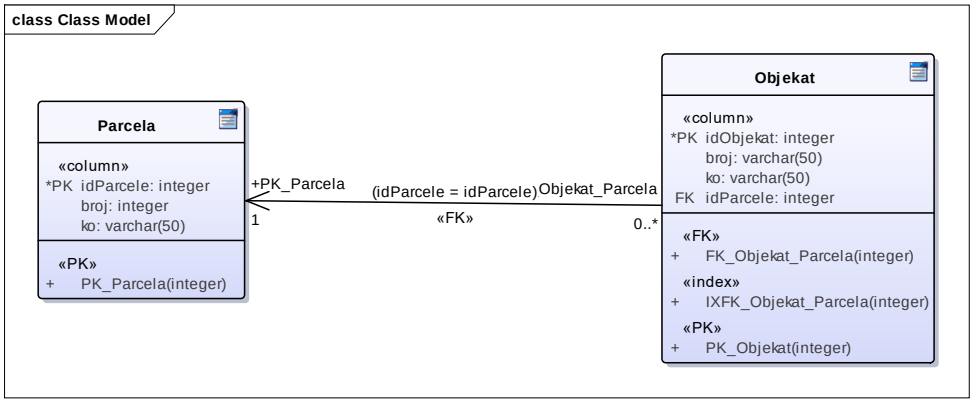
UML Object notation for a Shipment with two Products

Primer

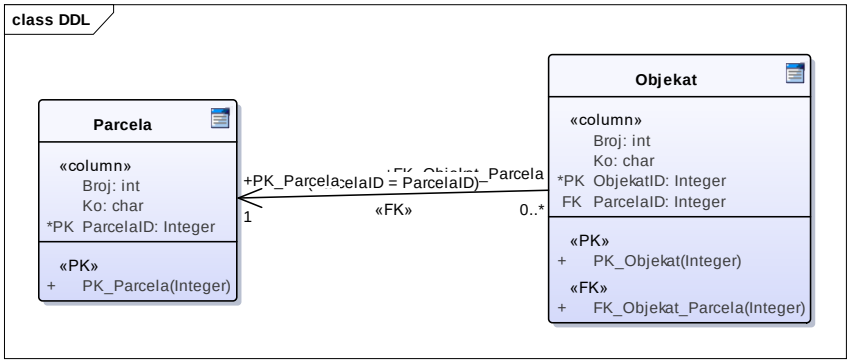
- Napraviti konceptualni model u Enterprise Architect-u za prikaz prostornih entiteta u katastru nepokretnosti

Dijagram klasa

Dijagram tabela



Dijagram tabela dobijen automatskom transformacijom dijagrama klasa



Tipovi podataka se moraju uskladiti sa tipovima nekog SDBMS

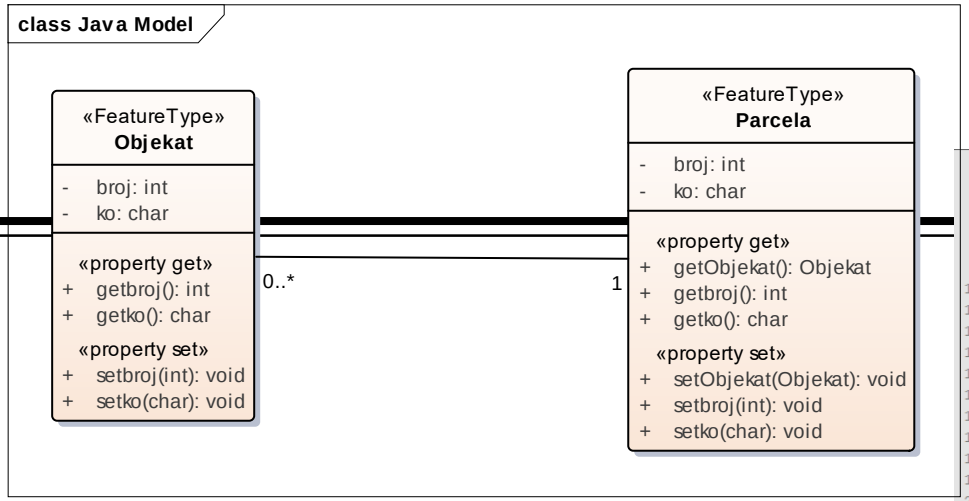
PostgreSQL tipovi podataka

Primer

Rezultat – šema baze podataka

```
/* ---------------------------------------------------- */
/*  Generated by Enterprise Architect Version 12.1      */
/*  Created On : 15-Oct-2020 11:13:05 AM                */
/*  DBMS       : PostgreSQL                          */
/* ---------------------------------------------------- */

/* Drop Tables */

DROP TABLE IF EXISTS Objekat CASCADE
;

DROP TABLE IF EXISTS Parcela CASCADE
;

/* Create Tables */

CREATE TABLE Objekat
(
    idObjekat integer NOT NULL,
    broj varchar(50) NULL,
    ko varchar(50) NULL,
    idParcele integer NULL
)
;

CREATE TABLE Parcela
(
    idParcele integer NOT NULL,
    broj integer NULL,
    ko varchar(50) NULL
)
;

/* Create Primary Keys, Indexes, Uniques, Checks */

ALTER TABLE Objekat ADD CONSTRAINT PK_Objekat
    PRIMARY KEY (idObjekat)
;

CREATE INDEX IXFK_Objekat_Parcela ON Objekat (idParcele ASC)
;

ALTER TABLE Parcela ADD CONSTRAINT PK_Parcela
    PRIMARY KEY (idParcele)
;

/* Create Foreign Key Constraints */

ALTER TABLE Objekat ADD CONSTRAINT FK_Objekat_Parcela
    FOREIGN KEY (idParcele) REFERENCES Parcela (idParcele) ON DELETE No Action ON UPDATE No Action
;

```

Rezultat – Java klase




```java
/**
 * @author sanja
 * @version 1.0
 * @created 15-Oct-2020 11:56:03 AM
 */
public class Objekat {

    private int broj;
    private char ko;

    public Objekat(){

    }

    public void finalize() throws Throwable {

    }
    public int getbroj(){
        return broj;
    }

    /**
     *
     * @param newVal
     */
    public void setbroj(int newVal){
        broj = newVal;
    }

    public char getko(){
        return ko;
    }

    /**
```

```java
 * @author sanja
 * @version 1.0
 * @created 15-Oct-2020 11:56:05 AM
 */
public class Parcela {

    private int broj;
    private char ko;
    private Objekat m_Objekat;

    public Parcela(){

    }

    public void finalize() throws Throwable {

    }
    public Objekat getObjekat(){
        return m_Objekat;
    }

    /**
     *
     * @param newVal
     */
    public void setObjekat(Objekat newVal){
        m_Objekat = newVal;
    }

    public int getbroj(){
        return broj;
    }

    /**
     *
     * @param newVal
     */
    public void setbroj(int newVal){
        broj = newVal;
    }

    public char getko(){
        return ko;
    }

    /**
     *
     * @param newVal
     */
    public void setko(char newVal){
        ko = newVal;
    }
```