

# Modelovanje sistema korišćenjem UML-a

# Šta je UML?

- Unified Modeling Language (UML) je jezik za specifikaciju, vizuelizaciju, konstrukciju i dokumentovanje elemenata softverskog sistema, kao i poslovno modelovanje.

U ~

**Unified:**

- Ujedinjuje sve postojeće notacije

M ~

**Modeling:**

- Koristi se za modelovanje softverskih elemenata

L ~

**Language:**

- Sredstvo komunikacije

# Objektno modelovanje - UML

- ❑ **UML (*Unified Modeling Language*)** - objedinjeni vizuelni jezik za poslovno i softversko modelovanje u svim fazama razvoja i za sve tipove sistema, kao i za generalno modelovanje kojim se definišu statičke strukture i dinamičko ponašanje.
- ❑ Standardni jezik za:
  - ❖ vizuelizaciju
  - ❖ specifikaciju
  - ❖ konstruisanje i
  - ❖ dokumentovanje softverskih sistema
- ❑ UML kombinuje najbolje iz:
  - ❖ Koncepta “Data Modeling” (*Entity Relationships Diagrams*)
  - ❖ Poslovnog modelovanja (*work flow*)
  - ❖ Objektnog i komponentnog modelovanja

# UML

- UML je projektovan kao vrlo fleksibilan i prilagodiv jezik, koji omogućava vrlo različite vrste modelovanja, uključujući:
  - ❖ modele koji olakšavaju razumevanje poslovnih procesa,
  - ❖ odvijanja tokova događaja,
  - ❖ sekvenci upita,
  - ❖ aplikacija,
  - ❖ baza podataka,
  - ❖ arhitektura i drugog.

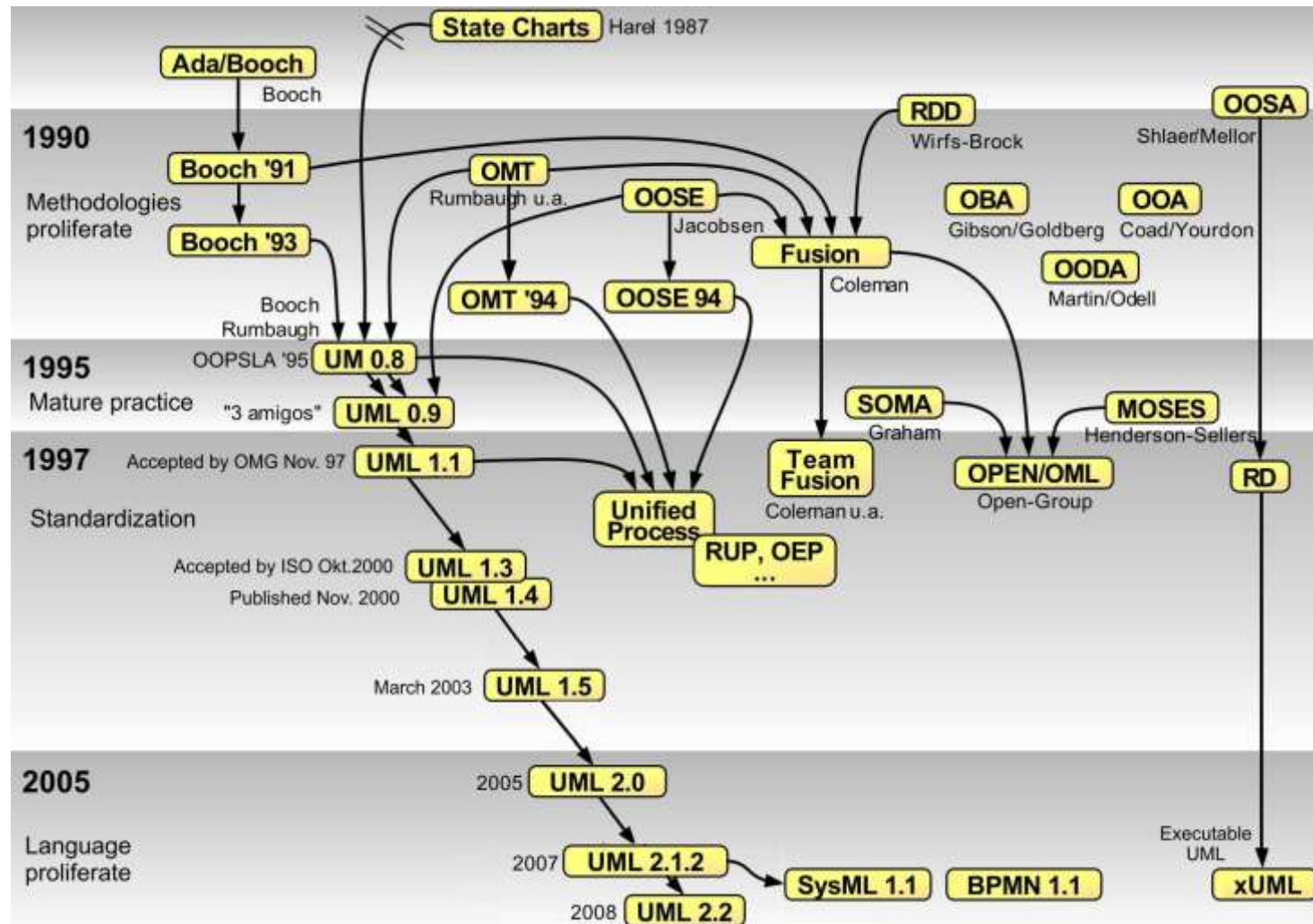
# UML

- ❑ UML je nastao kao rezultat evolucije objektno orijentisanih jezika za modelovanje.
- ❑ Razvila ga je kompanija *Rational Software* objedinjavanjem tri vodeće metode objektno orijentisanog modelovanja:
  - ❖ **Booch** koji je razvio *Grady Booch*,
  - ❖ **OMT** (*Object Modeling Technique*) koji je razvio *Jim Rumbaugh* i
  - ❖ **OOSE** (*Object-Oriented Software Engineering*) koji je razvio *Ivar Jacobson*.

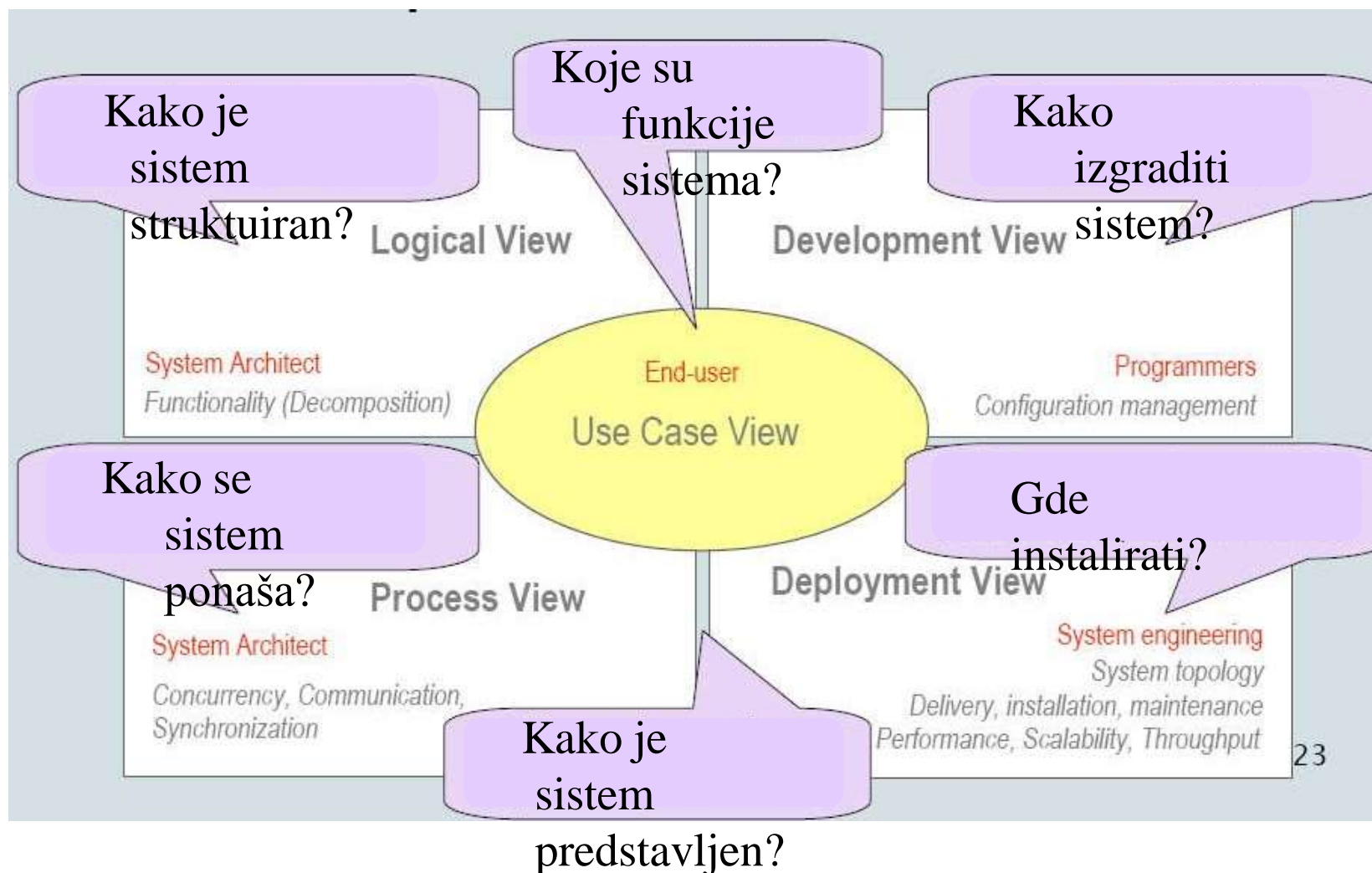
# OMG Formally Released Versions Of UML®

Version	Release Date	URL
<b>2.5.1</b>	<b>December 2017</b>	<a href="http://www.omg.org/spec/UML/2.5.1"><u>http://www.omg.org/spec/UML/2.5.1</u></a>
2.5	June 2015	<a href="http://www.omg.org/spec/UML/2.5"><u>http://www.omg.org/spec/UML/2.5</u></a>
2.4.1	August 2011	<a href="http://www.omg.org/spec/UML/2.4.1"><u>http://www.omg.org/spec/UML/2.4.1</u></a>
2.4	March 2011	<a href="http://www.omg.org/spec/UML/2.4"><u>http://www.omg.org/spec/UML/2.4</u></a>
2.3	May 2010	<a href="http://www.omg.org/spec/UML/2.3"><u>http://www.omg.org/spec/UML/2.3</u></a>
2.2	February 2009	<a href="http://www.omg.org/spec/UML/2.2"><u>http://www.omg.org/spec/UML/2.2</u></a>
2.1.2	November 2007	<a href="http://www.omg.org/spec/UML/2.1.2"><u>http://www.omg.org/spec/UML/2.1.2</u></a>
2.1.1	August 2007	<a href="http://www.omg.org/spec/UML/2.1.1"><u>http://www.omg.org/spec/UML/2.1.1</u></a>
2.0	July 2005	<a href="http://www.omg.org/spec/UML/2.0"><u>http://www.omg.org/spec/UML/2.0</u></a>
1.5	March 2003	<a href="http://www.omg.org/spec/UML/1.5"><u>http://www.omg.org/spec/UML/1.5</u></a>
1.4	September 2001	<a href="http://www.omg.org/spec/UML/1.4"><u>http://www.omg.org/spec/UML/1.4</u></a>
1.3	March 2000	<a href="http://www.omg.org/spec/UML/1.3"><u>http://www.omg.org/spec/UML/1.3</u></a>

# UML versions – 2.2



# Arhitektura softverskih sistema






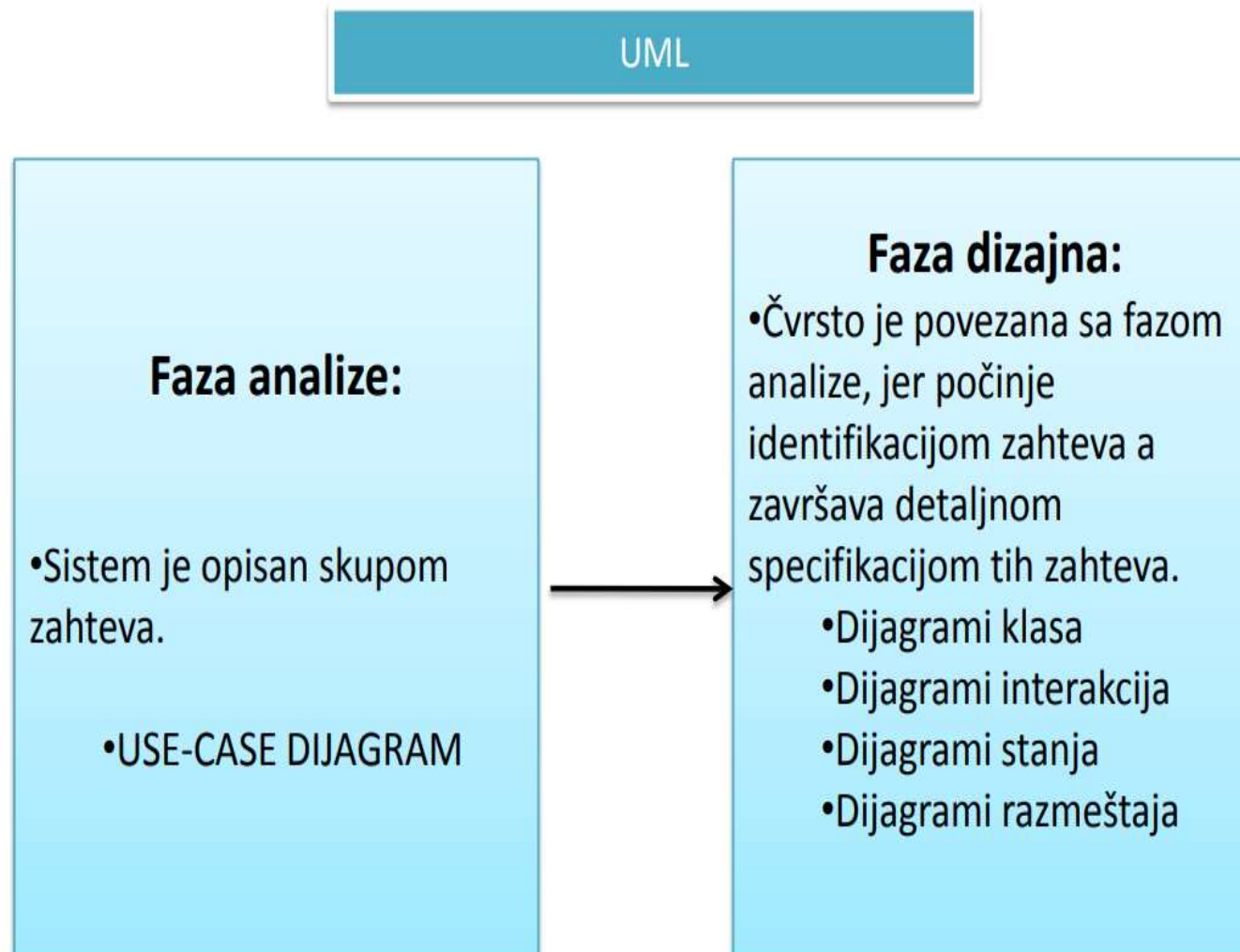
# Kategorije korisnika

- UML koriste sledeće kategorije korisnika
  - ❖ **Sistem analitičari i krajnji korisnici** – specifikacija zahtevane strukture i ponašanje sistema
  - ❖ **Arhitekta sistema** – projektanti sistema koji će zadovoljiti zahteve
  - ❖ **Razvojni inženjeri** (*developers*) – transformišu arhitekturu u izvršni kod
  - ❖ **Kontrolori kvaliteta** – provera strukture i ponašanje sistema
  - ❖ **Rukovodioci projekta** (*managers*) – vode i usmeravaju kadrove i resurse

# Tri osnovna gradivna bloka UML-a

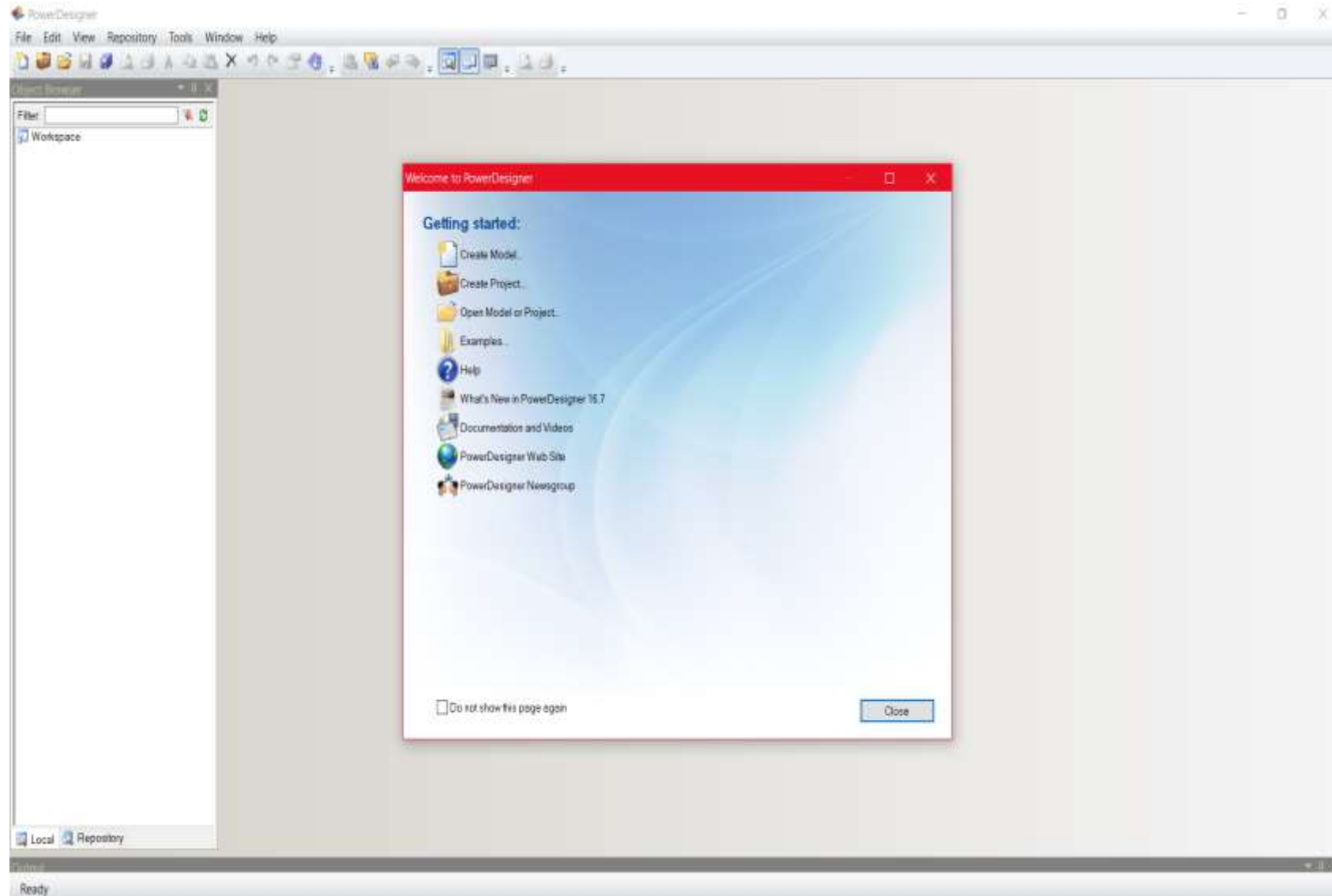
- Stvari – bitni koncepti
  - Relacije – povezivanje individualnih stvari
  - Dijagrami – grupisanje međusobno povezanih kolekcija stvari i relacija
- 
- Kratak osvrt*

# FAZE UML-a: ANALIZA i DIZAJN

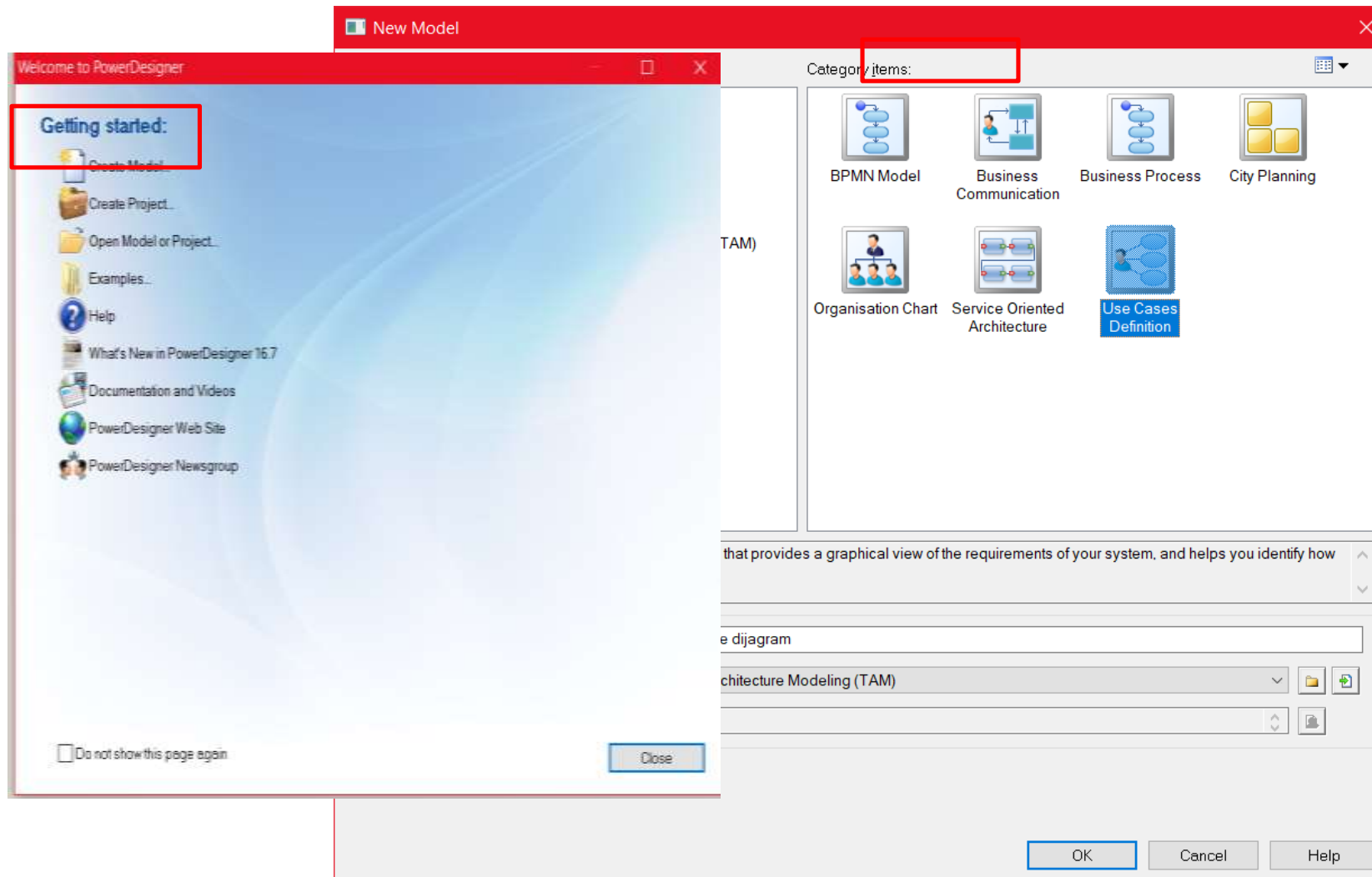


<https://getintopc.com/wait-for-resource-8/>

## Softver Power Designer

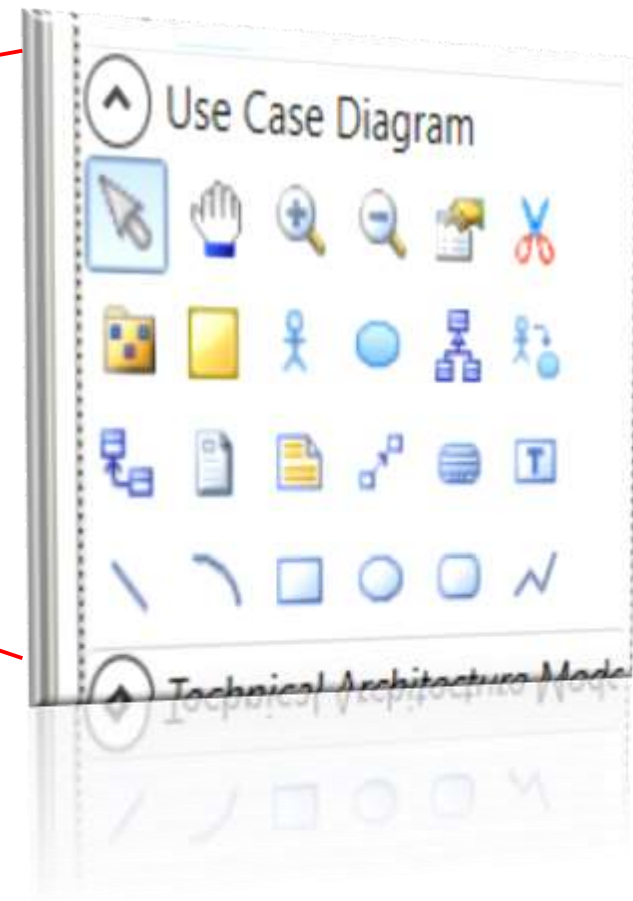
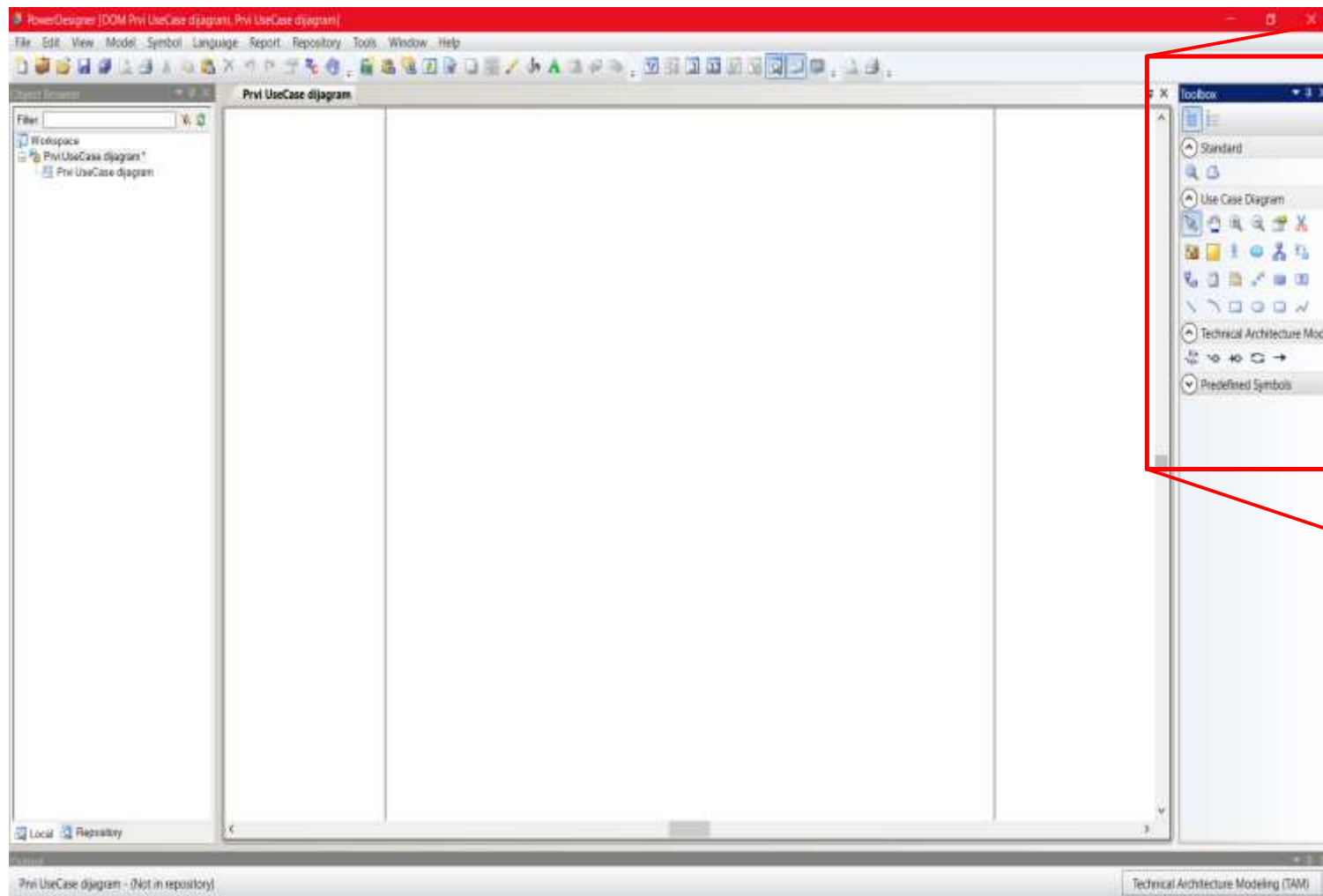


# Kreiranje novog modela

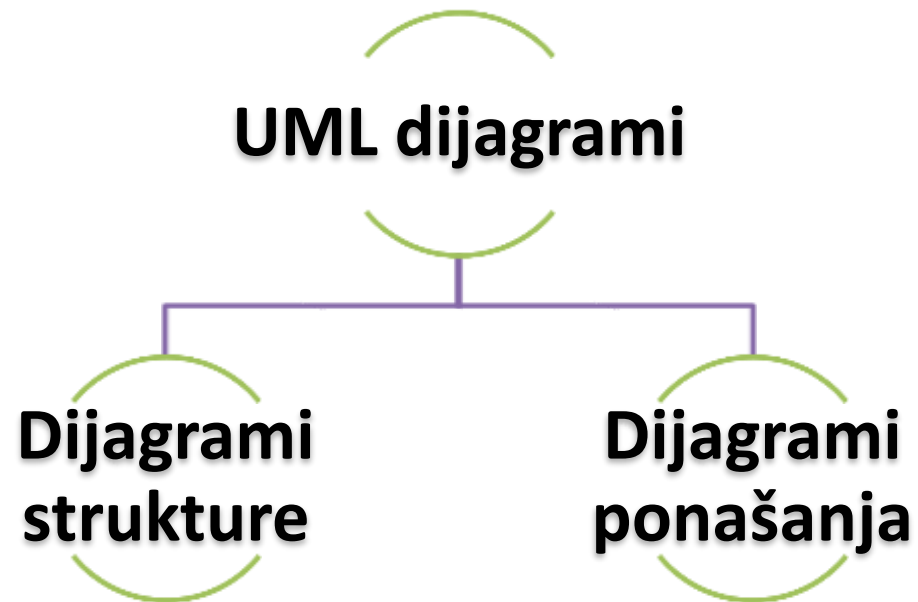


# Kreiranje novog modela

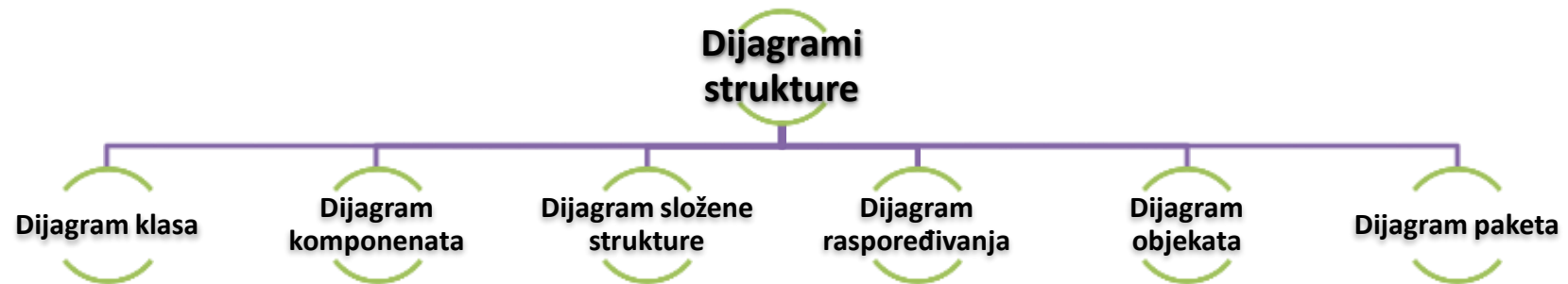
- Dobijamo prazan „papir“ a sa desne strane prozora su elementi koje možemo ubaciti u ovu vrstu dijagrama...



# UML vrste dijagrama

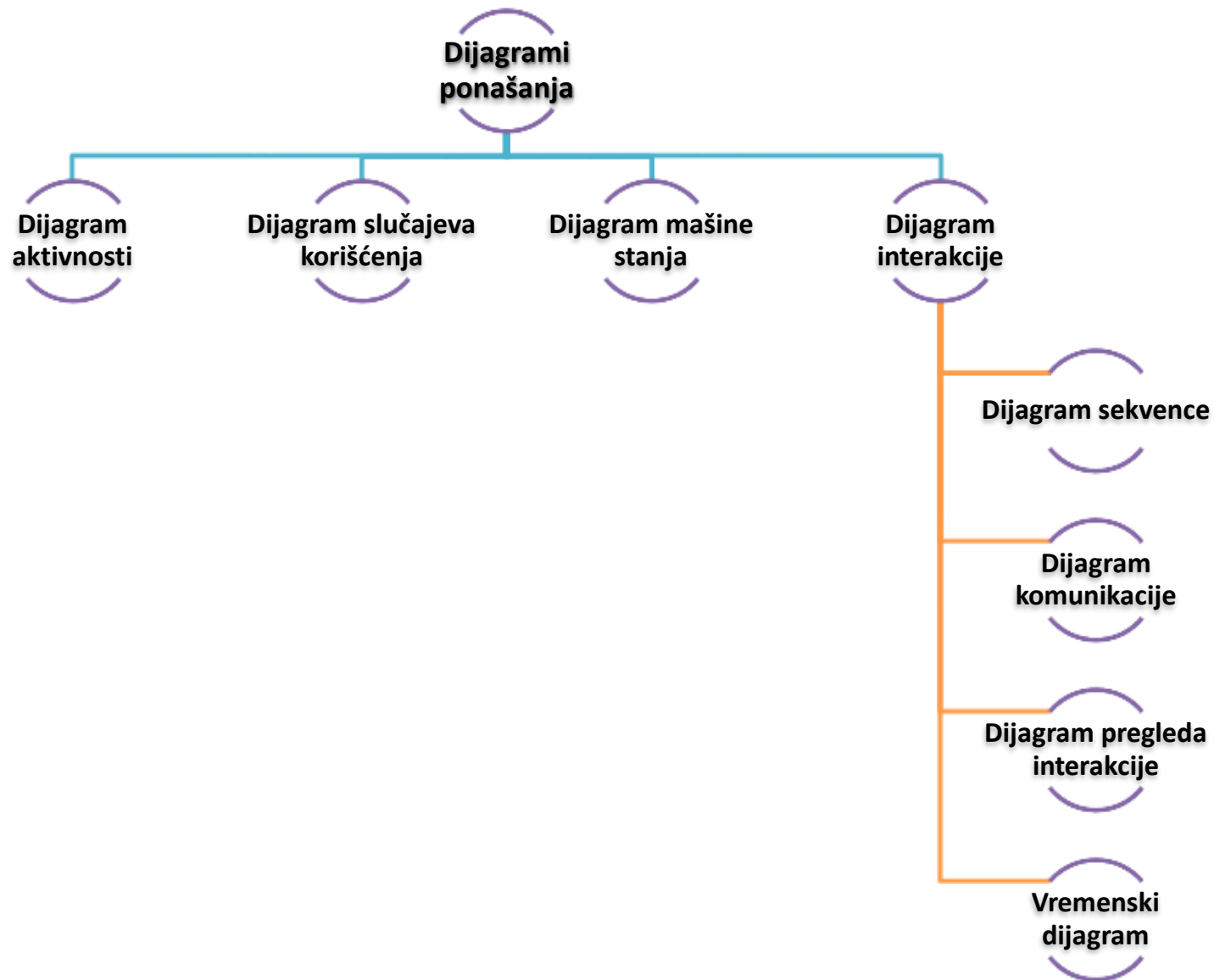


# UML vrste dijagrama





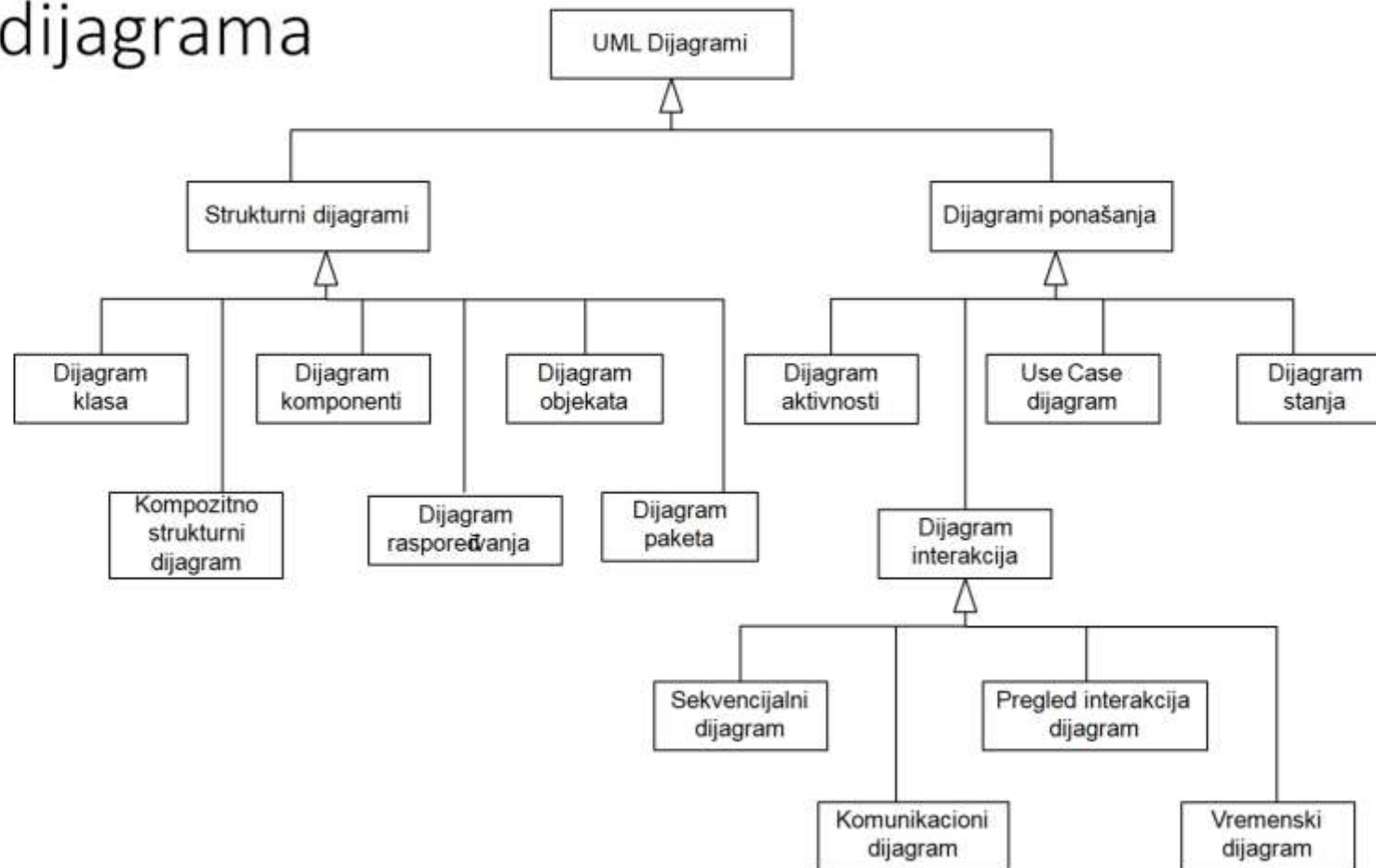
# UML vrste dijagrama



# UML dijagrami

- ❑ Dijagram u UML-u – grafička predstava skupa elemenata - iscrtan kao graf čvorova (elemenata) i veza (relacija)
- ❑ Dijagrami UML-a prikazuju sistem iz više uglova:
  - ❖ **Dijagram slučajeva upotrebe (Use-Case Diagram)**
  - ❖ **Dijagram klasa (Class Diagram)**
  - ❖ **Dijagram objekata (Object Diagram)**
  - ❖ **Dijagram sekvenci (Sequence Diagram)**
  - ❖ **Dijagram saradnje (Collaboration Diagram)**
  - ❖ **Dijagram promene stanja (State Diagram)**
  - ❖ **Dijagram aktivnosti (Activity Diagram)**
  - ❖ **Dijagram komponenti (Component Diagram)**
  - ❖ **Dijagram razvoja (Deployment Diagram)**

# Tipovi dijagrama



# Najčešće korišćeni UML dijagrami

- Dijagrami klasa – klase, odlike, veze
- Dijagrami interakcije –kombinacija dijagrama sekvenci i aktivnosti
- Use Case dijagrami – interakcija korisnika i sistema
- Dijagrami aktivnosti – proceduralno i paralelno ponašanje
- Dijagrami stanja – kako događaji menjaju objekat
- Dijagrami deployment-a (raspoređivanje) – fizička organizacija sistema

# Grativni blokovi UML-a







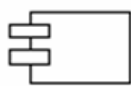

- ❑ Elementi (things)
- ❑ Relacije (relationships)

# Things

Postoje 4 vrste elementi (things):

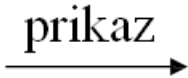
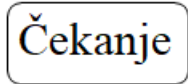
- ❖ **Elementi strukture** – statički delovi modela koji reprezentuju konceptualne ili fizičke elemente (imenice)
- ❖ **Elementi ponašanja** – dinamički delovi modela koji reprezentuju ponašanje kroz prostor i vreme (glagoli)
- ❖ **Elementi grupisanja** - organizacioni delovi modela
- ❖ **Elementi anotacije** – opisni delovi modela, komentari koji se primenjuju na bilo koji dokument

# Statički delovi modela

Ime	Simbol	Opis
Klasa		Opis skupa objekata koji dele iste attribute, operacije, veze i semantiku. Implementira 1 ili više interfejsa.
Interfejs		Kolekcija operacija koje opisuju servise klase ili komponente.
Kolaboracija (Saradnja)		Definiše interakciju i udružuje uloge i druge elemente tako da rade zajedno i obezbeđuju kolaborativno ponašanje.
Korisnik		Spoljašnji entitet koji komunicira sa sistemom, obično osoba.
Slučaj upotrebe		Opis skupa sekvenci akcija koje sistem izvodi da bi izvršio neki zahtev korisnika.
Aktivne klase		Klase čiji objekti poseduju jedan ili više procesa ili niti – mogu inicirati kontrolnu aktivnost.
Komponenta		Fizički i zamenljivi deo sistema koji obezbeđuje realizaciju skupa interfejsa
Čvor		Fizički element koji postoji u vreme izvršavanja i predstavlja računarski resurs – ima memoriju i mogućnost procesiranja.

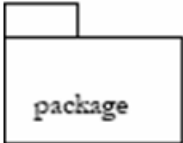
# Dinamički delovi UML modela

...

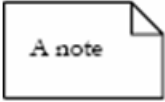
Ime	Simbol	Opis
Interakcija		Ponašanje prilikom razmene skupa poruka između skupa objekata da bi se objasnile specifične namene.
Prikaz stanja		Ponašanje specificirano sekvencom stanja objekta ili neke interakcije.



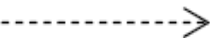

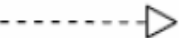
# Organizacioni delovi UML modela

Ime	Simbol	Opis
Paket		<p>Grupe na koje model može biti dekomponovan. Mehanizam opšte namene, za organizovanje elemenata u grupe.</p> <p>Paket je čisto konceptualan – postoji samo u vreme razvoja.</p>



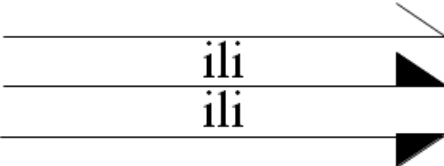



# Delovi za objašnjenja

Ime	Simbol	Opis
Anotacija		<p>Komentari kojima opisujemo, objašnjavamo i naznačavamo bilo koji element u modelu.</p> <p>Osnovna vrsta anotacije je napomena (note).</p>

# Relacije (relationships)

Ime	Simbol	Opis
Zavisnost		Semantička relacija između nezavisne i zavisne stvari. Nezavisna stvar utiče na semantiku zavisne.  Usmerenje – iz zavisnog slučaja.
Asocijacija	$\frac{0..1}{\text{radi}} \quad \frac{*}{\text{radj}}$	Strukturna relacija koja opisuje skup veza kojim se postavlja veza između objekata.
Generalizacija		Objekti specijalizovanih elemenata (dete) predstavljaju zamene za objekte generalizovanih elemenata (roditelj).  Vrh strelice na roditelju.
Realizacija		Semantička relacija između klasifikatora, gde jedan klasifikator specificira ugovor koji drugi klasifikator garantuje da će ispuniti.

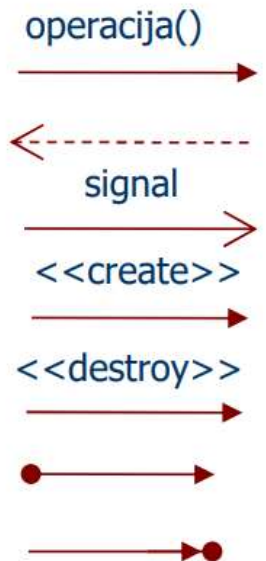
# Vrste akcija na osnovu poslatih poruka

Poruka	Opis akcije
Poziv (call) 	Pokreće operaciju objekta primaoca
Povratak (return) 	Vraća vrednost pozivaocu (opciono)
Slanje operacija (send) 	Asinhrono se šalje signal primaocu
<<create>> 	Kreira se objekat
<<destroy>> 	Uništava se objekat
<<become>> 	Objekat menja prirodu (na obe strane veze je isti objekat)

# Vrste akcija na osnovu poslatih poruka

UML predviđa sledeće vrste poruka:

- poziv (*call*) – pokreće operaciju uloge primaoca
- povratak (*return*) – vraća vrednost pozivaocu
- slanje (*send*) – asinhrono se šalje signal primaocu
- kreiranje (*create*) – kreira se objekat (primerak uloge)
- uništavanje (*destroy*) – uništava se objekat
- pronađena poruka (*found*) – poznat primalac, slanje nije opisano
- izgubljena poruka (*lost*) – poznat pošiljalac, prijem neodređen



# Vrste **akcija** na osnovu poslatih poruka

Pseudostanja - tipovi stanja:

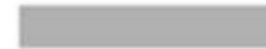
Početno stanje - *Initial*



Krajnje stanje – *Activity final*



Sinhronizacija - (*Fork i Join*)



Stanje odluke - grananje



# Indikatori multiplikativnosti asocijacije

Indikator	Značenje
0..1	Nula ili jedan
1 ili 1..1	Samo jedan
0..* ili *	Nula ili više
1..*	Jedan ili više
n	Samo n (gde je $n > 1$ )
0..n	Od nule do n (gde je $n > 1$ )
1..n	Od jedan do n (gde je $n > 1$ )

# Objektno orijentisana analiza



# Osnovni pojmovi

- Svaki ozbiljniji projekat prolazi kroz faze: analiza, projektovanje, implementacija, testiranje
  - slično je sa SW projektima, kroz faze se prolazi iterativno
- Objektno-orijentisana metodologija razvoja
  - dominantna u proizvodnji softvera danas
- Pojmovi
  - objektno-orijentisana analiza - OOA
  - **objektno-orijentisano projektovanje – OOD**
  - objektno-orijentisano programiranje - OOP
  - objektno-orijentisani jezik - OOL

# Principi OO modela

- Osnovni (obavezni)
  - Apstrakcija
  - Kapsulacija
  - Modularnost
  - Hijerarhija
  - Polimorfizam
- Dodatni (neobavezni)
  - Tipizacija
  - Konkurentnost
  - Perzistencija

# Apstrakcija i kapsulacija

- Apstrakcija ističe esencijalne karakteristrike objekta koje ga razlikuju od drugih vrsta objekata i tako definiše jasne konceptualne granice iz perspektive posmatrača
- Kapsulacija je proces sakrivanja onih elemenata apstrakcije koji definišu strukturu i ponašanje. Kapsulacija služi da razdvoji konceptualni interfesj od implementacije apstrakcije

# Modularnost i hijerarhija

Modularnost je osobina sistema da se razlaže na skup kohezivnih i slabo spregnutih modula

Hijerarhija je rangiranje ili uređivanje apstrakcija

Nasleđivanje - “*is a*” hijerarhija

- jednostruko/višestruko
- potpuno (javno)/strukturno (privatno)
- sadržavanje - hijerarhija
- po vrednosti/po referenci (relevantno u C++, ali ne u Javi)
- agregacija/kompozicija

# Tipizacija i polimorfizam

- Tipizacija je osobina da se objekti različitih klasa ne mogu uopšte ili se mogu zamenjivati na ograničene načine
  - stroga i slaba tipizacija
  - statička i dinamička tipizacija (vezivanje)
- Dinamička tipizacija i dinamičko vezivanje
  - tehnički preduslov za ispoljavanje polomorfizma
- Polimorfizam je osobina da se objekat kojem se pristupa kao objektu osnovne klase ponaša različito:
  - kao objekat osnovne klase ili kao objekat izvedene klase
  - ponašanje zavisi od dinamičkog tipa objekta, ne statičkog tipa reference
- Polimorfizam objekta se zasniva na virtuelnim metodama

# Konkurentnost i perzistencija

- Principi koji se dobro uklapaju u OO paradigmu
- Nisu suštinski principi koji određuju da li je softver OO
  - OO softver ih ne mora posedovati
  - Softver koji nije OO ih može posedovati
- Konkurentnost je osobina koja razlikuje aktivne objekte od pasivnih
  - *proces* - ima vlastiti adresni prostor (tipično njime upravlja OS)
  - *nit* - deli isti adresni prostor sa drugim nitima
- Perzistencija je osobina po kojoj se postojanje objekta proteže
  - kroz *vreme* (obj. nastavlja da živi nakon nestanka njegovog stvaraoca)
  - kroz *prostor* (obj. se premešta iz adresnog prostora u kojem je stvoren)

# Model i modeliranje

- Model je pojednostavljenje realnosti
- Model nekog sistema je apstrakcija tog realnog sistema iz određenog ugla posmatranja
- Osnovna namena modela
  - da se sistem koji se razvija bolje razume
- Modeliranje je važnije što je sistem kompleksniji
  - kompleksnost je odlika današnjih SW sistema
- Savremena metodologija razvoja softvera
  - *Model Driven Development* (MDD)

# Ciljevi modeliranja

- Model pomaže da se sistem vizuelizuje
- Model omogućava da se specificira
  - struktura sistema
  - ponašanje sistema
- Model daje šablon koji usmerava konstrukciju sistema
- Model dokumentuje projektne odluke koje se donose
- Model smanjuje cenu razvoja
  - omogućava ispitivanje projektnih odluka po nižoj ceni



# Logički i fizički aspekti modela

- Logički model sistema
  - opisuje ključne apstrakcije i mehanizme koji
    - obrazuju prostor problema ili
    - definišu arhitekturu sistema
  - definiše
    - strukturu i relacije između klasa
    - relacije i interakcije između objekata
- Fizički model sistema
  - opisuje konkretnu softversku i hardversku kompoziciju
  - definiše arhitekturu modula i arhitekturu procesa

# Statički i dinamički aspekti modela

- Statički aspekti modela se fokusiraju na
  - strukturu sistema
- Dinamički aspekti modela se fokusiraju na
  - ponašanje sistema
- Realni sistemi uvek imaju dinamičko ponašanje:
  - objekti se kreiraju i uništavaju
  - objekti šalju poruke drugim objektima nekim redosledom
  - spoljašnji događaji izazivaju reakcije izvesnih objekata

# Alati za modeliranje

- IBM Rational: Software Architect  
(Rose, Rose XDE Developer, Software Modeler)
  - <http://www-01.ibm.com/software/rational/products/swarchitect/>
- Borland: Together
  - <http://www.borland.com/products/Together/default.aspx>
- Gentleware: Poseidon for UML
  - <http://www.gentleware.com>
- Open Source: StarUML
  - <http://staruml.sourceforge.net/en/>
- Altova: Umodel
  - [http://www.altova.com/download/umodel/uml\\_tool.html](http://www.altova.com/download/umodel/uml_tool.html)
- Omondo: EclipseUML
  - <http://www.omondo.com>
- Sparx Systems: Enterprise Architect
  - <http://www.sparxsystems.com>
- Visual Paradigm: Visual Paradigm for UML
  - <http://www.visual-paradigm.com/product/vpuml>
- Embarcadero Technologies: ER/Studio Software Architect
  - <http://www.embarcadero.com/products/er-studio-software-architect>
- Pregled alata:
  - [http://en.wikipedia.org/wiki/List\\_of\\_Unified\\_Modeling\\_Language\\_tools](http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools)

# Case Enterprise Architect

# Model podataka i modeliranje

- Model je pojednostavljenje realnosti. To je apstrakcija realnog ili planiranog sistema iz određenog ugla posmatranja.
- Model podataka organizuje poznate činjenice i standardizuje kako se one odnose jedna prema drugoj.
- Modeliranje je važnije što je sistem kompleksniji.

# UML

Elementi UML-a su:

- Osnovni gradivni blokovi
- Pravila za povezivanje gradivnih blokova
- Opšti mehanizmi koji se primenjuju u UML-u

Gradivni blokovi UML-a

- Stvari (*things*): klasa, interfejs, komponenta, paket, napomena...
- Relacije (*relationships*): generalizacija, realizacija, asocijacija, zavisnost
- Dijagrami (*dijagrams*) grupišu interesantne skupove povezanih stvari: klasni dijagram, dijagram sekvence, aktivnosti, slučajeva korišćenja...

# UML dijagrami

Na osnovu uglova posmatranja, UML definiše tri grupe diagrama, sa ukupno trinaest tipova:

- Strukturalne: *Class, Object, Component, Composite Structure*, i *Deployment* diagram.
- Ponašanja: *Use Case, Activity Diagram*, i *State Machine* diagram.
- Interakcije: *Sequence, Communication, Timing*, i *Interaction Overview* diagram.

# Case alati za modelovanje UML-a

- Enterprise Architect - *Sparx Systems*
- PowerDesigner - *Sybase*
- Eclipse (UML2 Tools) - *Eclipse Foundation*
- UModel - *Altova*
- Visio - *Microsoft*
- ...



# Enterprise Architect

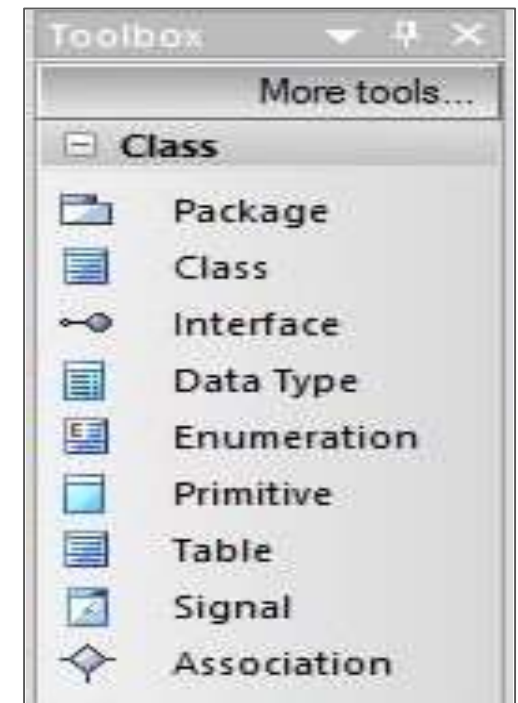
- File > New Project
- *Zadati lokaciju i ime projektu*
- Čarobnjak:
  - Tab Model Patterns:
    - Technology: Basic UML 2 Technology
    - Name: **Class** (*klasni dijagram*)
- Kliknuti: OK

Ovim postupkom kreirali smo novi projekat, sa jednim klasnim dijagramom.

# Enterprise Architect - alati

## Toolbox:

- Class, dodavanje:
  - Package - organizacioni paket elemenata (namespace)
  - Class - klasa (Automobil)
  - Interface - interfejs (InstrumentTabla)
  - Data Type - složeni tip, struktura (DateTime)
  - Enumeration - enumeracija (RED, BLUE, YELLOW)
  - Primitive - osnovni tip (Integer, String)
  - Table - tabela za baze podataka
  - Signal - asihrona akcija (send email, place order)
  - Association - višestruka asocijacija

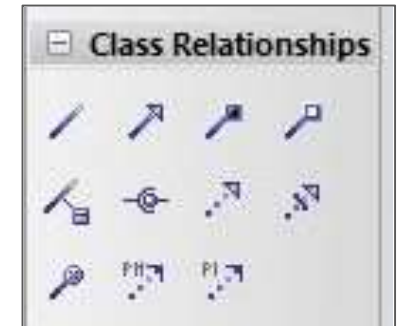


# Enterprise Architect - alati

Toolbox:

- Class Relationships, relacije:

- Associate - *asocijacija*, odnos dve klase (Automobil - Vlasnik)
- Generalize - nasleđivanje (Automobil - Jugo)
- Compose - *kompozicija*, sastavljen od zavisnih elemenata (Automobil-Motor; Knjiga-Stranica)
- Aggregate - *agregacija*, sadrži zavisne elemente (Automobil - Točak)
- Association Class - dodatak vezi (Automobil - Putovanje + Potrošnja tokom puta)
- Assembly - povezivanje komponenti
- Realize - implementacija interfejsa (Automobil - Instrument tabla, Alarm, GPS Navigacija)
- Template Binding - generička klasa (class Distributer<X> where X : ZastavaKG)
- Nesting - unutrašnja klasa (Automobil - DeloviAutomobila)
- Package Merge - spajanje paketa
- Package Import - uvoz paketa

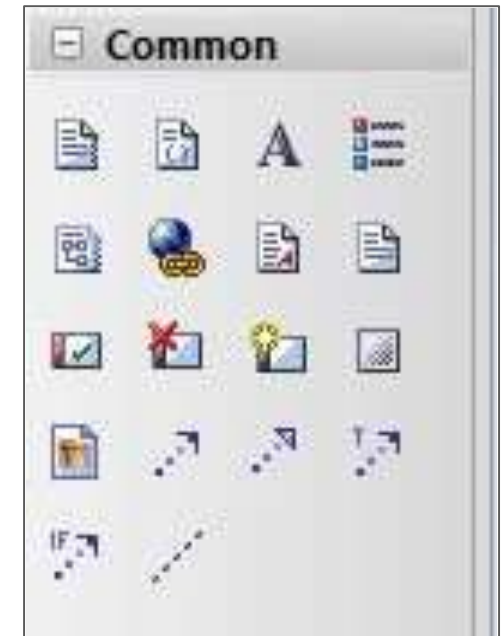


# Enterprise Architect - alati

## Toolbox:

- Common, opšte:

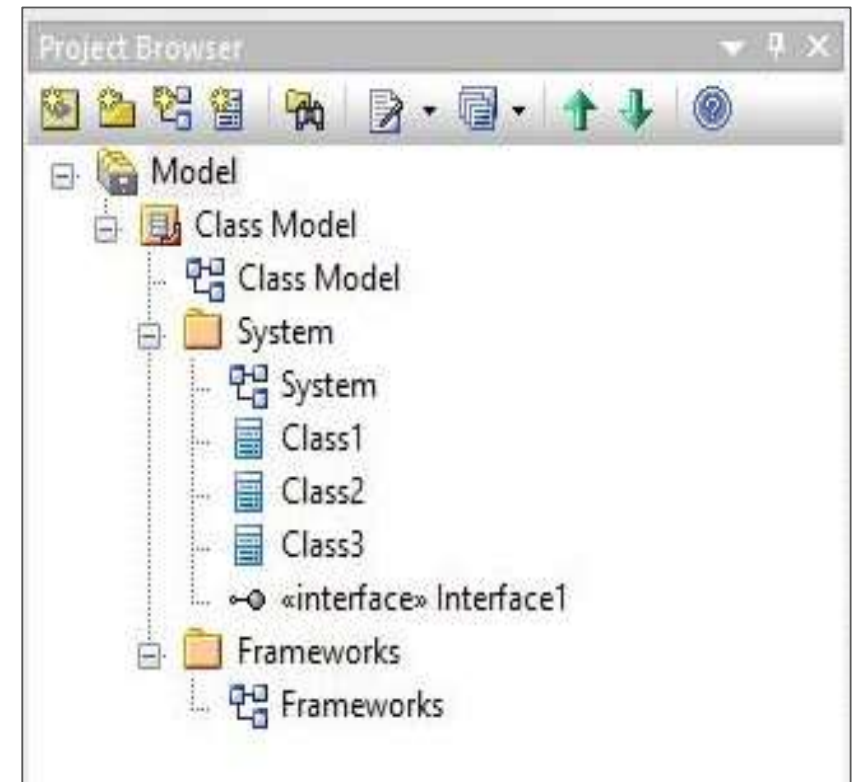
- Note
- Constraint
- Text Element
- Diagram
- Legend
- Diagram Notes
- Hyperlink
- Document
- Document
- Artifact
- Requirement
- Issue
- Change
- Boundary
- Image



# Enterprise Architect - Organizacija projekta

## Project Browser

- Model - sistem koji se razvija
  - Class Model - dijagram
    - delovi klasnog dijagrama
      - paketi
      - sheme
      - klase
      - interfejsi
      - ...
- Podešavanje jezika:
  - Project / Settings / Project Options
    - > Source Code Engineering / Default Language for Code Generation: C#



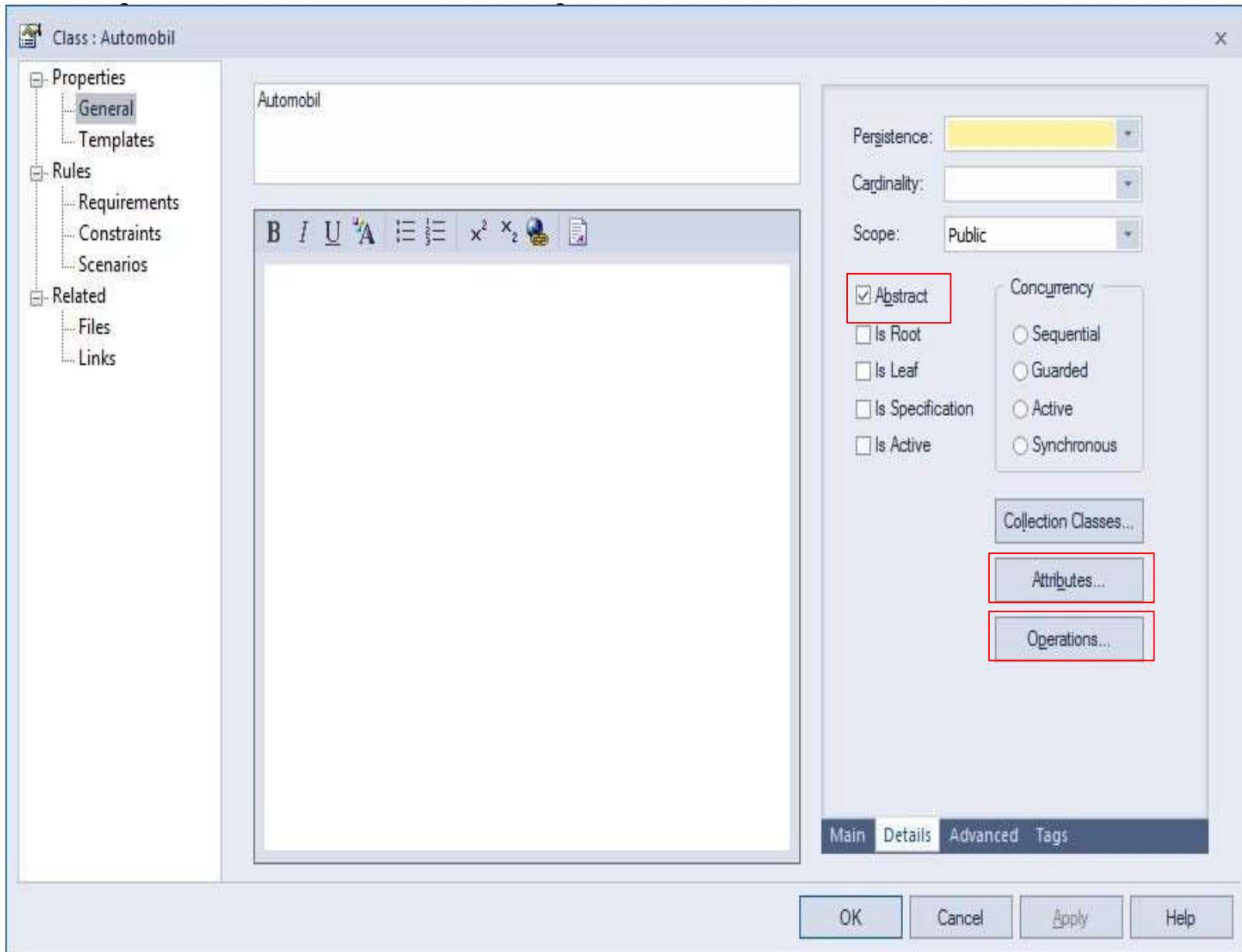
# Dijagram klasa - klasa



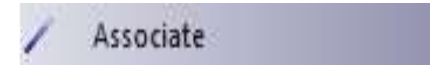
- Klasa može biti apstraktna, sa privatnim/javnim/zaštićenim poljima i sa privatnim/javnim/zaštićenim metodama.



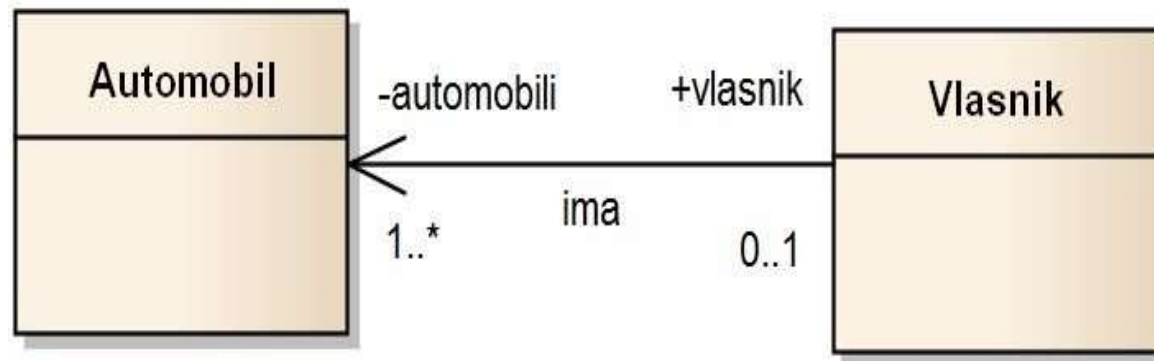
# Dijagram klasa - klasa



# Dijagram klasa - asocijacija



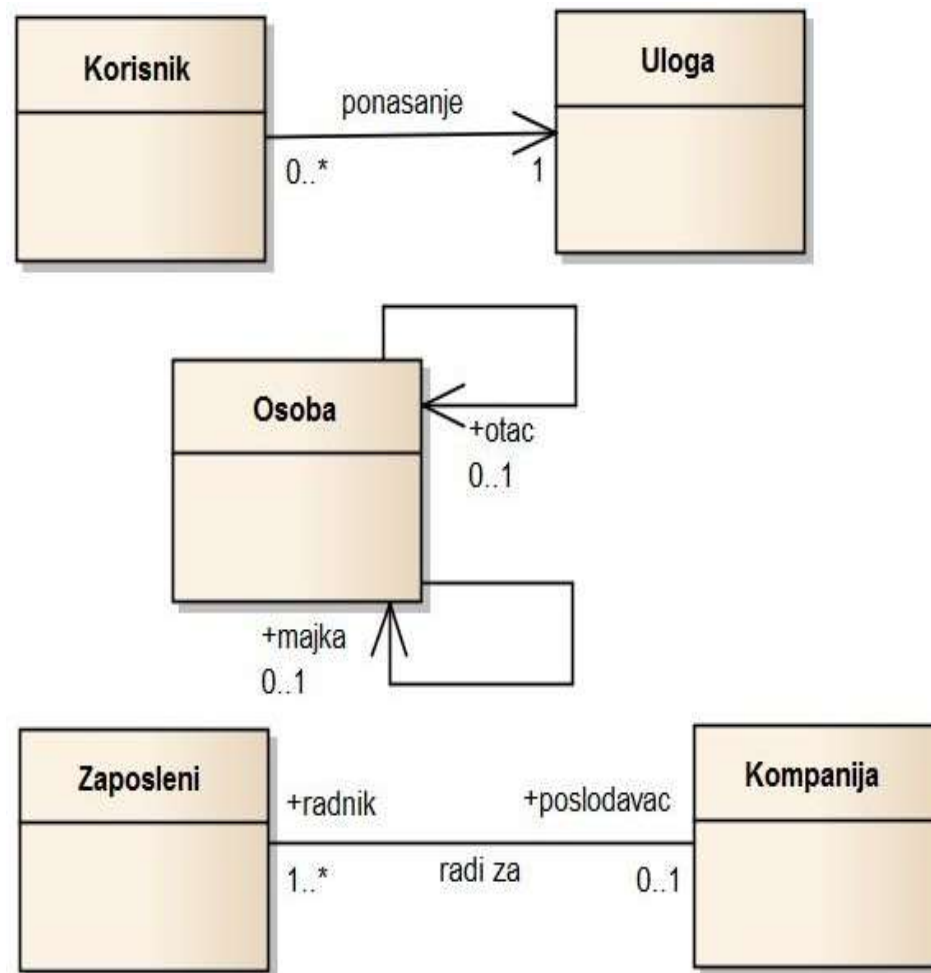
- Klase Automobil i Vlasnik su u asocijaciji, **povezane su**.
- Vlasnik ima jedan ili više automobila. On zna gde su oni, jer postoji navigacija ka automobilima, ali to je privatna informacija.
- Automobil može da ima jednog vlasnika, jer u suprotnom još nije prodat. On nema informaciju o tome koje njegov vlasnik, jer nema navigaciju ka njemu.






# Dijagram klasa - asocijacija (dodatak)

Associate



# Dijagram klasa - asociacija

 Associate

Association Properties

General  
Role(s)  
Constraints  
Binding

SOURCE Automobil

automobili

Multiplicity

Multiplicity	1..*
Ordered	False
Allow Duplic...	False

Detail

Stereotype	
Alias	
Access	Private
Navigability	Navigable
Aggregation	none
Scope	instance
Constraints	
Qualifiers	

Advanced

Multiplicity

1..\*

TARGET Vlasnik

vlasnik

Multiplicity

Multiplicity	0..1
Ordered	False
Allow Duplic...	False

Detail

Stereotype	
Alias	
Access	Public
Navigability	Non-Navigable
Aggregation	none
Scope	instance
Constraints	
Qualifiers	

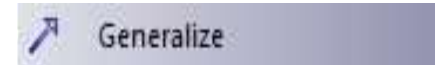
Advanced

Multiplicity

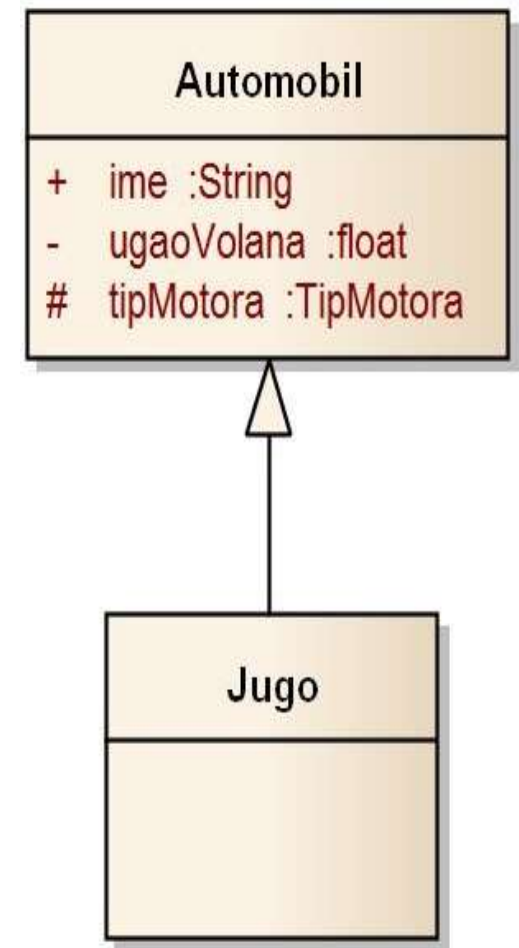
0..1

OK Cancel Help

# Dijagram klasa - nasleđivanje

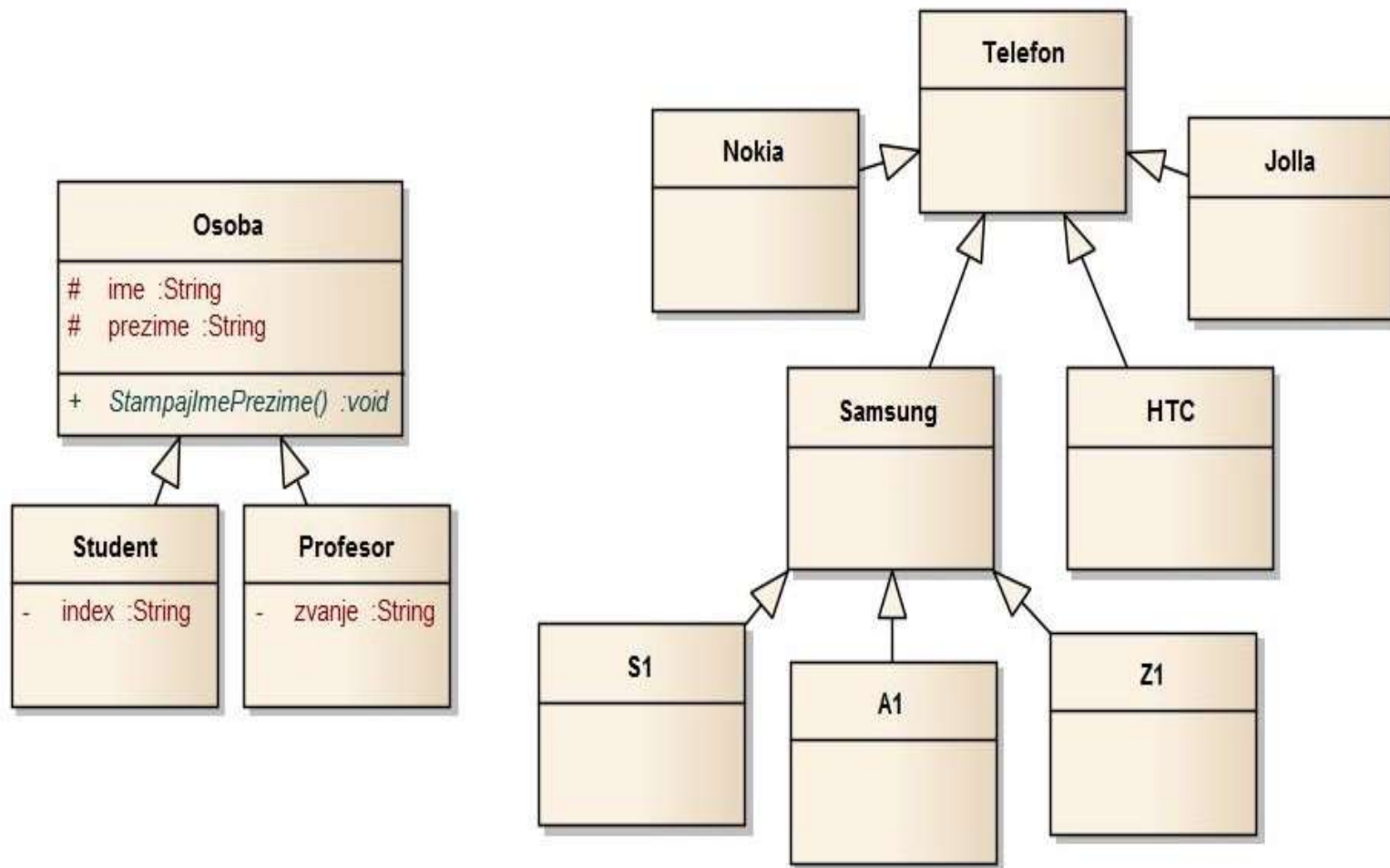


- Klasa Jugo **nasleđuje** klasu Automobil.
- Automobil je *roditeljska* klasa. Ne zna ko su joj nasednici, pa ne može ni da ima njihova ponašanja. Jedna klasa može imati proizvoljan broj naslednika. Posедуje javna, privatna i zaštićena polja.
- Jugo je klasa *naslednik*. Zna ko joj je roditelj, pa tako može i da se ponaša kao roditelj (*casting*). Jedna klasa može imati samo jednog roditelja. Ima pristup javnim i zaštićenim poljima roditeljske klase.
  - public - vidljivo svima
  - protected - vidljivo samo naslednicima
  - private - vidljivo samo unutar klase

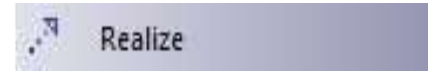


# Dijagram klasa - nasleđivanje (dodatak)

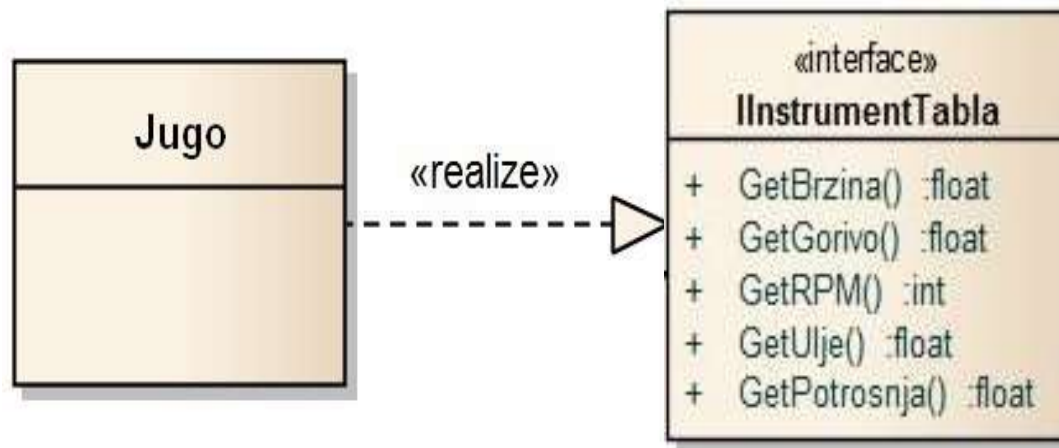
Generalize



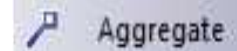
# Dijagram klasa - implementacija



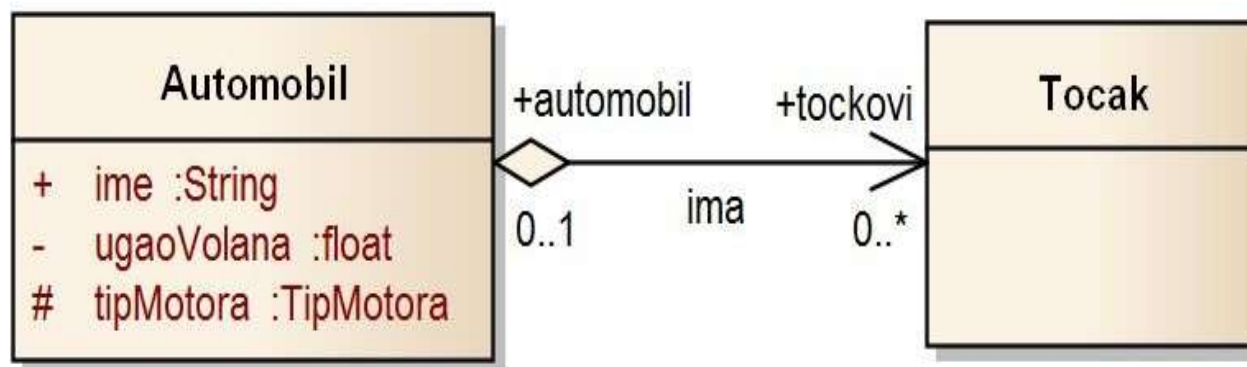
- Klasa Jugo **implementira** interfejs IInstrumentTabla. Interfejs je kolekcija operacija definisana tako da idealistički opisuje određeno ponašanje. Ponašanje **ostvaruje** klasa. Svaka klasa može da implementira proizvoljan broj interfejsa. U C# notaciji imena interfejsa počinje slovom 'I'.
- Interfejs može biti i deklarativan, bez operacija (IAutomobilOtporanNaMetke).



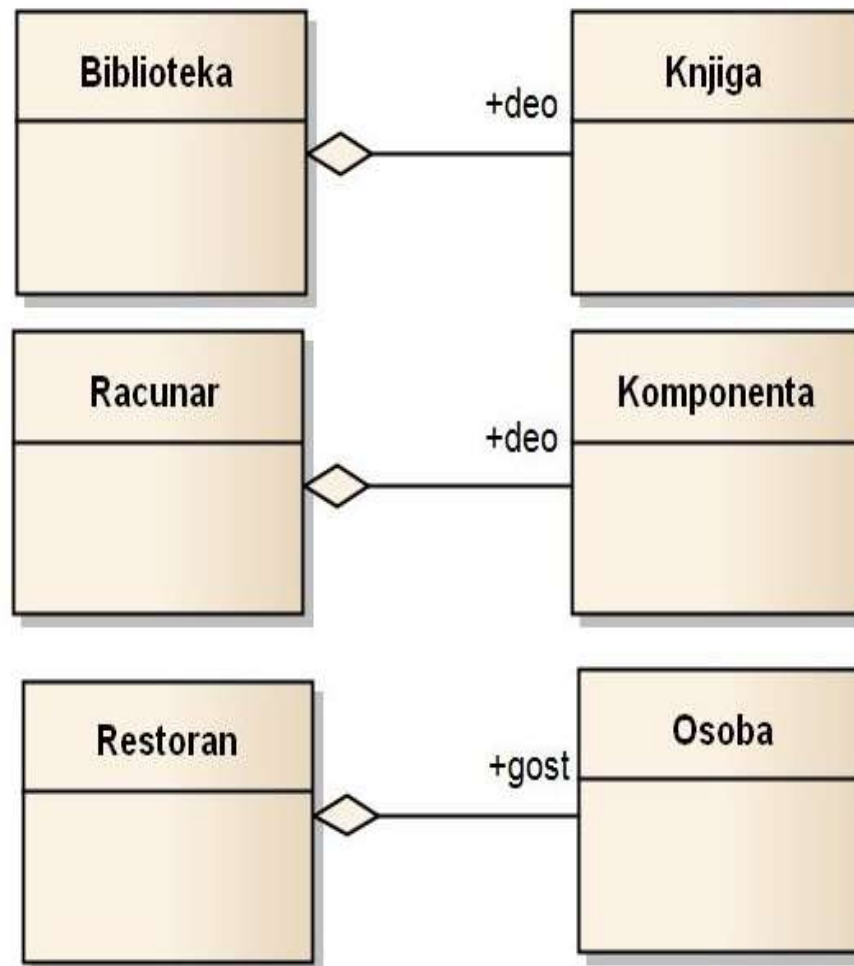
# Dijagram klasa - agregacija



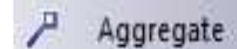
- Klasa Automobil agregira klasu Točak, **sadrži je**.
- Automobil **se sastoji** od točkova, odnosno, točak **je deo** automobila.
- Agregacija je specijalni slučaj asocijacije, odnos deo-celina.
- Romb se nalazi pored kontejnerske klase.
- Klase mogu postojati kao nezavisni elementi.



# Dijagram klasa - agregacija (dodatak)



# Dijagram klasa - agregacija



Aggregation Properties

General  
Role(s)  
Constraints

SOURCE Tocak

tockovi

Multiplicity

Multiplicity	0..*
Ordered	False
Allow Duplic...	False

Detail

Stereotype	
Alias	
Access	Public
Navigability	Navigable
Aggregation	none
Scope	instance
Constraints	
Qualifiers	

Advanced

Multiplicity

0..\*

TARGET Automobil

automobil

Multiplicity

Multiplicity	0..1
Ordered	False
Allow Duplic...	False

Detail

Stereotype	
Alias	
Access	Public
Navigability	Non-Navigable
Aggregation	shared
Scope	instance
Constraints	
Qualifiers	

Advanced

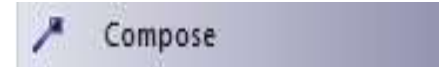
Multiplicity

0..1

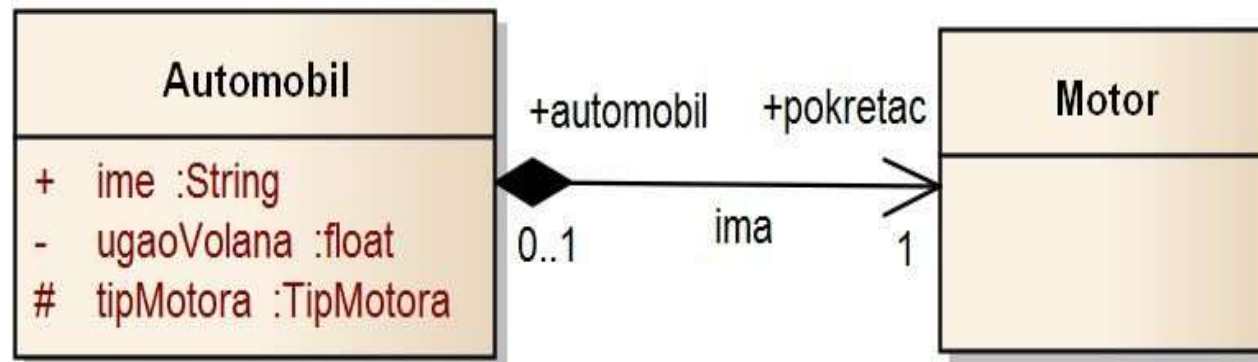
OK Cancel Help



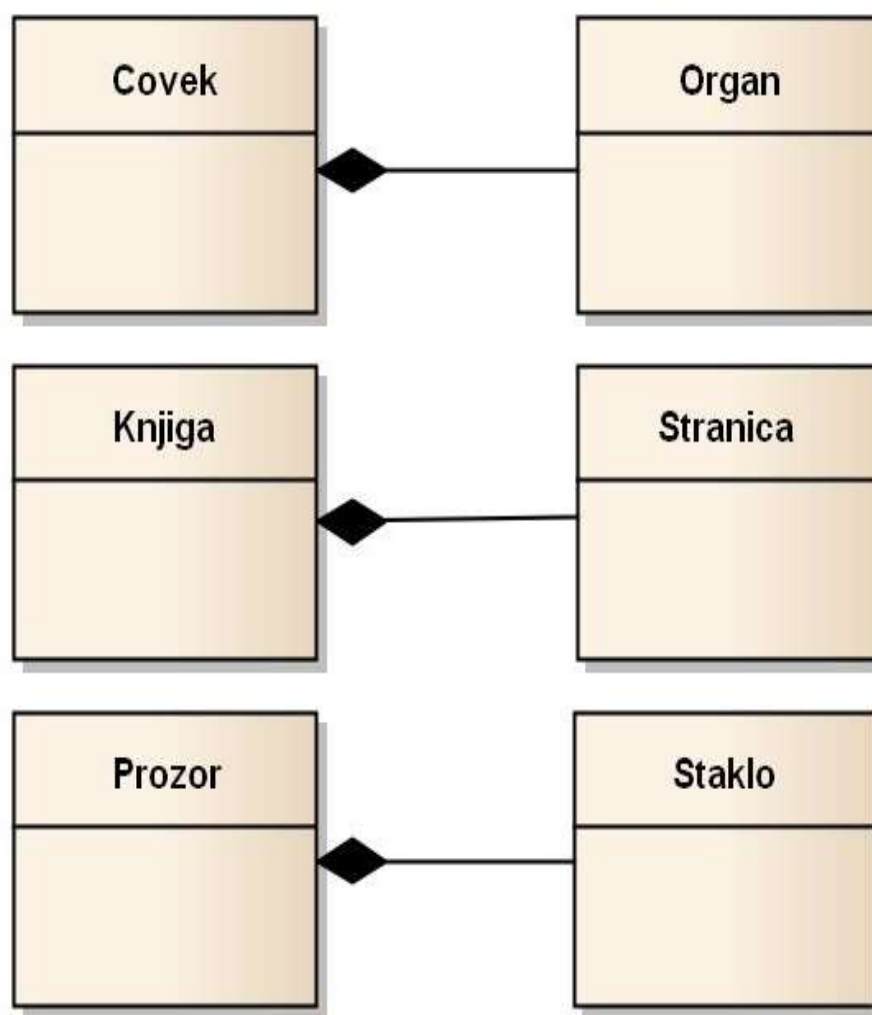
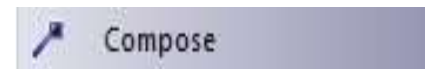
# Dijagram klasa - kompozicija




- Klasa Automobil je u kompoziciji sa klasom Motor
- Osnovni deo Automobila je motor, automobil bez motora nije automobil.
- Kompozicija je specijalni slučaj agregacije, deo i celina su čvrsto povezani.



# Dijagram klasa - kompozicija (dodatak)



# Dijagram klasa - kompozicija (parametri)

 Compose

Aggregation Properties

General  
Role(s)  
Constraints

SOURCE Motor

pokretac

Multiplicity

Multiplicity	1
Ordered	False
Allow Duplic...	False

Detail

Stereotype	
Alias	
Access	Public
Navigability	Navigable
Aggregation	none
Scope	instance
Constraints	
Qualifiers	

Advanced

Multiplicity

1

TARGET Automobil

automobil

Multiplicity

Multiplicity	0..1
Ordered	False
Allow Duplic...	False

Detail

Stereotype	
Alias	
Access	Public
Navigability	Non-Navigable
Aggregation	composite
Scope	instance
Constraints	
Qualifiers	

Advanced

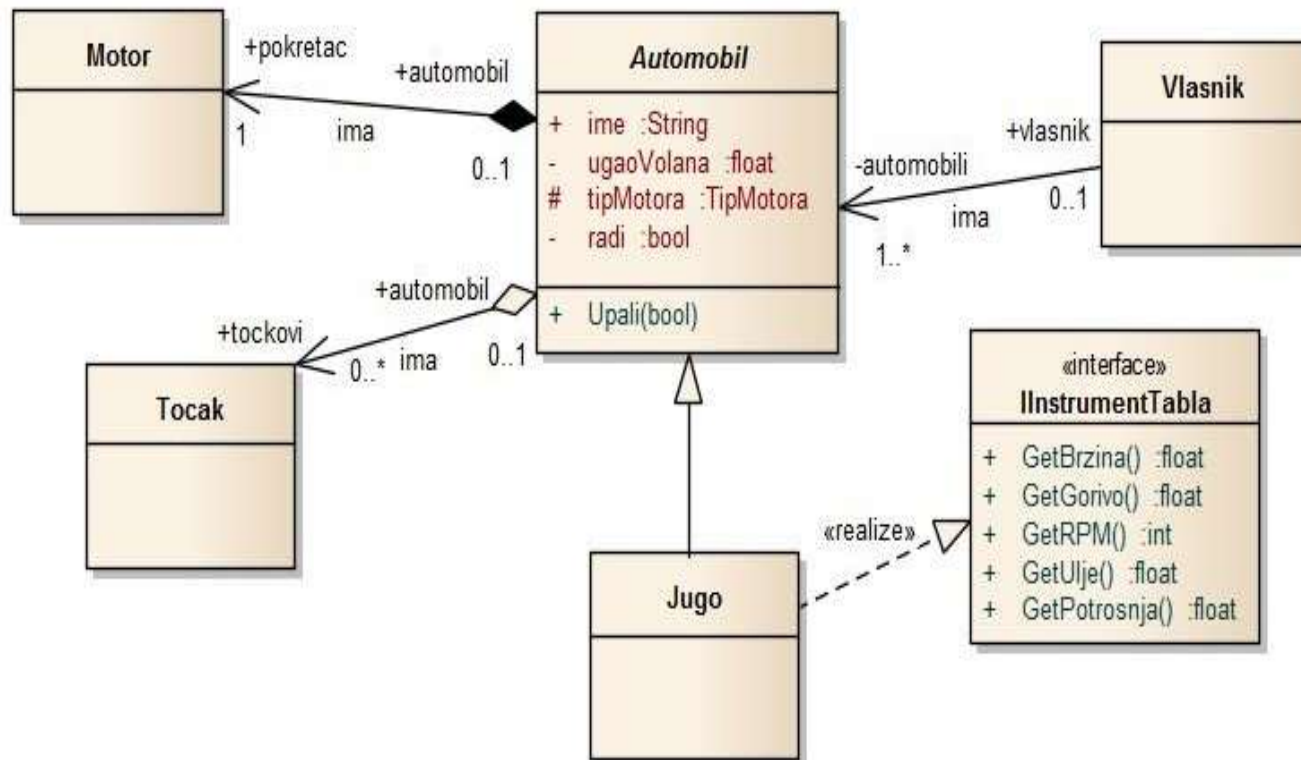
Multiplicity

0..1

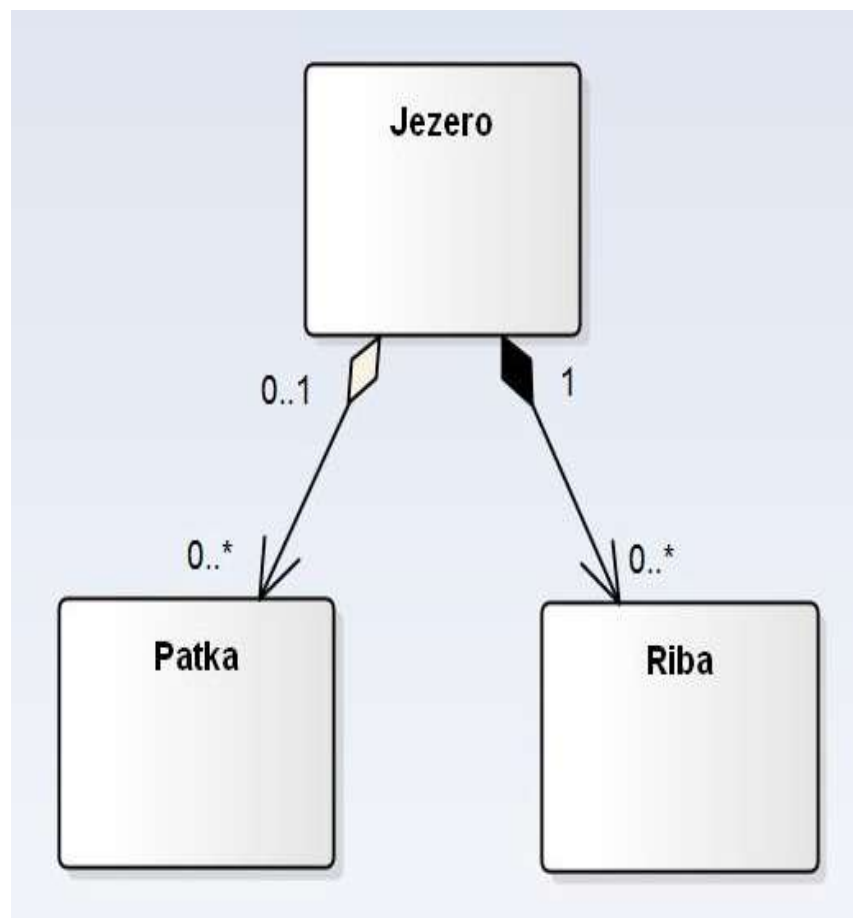
OK Cancel Help

# Dijagram klasa

- Domaći zadatak: proširiti dati dijagram sa dodatnim poljima, odnosima, klasama i interfejsima koji bi učinili specifikaciju potpunijom.



# Agregacija ili kompozicija?



# UML i generisanje koda

# UML dijagram klasa fakulteta

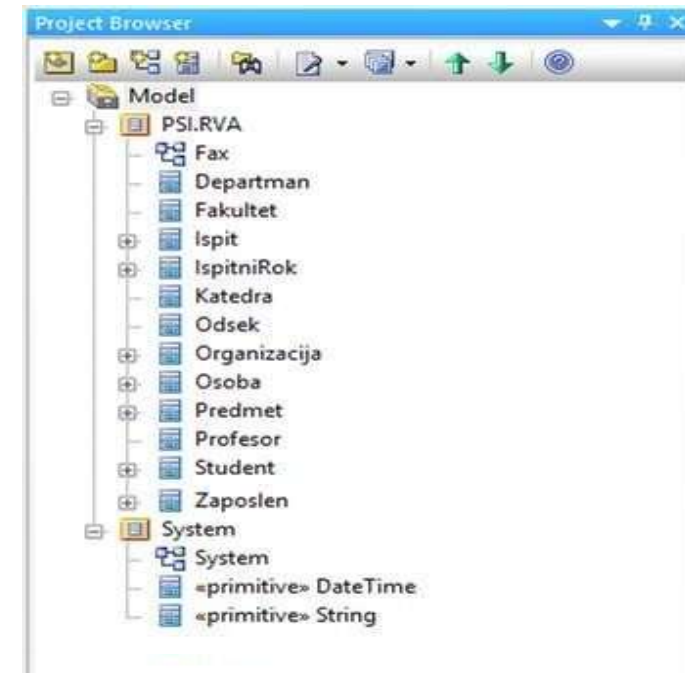
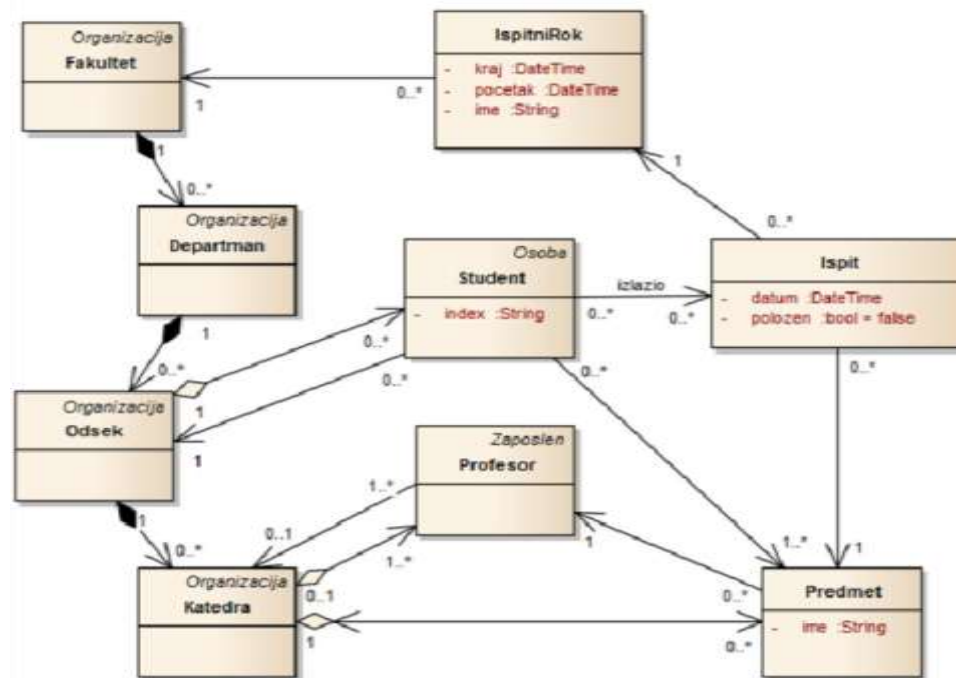
Kreirati novi prazan projekat  
“PSI.RVA.Fakultet”.

U model dodati dva paketa: PSI.RVA i System.

U pakete dodati Fax i System klas dijagame, respektivno.

Potom dodati model po uzoru na sliku...

\*String i DateTime dodati u paket System kao klase sa naznakom (stereotype: primitive)



# UML generisanje koda

UML se može iskoristiti za generisanje koda (konstruisanje)

Generiše se izvorni kod koji reflektuje UML klas dijagram.

Arhitekta projektuje visoko apstraktni model, koji se koristi za generisanje osnovnog koda, tzv. kostura, koji se koristi za dalji razvoj aplikacije.

Moguće je i obrnuto: UML se formira tako što reflektuje izvorni kod.

Na osnovu izvornog koda, i odnosa klasa, moguće je formirati UML model.



# UML generisanje koda (priprema projekta)

Otvoriti Visual Studio i kreirati novi projekat:

File > New Project

odabrati: Console Application

ime projekta: Fax

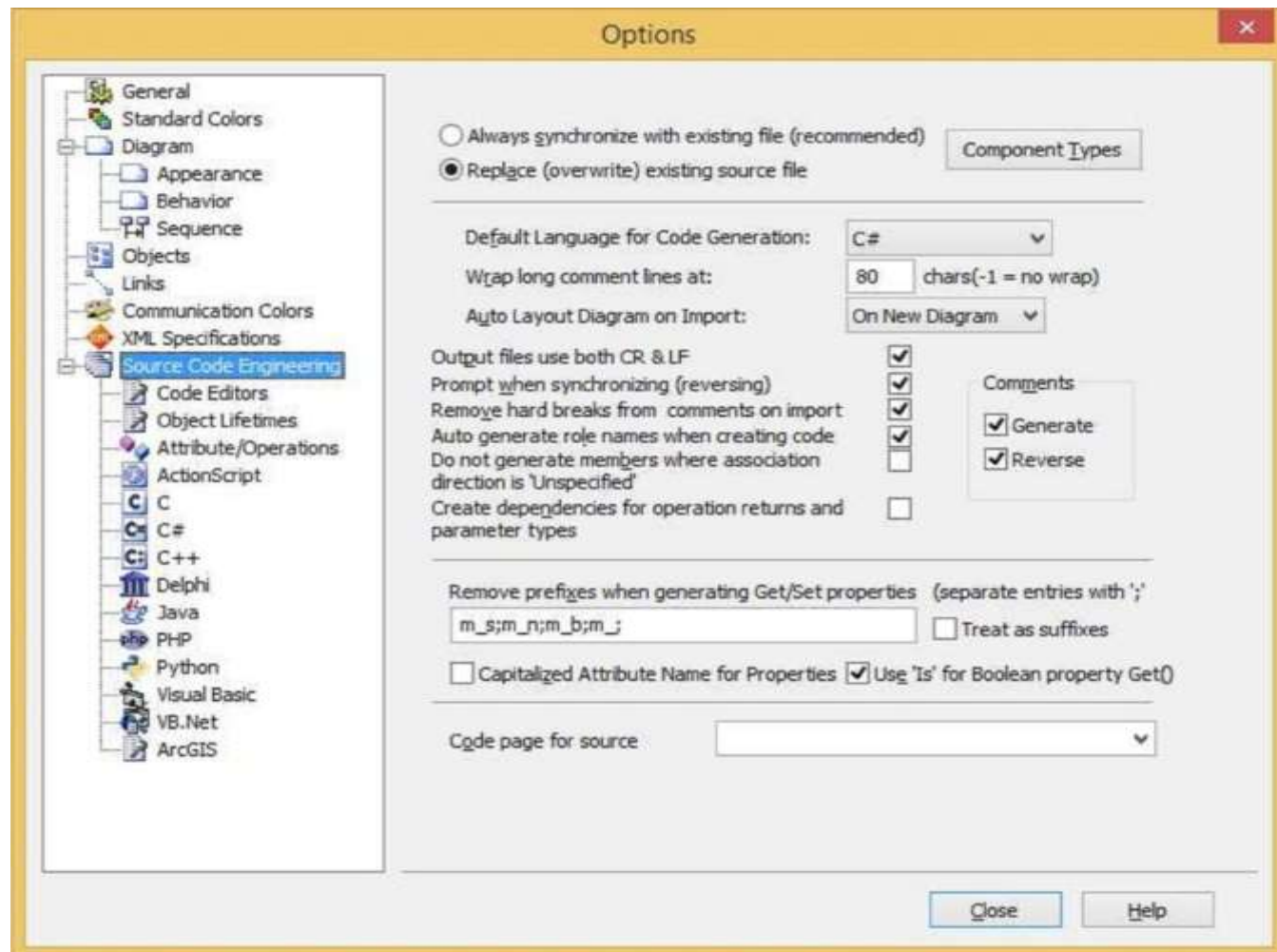
zadati lokaciju

klik na OK

# UML generisanje koda

Unutar Enterprise Architect alata,  
otvoriti podešavanja:

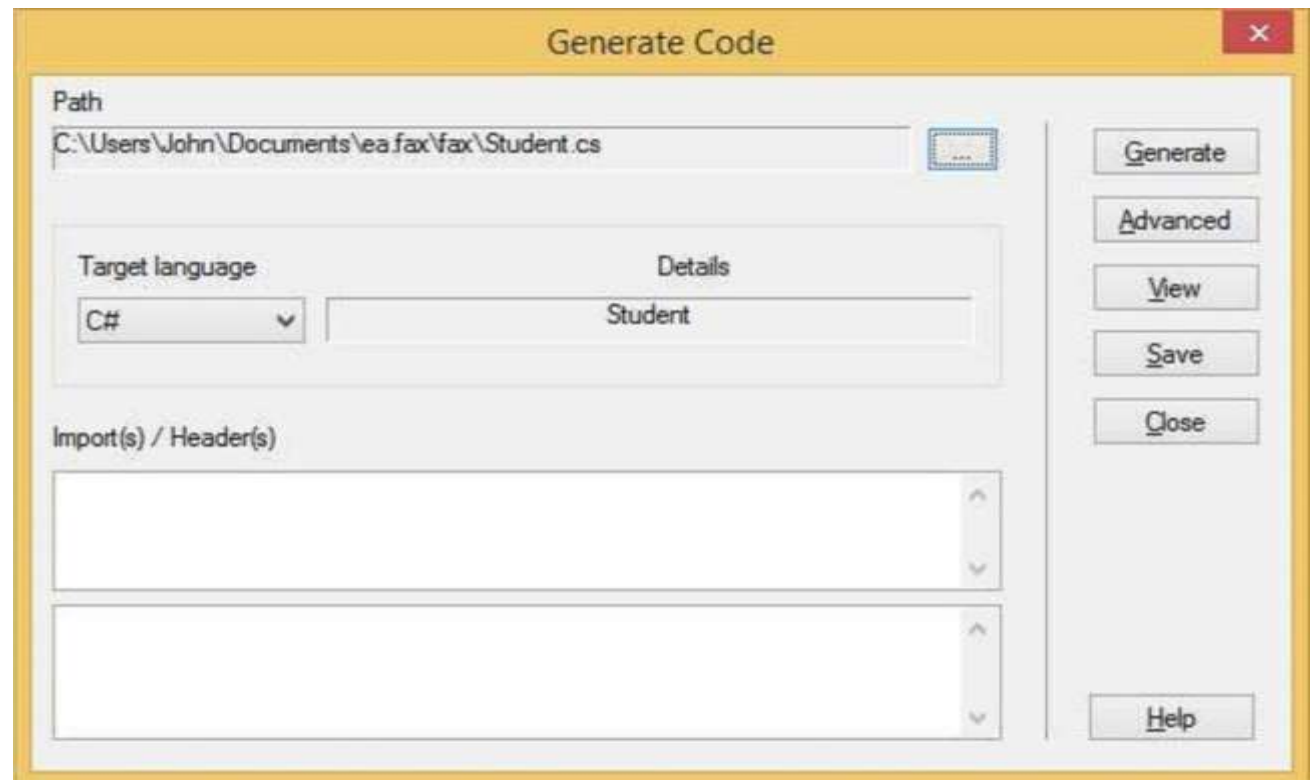
Tools > Options



# UML generisanje koda

Desni klik na željenu klasu  
> Generate Code...

zadati putanju do prethodno  
kreiranog projekta.



# UML generisanje koda - VS

Uključiti izgenerisane fajlove u VS projekat.

Desni klik na projekat: Add > Existing Item...

Proveriti validnost generisanih klasa.

Dobijeni kod...

```
using System;
using PSI.RVA;
namespace PSI.RVA {
    public class Student : Osoba {

        private System.String index;
        public PSI.RVA.Predmet Predmet;
        public PSI.RVA.Ispit Ispit;
        public PSI.RVA.Odsek Odsek;

        public Student(){

        }

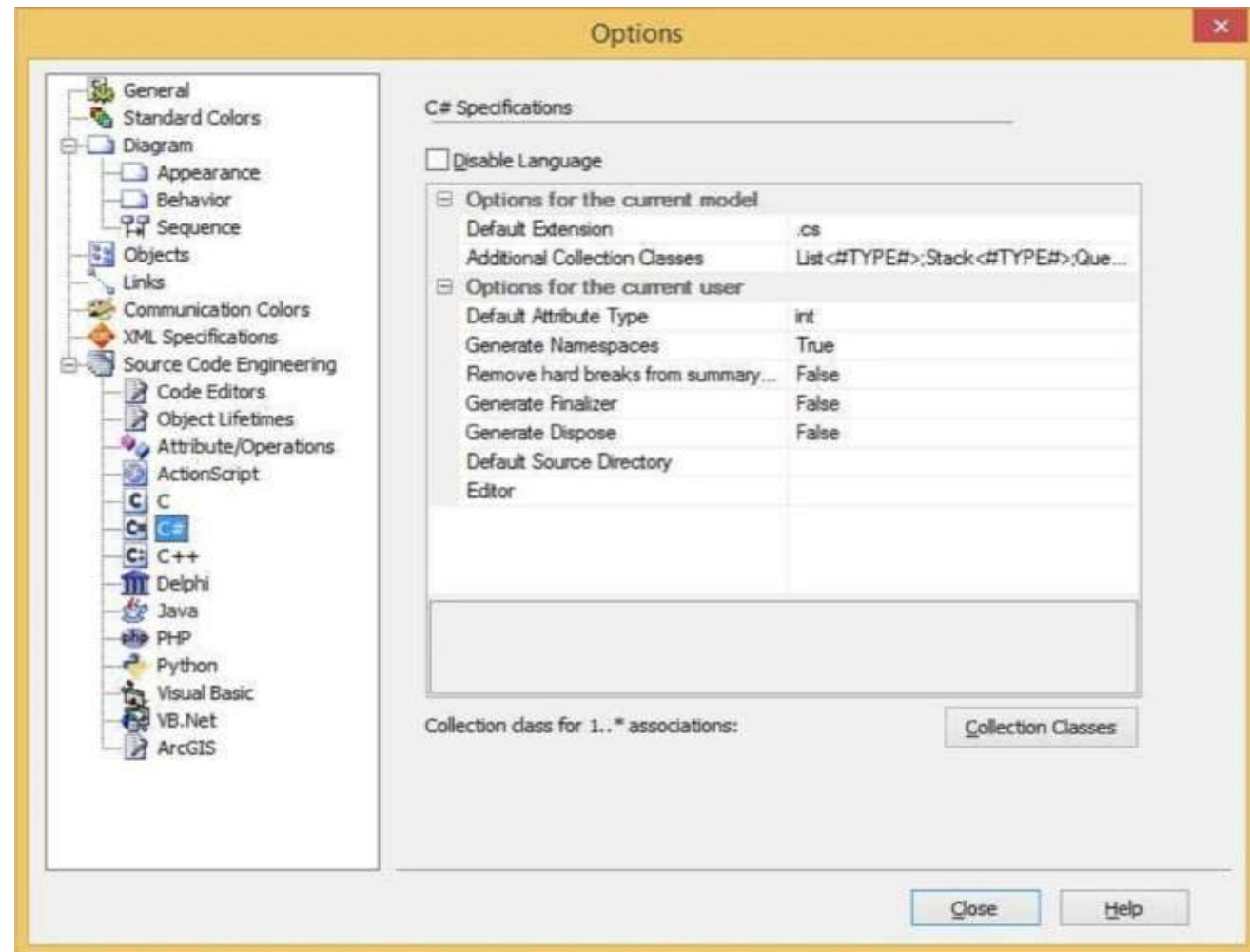
        ~Student(){

        }

    } //end Student
} //end namespace PSI.RVA
```

# UML generisanje koda

Podešavanje generisanja C# jezika.

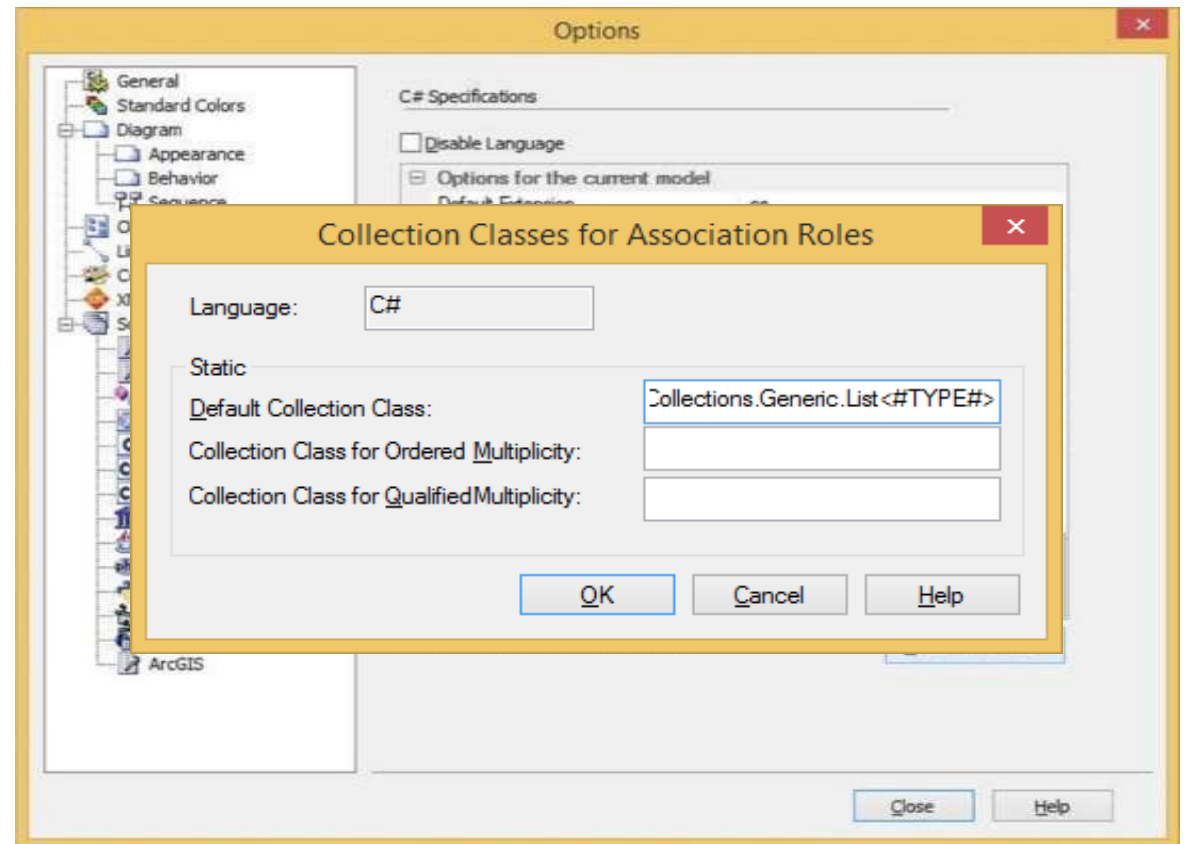


# UML generisanje koda

Podešavanje generisanja kolekcija:

System.Collections.Generic.

List<#TYPE#>



# UML generisanje koda

Dobijeni kod...

```
using System;
using PSI.RVA;
namespace PSI.RVA {
    public class Student : Osoba {

        private System.String index;
        public System.Collections.Generic.List<Predmet> Predmet;
        public System.Collections.Generic.List<Ispit> Ispit;
        public PSI.RVA.Odsek Odsek;

        public Student(){

        }

        ~Student(){

        }

        public String Index{
            get{
                return index;
            }
            set{
                index = value;
            }
        }

    } //end Student
} //end namespace PSI.RVA
```