Visoka tehnička škola Niš

Savremene računarske tehnologije

Predmet: **Objektno orijentisano programiranje - OOP**

Prof. dr Zoran Veličković, dipl. inž. el.

2019/20.

Prof. dr Zoran Veličković, dipl. inž. el.

Objektno orijentisano programiranje - OOP

Principi objektno orijentisane paradigme (2)

Sadržaj

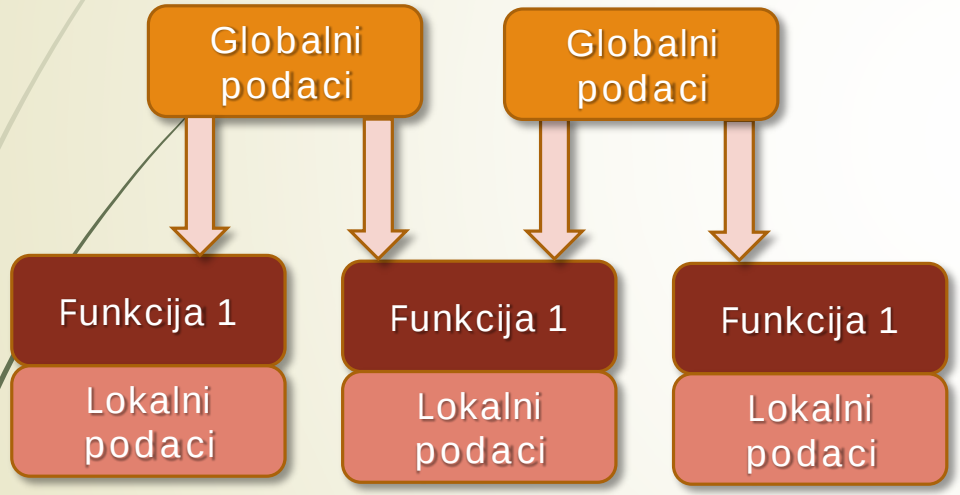
- **OO PARDIGMA**
  - Karakteristike OO paradigme
- **STUBOVI OO PARADIGME**
  - **Apstrakcija**
    - Podataka
    - Programskog koda
  - **Kapsuliranje**
    - Klase
    - Objekti
  - **Nasleđivanje**
    - Nasleđivanje klasa
    - Instanciranje klasa
  - **Polimorfizam**
    - Korišćenje stek strukture

Objektno orijentisana paradigma (1)

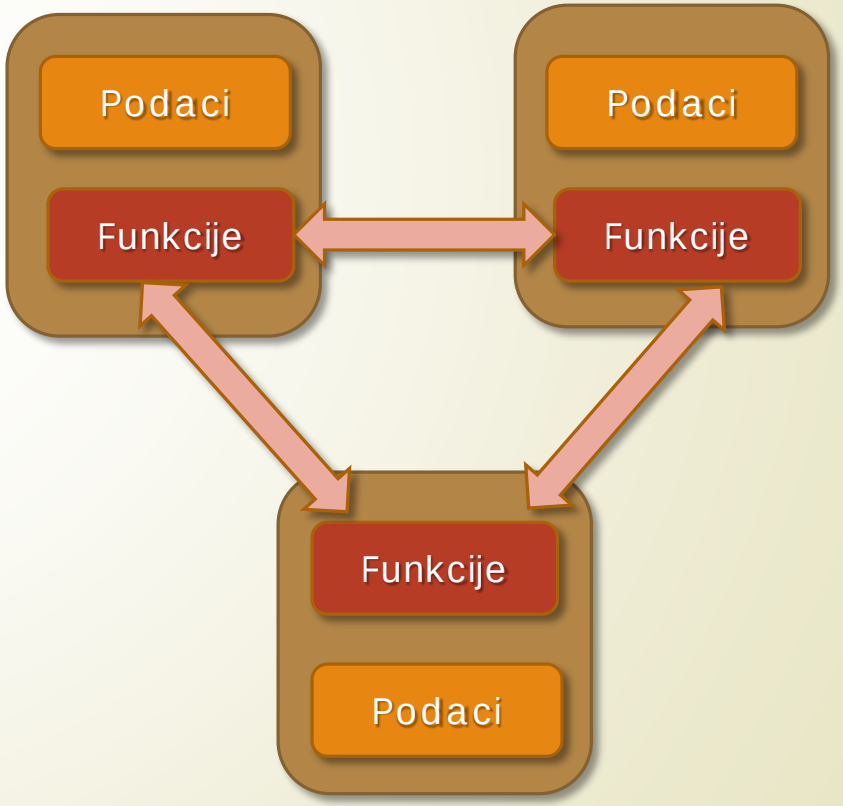
- **U OBJEKTNO ORIJENTISANOJ** (OO) **PARADIGMI**, program se sastoji od **OBJEKATA** koji međusobno interreaguju.
- **OBJEKTI** kapsuliraju **PODATKE** i **ALGORITME** koji rade sa tim podacima.
- **PODACI** definišu **STANJE** objekta, a **ALGORITMI** (kodirane metode) definišu **PONAŠANJE** objekta.
- OO paradigma se zasniva na međusobnom **KOMUNICIRAJU** između objekata razmenom poruka.
- Kada objekat primi poruku, on reaguje izvršavanjem jednog od svojih **ALGORITAMA**, koji mogu **MODIFIKOVATI** njegovo **STANJE**.
- Već znate, podaci i algoritmi su **RAZDVOJENI** u imperativnoj i funkcionalnoj paradigmi.
- Kod **OBJEKTNO ORIJENTISANE PARADIGMIE** podaci i algoritmi su **KOMBINOVANI U JEDNOM ENTITETU**, koji se naziva **OBJEKT**.

Objektna nasuprot funkcionalne paradigme

Proceduralna paradigma



Objektno orijentisana paradigma



Objektno orijentisana paradigma (2)

- **KLASE** su **OSNOVNE JEDINICE PROGRAMIRANJA** u objektno orijentisanoj paradigmi.
- Slični objekti su **GRUPISANI U JEDNU DEFINICIJU** koja se zove **KLASA**.
- Definicija klase se koristi za **KREIRANJE OBJEKTA** - objekt je poznat i kao **INSTANCA KLASE**.
- **KLASA** se sastoji od
  1. **PROMENLJIVIH** i
  2. **METODA** instance.
- **VREDNOSTI PROMENLJIVIH** instance objekta određuju **STANJE OBJEKTA**.
- Različiti objekti klase održavaju **SVOJA STANJA ODVOJENO** - svaki objekat klase ima **SVOJU KOPIJU** promenljive instance.
- Stanje objekta čuva se **PRIVATNO** za taj objekat – stanju objekta se **NE MOŽE** pristupiti ili direktno modifikovati izvan objekta.
- **METOD** je kao procedura (ili potprogram) u proceduralnoj paradigmi - **METODE** mogu pristupiti/modifikovati stanje objekta.

OO paradigma: klase i objekti

- **KLASA** predstavlja programski kod koji **DEFINIŠE OBJEKT**.
- Dakle, **KLASE** služe za kreiranje objekata korišćenjem ključne reči **new** u programskim jezicima, primer: Java, C#, JavaScript, VB .NET, ...
- Klasa je **APSTRAKTNA PREDSTAVA** stvari iz realnog života i predstavlja **OSNOVNI OBLIK** iz koga se kreiraju **PRIMERCI** (instance) određenih objekata.
- Na osnovu klase može se kreirati **NEOGRANIČENI BROJ OBJEKATA** (ograničenja su vezana samo za primenjenj hardver).
- Svaki od kreiranih objekata predstavlja **PO JEDA PRIMERAK** te klase (svi objekti su **ISTOVETNI** ostalima, osim što sadrži **SOPSTVENE PODATKE**).
- Pristup objektima je omogućen samo preko **PODRŽANIH INTEFEJSA**.

Klasa Person – primer u Javi

```
package com.jdojo.concepts;
public class Person {
   private String name;
   private String gender;
   public Person(String initialName, String initialGender) {
   name = initialName;
   gender = initialGender;
 }
public String getName() {
  return name;
 }
public void setName(String newName) {
  name = newName;
 }
public String getGender() {
  return gender;
 }
}
```

Jedan konstruktor

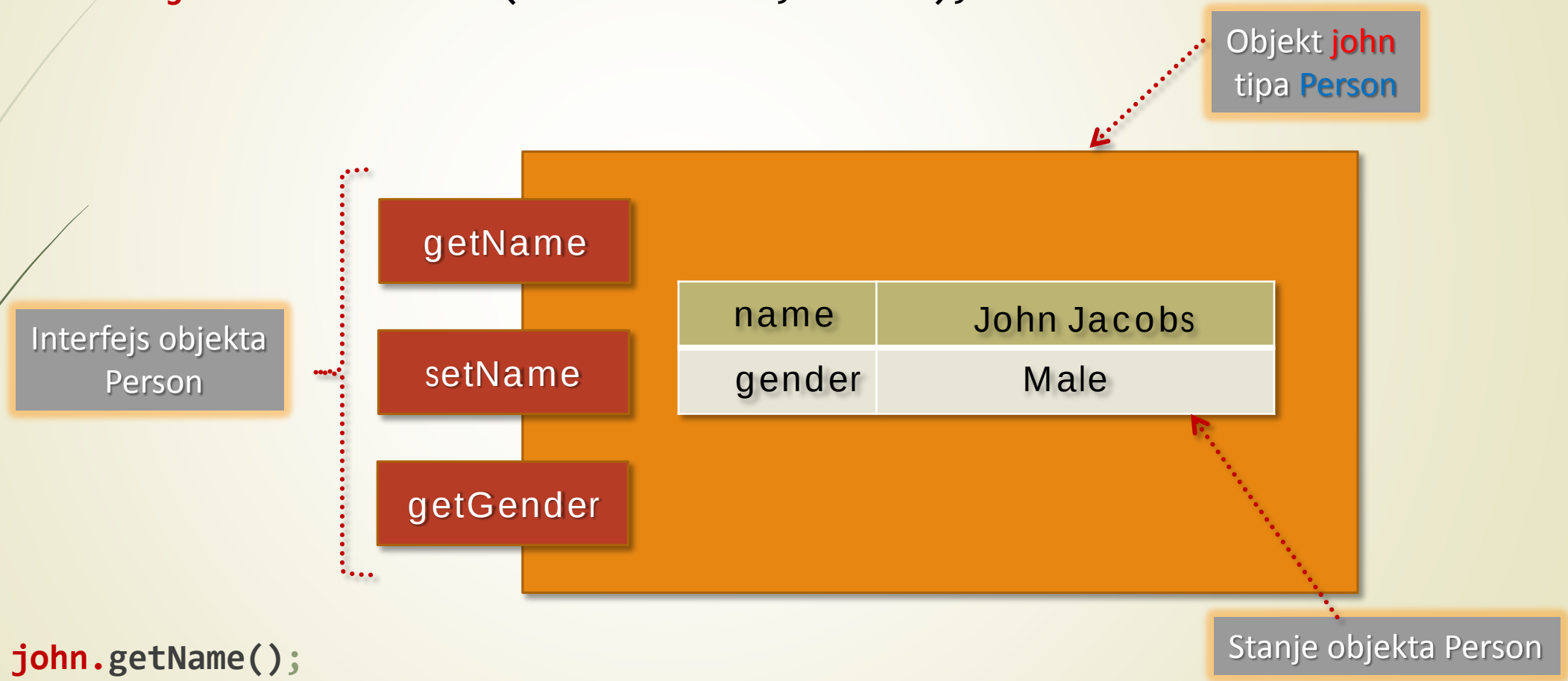
Klasa

Dve promenljive instanci

Tri metode

Klasa **Person** – primer u Javi

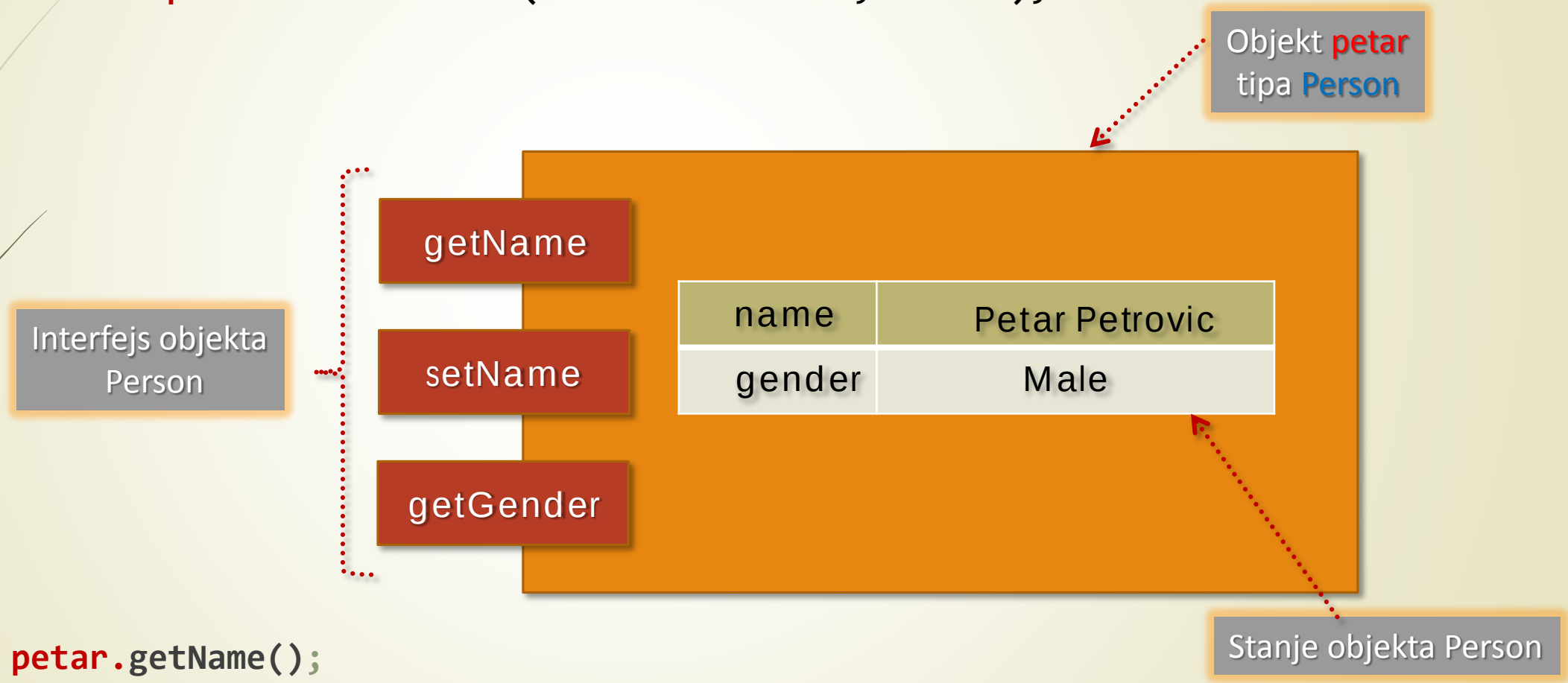
```
Person john = new Person("John Jacobs", "Male");
```




```
john.getName();
john.setName();
```

Kreiranje objekta **petar** tipa **Person**

```
Person petar = new Person("Petar Petrovic", "Male");
```




```
petar.getName();
petar.setName();
```

Karakteristike OO paradigme

- Kod **OBJEKTNO-ORIJENTISANE PARADIGME**, podaci i programi su kombinovani u **JEDAN ENTITET**, koji se naziva **OBJEKT** (engl. *object*).
- U objektno-orijentisanoj (**OO**) paradigmi, **PROGRAM** se sastoji od **INTERAKCIJE** između objekata koji kapsuliraju i podatke i programe.
- **PODACI** se koriste da definišu **STANJE OBJEKTA**, dok programi (algoritmi) definišu **PONAŠANJE** objekta.
- Objektno orijentisana paradigma podržava četiri glavna **OO PRINCIPA** (stuba), **AKPN**:
  - **Apstrakcija,**
  - **(En)kapsulacija,**
  - **Polimorfizam i**
  - **Nasljeđivanje.**

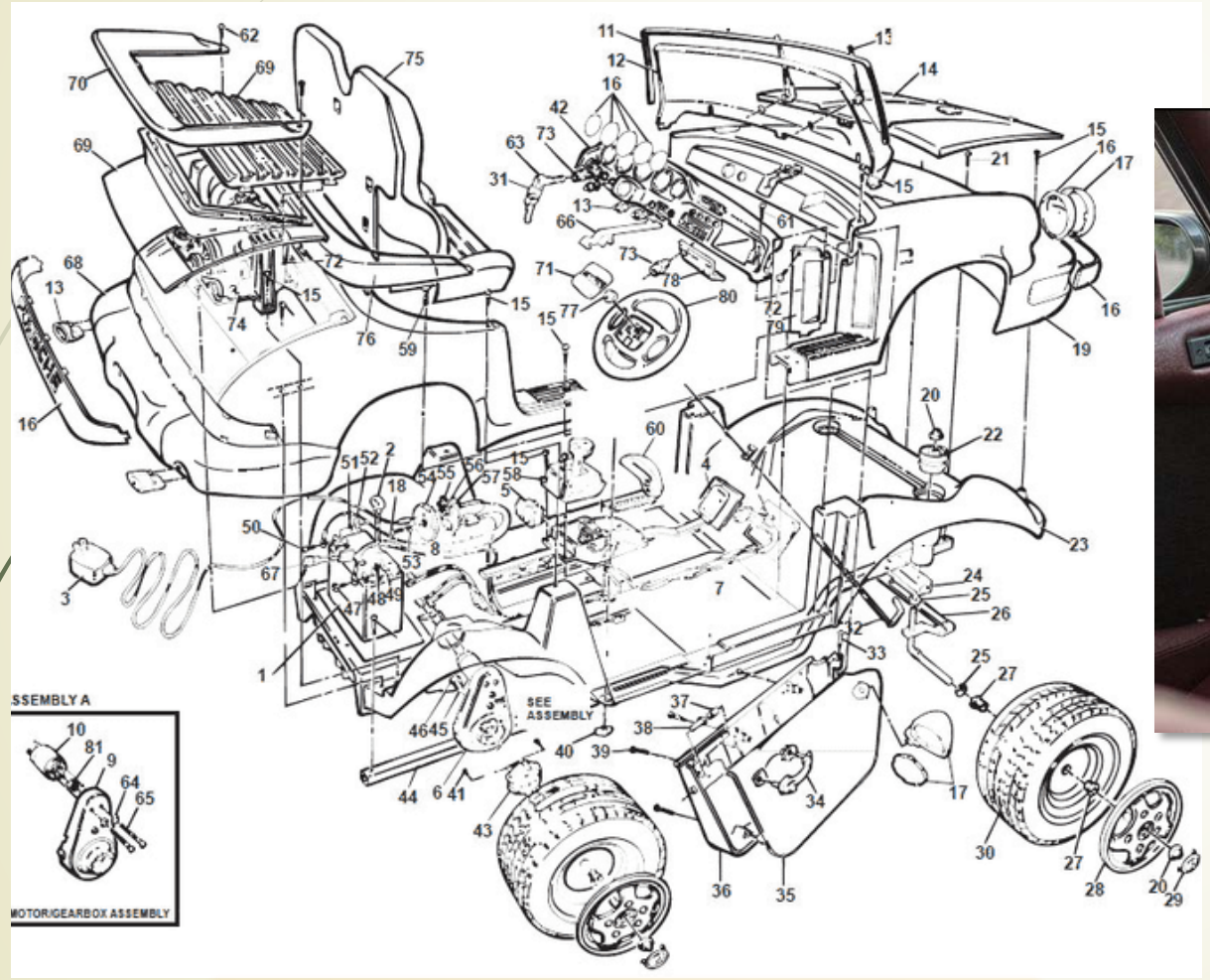
Stubovi objektno orijentisane paradigme

- **APSTRAKCIJA** je proces otkrivanja bitnih detalja entiteta, a pri tome se ignorišu **NEVAŽNI DETALJI** entiteta kako bi se smanjila složenost za korisnike. Smanjenje kompleksnosti entiteta pospešuje **ODRŽAVANJE I NADOGRADNJU** programa.
- **(EN)KAPSULACIJA** je proces **OBJEDINJAVANJA PODATAKA** i **PROGRAMASKOG KODA** koji radi upravo sa podacima koji pripadaju jednom entitetu. **BEZBEDNOST** korišćenja entiteta se na ovaj način značajno podiže.
- **NASLEĐIVANJE** se koristi za izvođenje **NOVOG TIPA** iz **POSTOJEĆEG TIPA**, čime se uspostavlja **RODITELJSKA ODNOS** između entiteta. Svojstva jednog entiteta se kroz **PROCES NASLEĐIVANJA** mogu preneti na drugi.
- **POLIMORFIZAM** omogućava entitetu da uzima **RAZLIČITA ZNAČENJA** u zavisnosti od prmenjenog **KONTEKSTA**. Različiti konteksti obezbeđuju i različita značenja entiteta uz primenu **ISTOVETNOG PROGRAMSKOG KODA**.

Stubovi OO paradigme: Apstrakcija (1)

- Objektno orijentisano (OO) programiranje je inspirisano pojmom apstrakcije koja se bazira na procesu **ZANEMARIVANJA DETALJA** u cilju **UOČAVANJA FUNKCIJE**.
- Primer automobila:
  - Da li se prilikom vožnje automobil doživljava kao složeni mehanizam od 10000 delova? Ne!
  - Automobil se prihvata kao dobro definisan objekat koji se ponaša na jedinstven način.
  - Apstrakcija kod automobila omogućava prevoženje na odredište bez opterećenja o složenosti mehanizma samog automobila.
- Isti princip apstrakcije se koristi i u programiranju. Programski kod može biti u rasponu od nekoliko redova pa do a nekoliko **MILIONA** linija - sve u zavisnosti od problema koji se rešava.
- Program se može se napisati **KAO MONOLITNA STRUKTURA** kod koje se izvršavanje započinje od prve, a završava do poslednom – milionitom linijom.
- Međutim, monolitne programske strukture duže od 50 linija postaju **TEŽE ZA PISANJE**, **RAZUMEVANJE** i **ODRŽAVANJE**!

Stubovi OO paradigme: Apstrakcija (2)



Automobil kao skup 10000 komponenti

Automobil kao OBJEKT

Apstrakcija programskog koda (1)

- **APSTRAKCIJA** u OO programiranju podrazumeva apstrakciju
  - programskog koda,
  - podataka.
- Zbog lakšeg održavanja, veliki - monolitni programski kod mora biti **DEKOMPONOVAN** na manje celine - **POTPROGRAME**.
- **POTPROGRAMI** moraju biti jednostavni i **DOVOLJNO MALI** da bi se razumeli, a potom kada se sastave u veću celinu, moraju **EFIKASNO REŠITI PRIMARNI PROBLEM**.
- Da bi se **KORIŠĆENJE** složenih programskih struktura i podataka **POJEDNOSTAVILO**, a istovremeno omogućilo **BEZBEDNO KORIŠĆENJE**, kreira se **ZAŠTITNA ČAURA** oko njih.
- Programima i podacima **UNUTAR ZAŠTITNE ČAURE** se može pristupiti samo **POD ODREĐENIM USLOVIMA** čime se sprečava proizvoljan – **NEKONTROLIANI PRISTUP**.
- Pojam **APSTRAKCIJE** je direktno povezan sa pojmom **KAPSULIRANJA** koji se odnosi na mehanizam **POVEZIVANJA PODATAKA I PROGRAMSKOG KODA** na koje se odnosi.

Apstrakcija programskog koda (2)

- **POTPROGRAM** definiše **INTERFEJS** za interakciju sa drugim entitetima.
- **UNUTRAŠNJI DETALJI** entiteta su **SKRIVENI** od spoljnog sveta.
- Sve dok je interfejs potprograma **NEPROMENJEN**, promene u unutrašnjim **DETALJIMA NE BI TREBALO** da utiču na druge potprograme koji su u interakciji sa njim.
- Sve ove karakteristike postižu se tokom **DEKOMPOZICIJE PROBLEMA** (ili programa koji rešava problem) koristeći proces koji se naziva apstrakcija.
- **APSTRAKCIJA** je način da se izvrši dekompozicija problema **FOKUSIRAJUĆI SE NA RELEVANTNE DETALJE** i ignorišući irelevantne detalje u određenom kontekstu.
- Neki detalji mogu biti **RELEVANTNI** u jednom kontekstu, a **IRELEVANTNI** u drugom.
- Dakle "**KONTEKST**" odlučuje koje su pojedinosti relevantne a koje irelevantne.

Apstrakcija podataka

- Objektno orijentisano programiranje pored apstrakcije **PROGRAMSKOG KODA** može koristiti i **APSTRAKCIJU PODATAKA**.
- U cilju apstrakcije podataka, definisana su sledeća **SVOJSTVA**:
  - **SKUP VREDNOSTI** koje se podacima mogu dodeliti,
  - **SKUP OPERACIJA** koje se mogu obaviti nad vrednostima u skupu,
  - **NAČIN PREDSTAVLJANJA** odnosno, čuvanja podataka.
- Programski jezici obezbeđuju **PREDEFINISANE TIPOVE** podataka, koji se nazivaju **UGRAĐENIM** ili **PRIMITIVNIM** tipovi podataka.
- **JAVA** ima ugrađene - primitivne tipove podataka (int, float, boolean, char, ...) kod kojih se **NE MOGU** menjati podrazumevana svojstva, ali se mogu davati **IMENA I VREDNOSTI** promenljivama.
- Mogu se definisati i **KORISNIČKI TIPOVI** podataka – **REFERENCNI** tipovi podataka.

Stubovi OO paradigme : Kapsuliranje (1)

- Pojam **KAPSULIRANJA** se odnosi na mehanizam koji povezuje **PODATKE** i **PROGRAMSKI KOD** koji manipuliše tim podacima.
- Programski kod koji je kapsuliran pravi **ZAŠTITNU ČAURU** oko podataka i naredbi nedozvoljavajući proizvoljan – nekontrolisani pristup.
- Pristupanje kapsuliranim podacima i naredbama je **STROGO DEFINISANO**.
  - **Primer iz automobila**: menjač brzina se može šaltati samo preko ručice menjača, i to nema uticaja na migavce automobila!
- Osnovna jedinica kapsuliranja u OO programiranju pa i Javi KLASA (engl. Class).
  - **ZAPAMTITE**: klase definišu nove tipove podataka!
- **ČLANOVI KLASE** (engl. *class member*) mogu biti:
  - programski kod i
  - podaci.

Stubovi OO paradigme: Kapsuliranje, Klase

- Zapamtite terminološke razlike - funkcije su:
  - **PROCEDURE** u proceduralnom programiranju, a
  - **METODE** u OO programiranju.

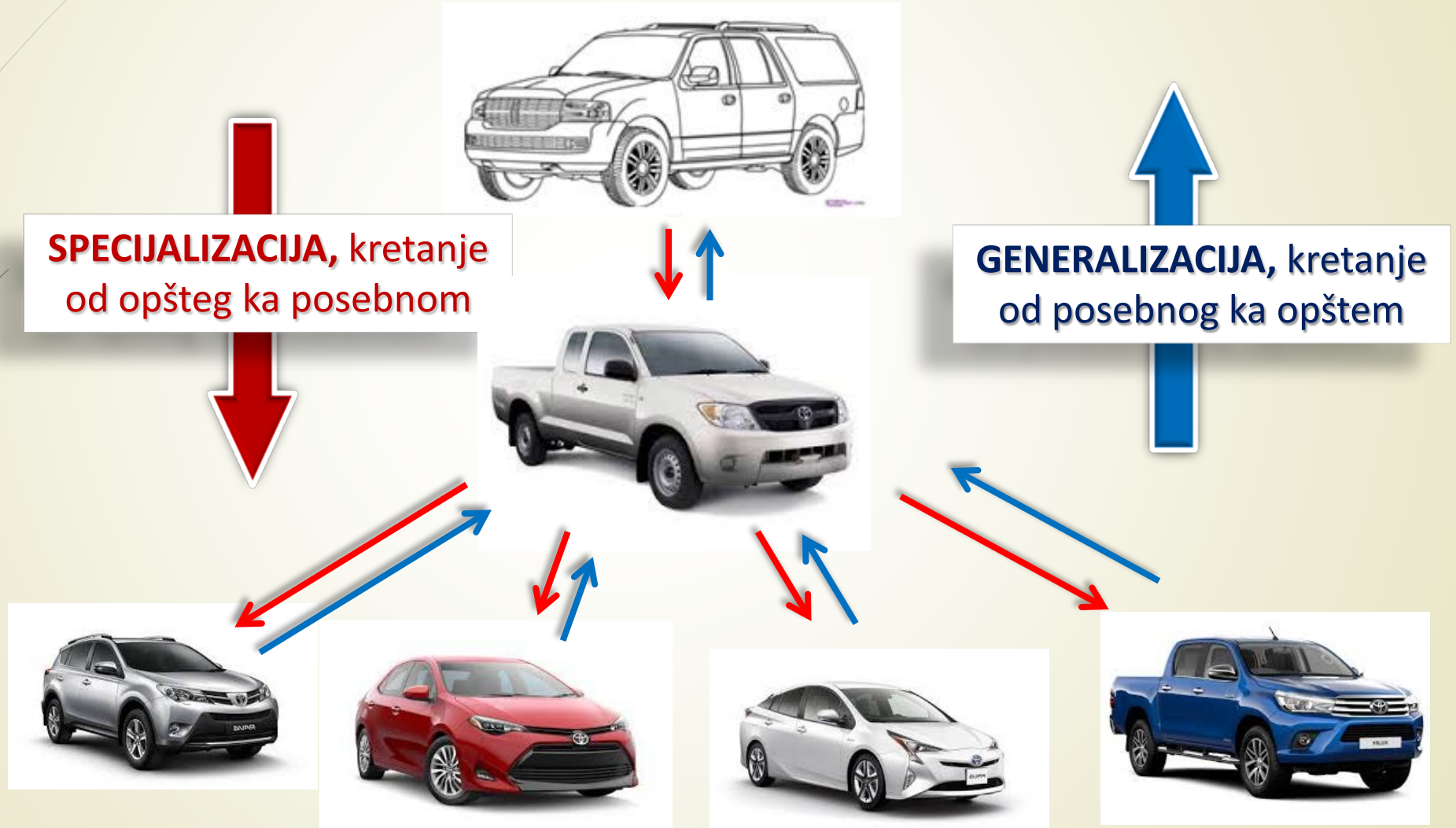
Stubovi OO paradigme : Kapsuliranje, podaci

- **PODACI** se nazivaju i članovima klase – i predstavljaju promenljive klase.
- **KLASOM** se definiše struktura i zajedničko ponašanje objekta (nastalog – iniciranog od klase).
- **OBJEKTI** su inicirani (kaže se i instancirani) primerci klase – još se kaže da su objekti instance klase.
- Klase uobičajeno imaju **JAVNI i PRIVATNI** deo.
- **JAVNI** deo klase je dostupan **SPOLJNIM KORISNICIMA** (engl. public).
- **PRIVATNI** deo klase je dostupan **SAMO ČLANOVIMA KLASE**.
- Na ovaj način **SE KAPSULIRA SLOŽENOST** programske strukture i podataka.
- **ISTOVREMENO** se na ovaj način obezbeđuje zaštita od zlonamernog pristupa podacima ili metodama.

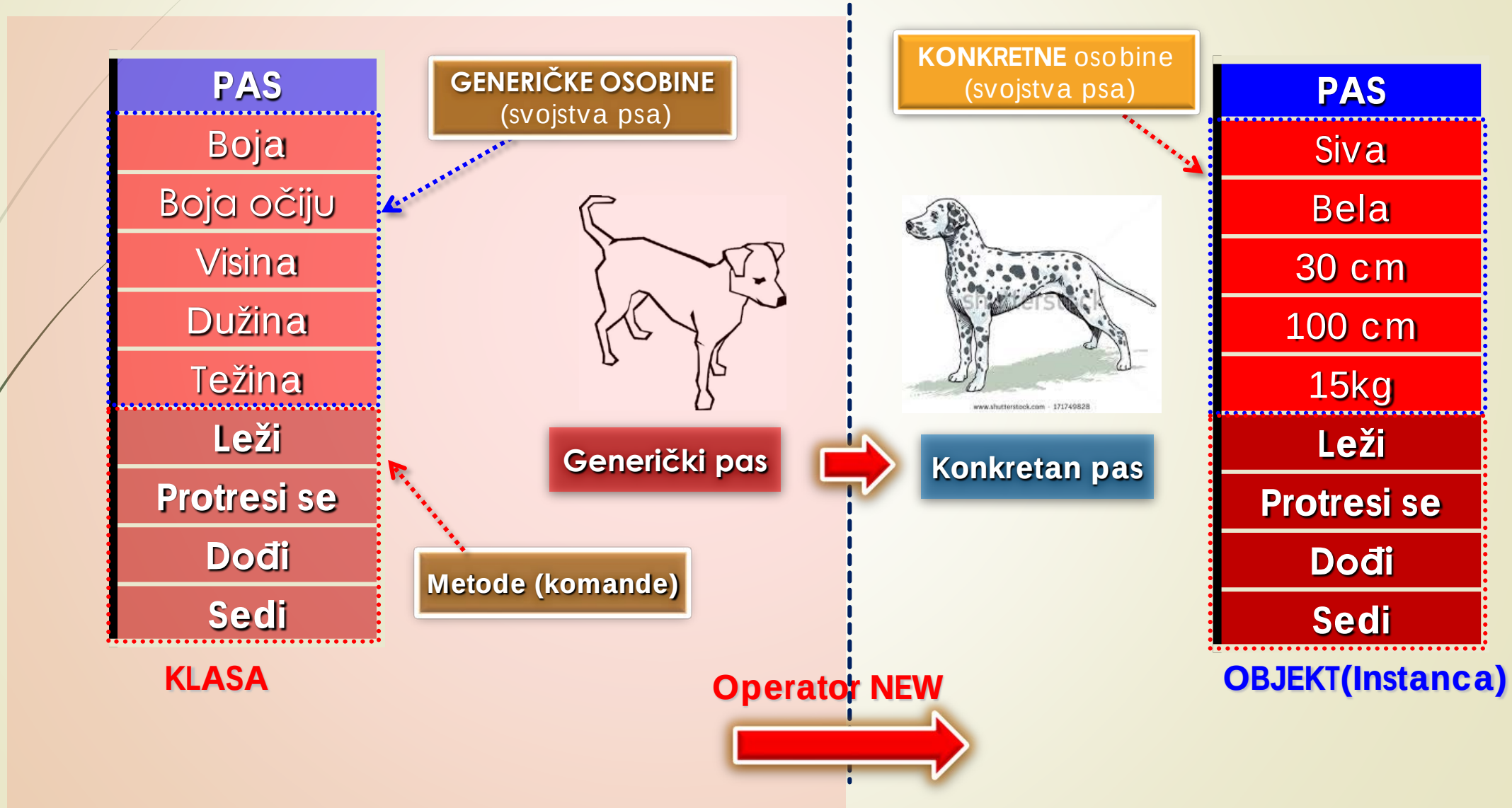
Stubovi OO paradigme : Nasleđivanje (1)

- **NASLEĐIVANJE** (engl. inheritance – nasleđe, baština) je proces u OO programiranju kojim jedan objekat nasleđuje **SVE OSOBINE** drugog.
- Ovaj koncept nasleđivanja podrazumeva **HIJERAHIJSKU** klasifikaciju objekata.
- Bez postojanja hijerarhije, svakom objektu bi morali **IZNOVA** zadavati sve njegove karakteristike!
- Sa konceptom nasleđivanja dovoljno je samo definisati **SPECIFIČNE KARAKTERISTIKE** tog objekta koje će ga činiti **JEDISTVENIM** u klasi, ali sa svim prethodno definisanim (prihvatljivim) karakteristikama.
- **OPŠTA SVOJSTVA** objekti nasleđuju od svojih "roditelja".
- Pogledajmo na sledećem slajdu ilustrovan **KONCEPT NASLEĐIVANJA** primenjen u OO programiranju.

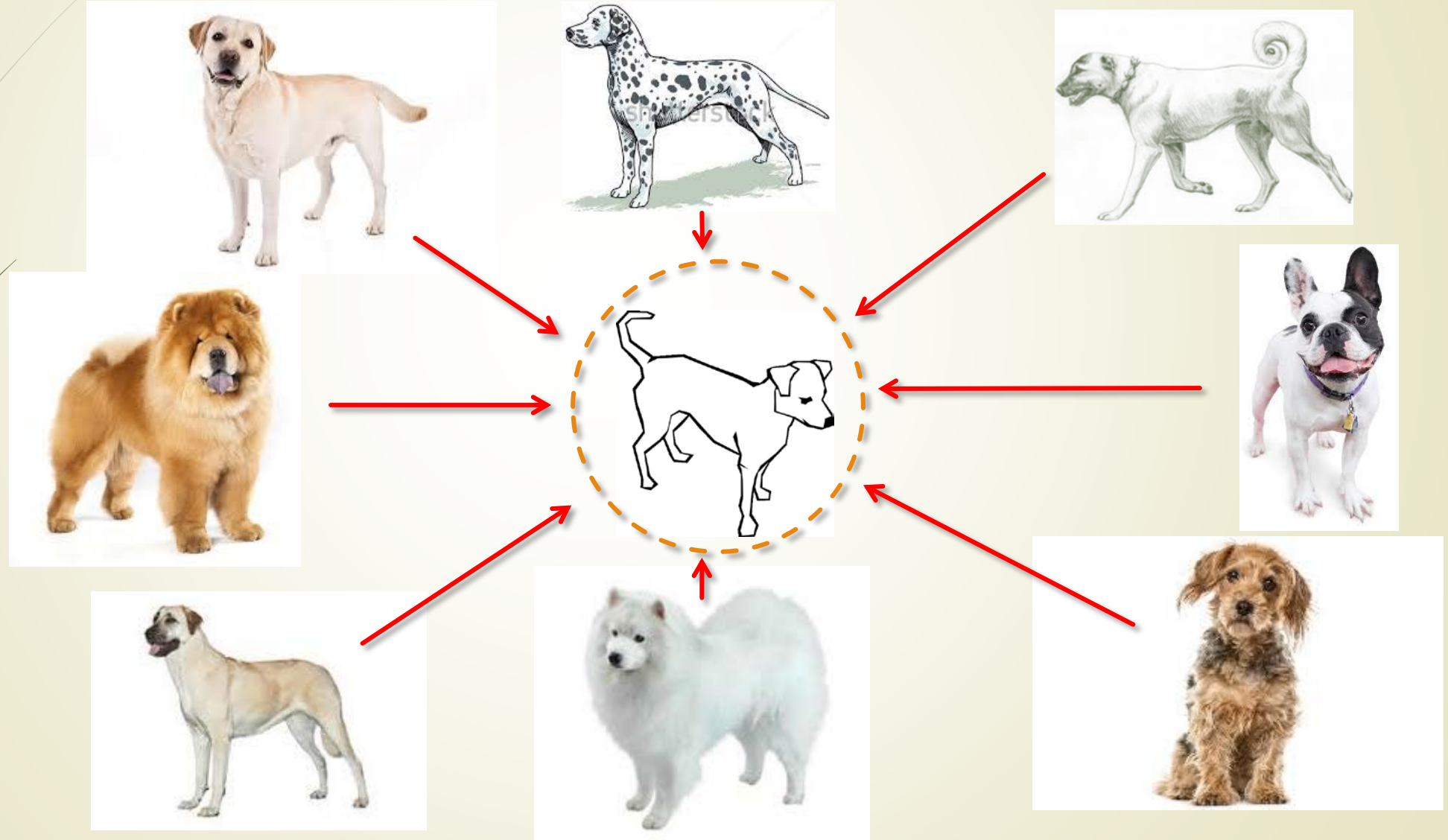
Stubovi OO paradigme : Nasleđivanje (2)



Stubovi OO paradigme: Klase i nasleđivanje

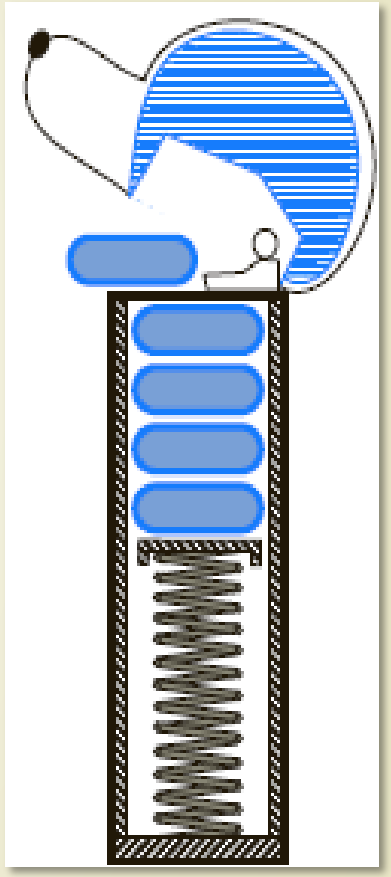


Stubovi OO paradigme : Instance klase pas



Stubovi OO paradigme: Polimorfizam (1)

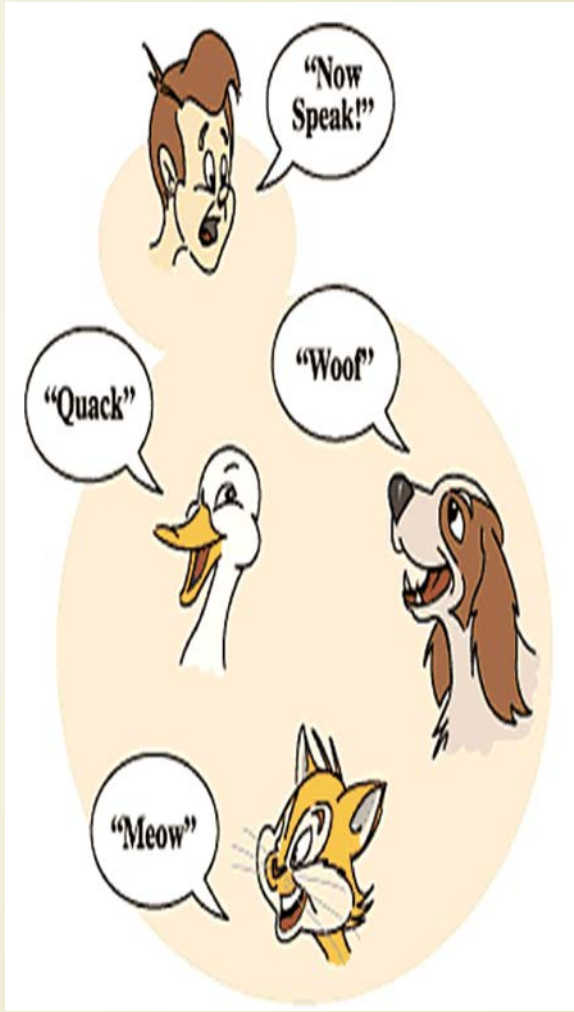
- **POLIMORFIZAM** se zasniva na **JEDINSTVENOM NAČINU** pristupa za opštu klasu akcija (kontekst poziva akcije određuje njenu specifičnost).
- **Primer_1**: Korišćenje čula mirisa kod psa.
  - Jednim istim čulom pas razlikuje i mačku i hranu!
- **Primer_2**: Korišćenje **STEK STRUKTURE** za različite tipove podataka.
  - **ALGORITAM** kojim se kreira i upravlja stek strukturom je UVEK ISTI i ne zavisi od toga da li se smeštaju iteger, char ili object tipovi podataka.
  - U OO programiranju je dakle moguće napisati **OPŠTI PROGRAM** za rad sa stekom, a metode će imati ista imena nezavisno od tipa podataka.
  - Kod proceduralnih i funkcionalnih jezika moraju se napisati **TRI VERZIJE METODA** - po jedna za svaki tip podataka.



Primer Stek-a

Stubovi OO paradigme: Polimorfizam (2)

- Kod OO programskih jezika - **PREVODILAC**, (**JIT** kompajler) određuje metodu koja će biti pozvana u zavisnosti od **KONTEKSTA** samog poziva!
- Ovo ima za posledicu **SMANJENJE SLOŽENOSTI** programskog koda.
- Tokom vremena, **MOŽE SE MENJATI UNUTRAŠNJA KONSTRUKCIJA KLASE** ne remeteći kod koji se oslanja na javni interfejs klasa.
- Koncept **Jave** je slično kao i **C#** zasnovan na specijalizovanom izvršnom okruženju – **Javinoj Virtualnoj Mašini** – JVM, odnosno **CLR**-u (eng. *Common Language Runtime*).
- Pogledajmo na sledećem slajdu koncept OO programskih jezika kojim se prevazilazi **PROBLEM HETEROGENOSTI TERMINALNE OPREME**, što je jedan od problema vezan za savremene primene.



Javina virtuelna mašina JVM

Izvorni kod

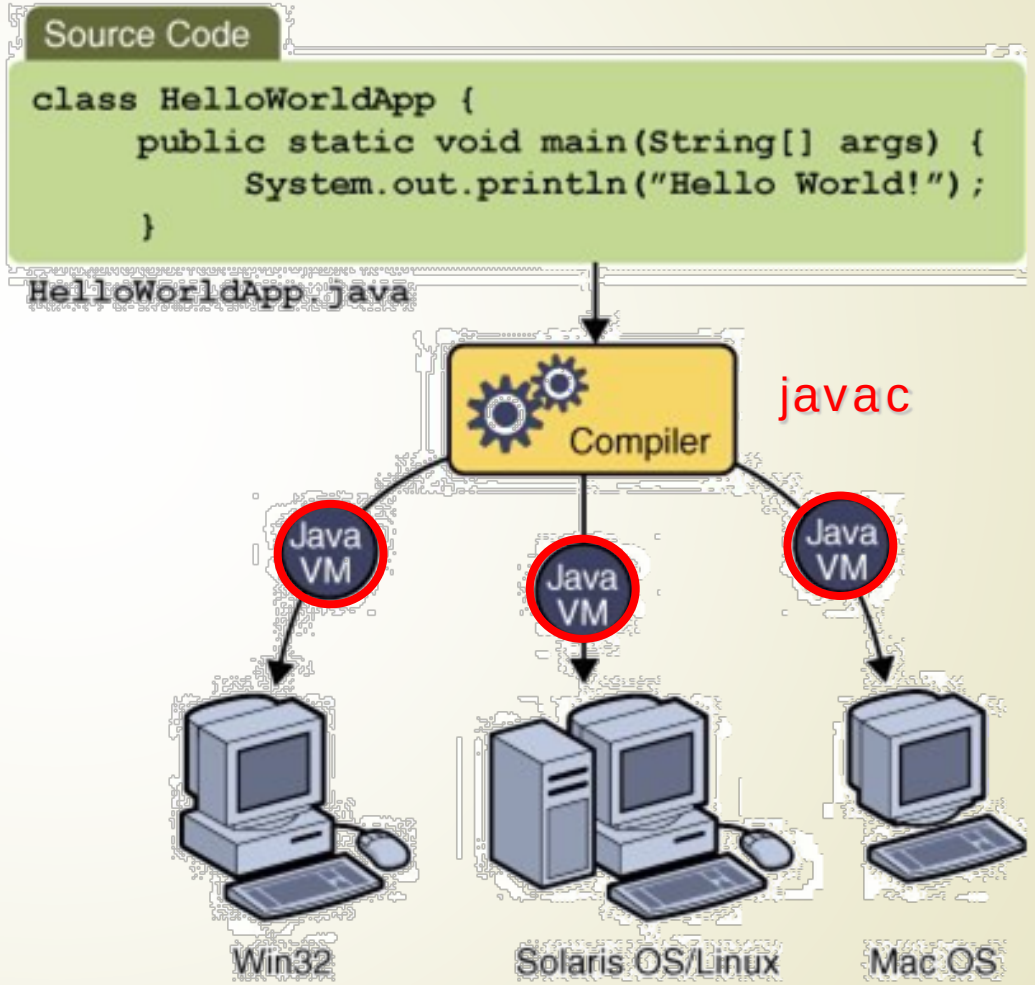
*.java

Bajtkod

*.class

Izvršni kod

*.exe



Stubovi OO paradigme: Polimorfizam (3)

- **POLIMORFIZAM** se ogleda u mogućnosti da se pišu metode koje mogu raditi sa objektima nastalim **iz VIŠE KLASA**.
- Objektima se pristupa iz **RAZLIČITIH KLASA** na **ISTI NAČIN**.
- Polimorfizam dopušta postojanje dvaju ili više klasa sa **RAZLIČITOM IMPLEMENTACIJOM** ali sa **ZAJEDNIČKIM SKUPOM METODA**, **SVOJSTAVA** ili **DOGAĐAJA**.
- **POLIMORFIZAM** omogućava pisanje programskog koda koji **NE MORA VODITI RAČUNA** o kom tipu objekta se radi **U TRENUTKU IZVRŠAVANJA**.
- U programskim jezicima podržava se polimorfizam na dva načina:
  - Putem **KASNOG** određivanja tipa podataka,
  - Primenom **VIŠESTRUKIH INTERFEJSA**.
- **POLIMORFIZAM** je usko povezan sa **NASLEĐIVANJEM** ali se može koristiti i **BEZ NJEGA**.

Uputstva za kreiranje klasa i metoda

- Pre implementacije klasa treba odlučiti **KOJE** su klase potrebne.
- Predloženi način za rešavanje ovog problema:
  - U opisanom problemu koji se rešava treba pronaći **OBJEKTE**.
  - Za pronađene objekte **ODREDITI SVOJSTVA**
    - **Karakteristike**
    - **Kvalitet**
    - **Metode** koje opisuju ponašanje objekata
- Analizirajte tekstualni opis problema
  - **IMENICE** su kandidati za KLASE
  - **GLAGOLI** su potencijalni kandidati za METODE
- Posle definisanja **OBJEKTA I NJEGOVIH SVOJSTAVA I METODA** može se napisati kod kojim se klasa opisuje.