



Академија струковних
студија Шумадија
одсек у Крагујевцу

Увод у програмирање

Презентација 11

Академија струковних студија Шумадија

Одсек у Крагујевцу

Студијски програм Информатика

Крагујевац, 2020. година



Објектно оријентисано програмирање

- Прошлог часа учили смо о појму **енкапсулација** - скривање детаља имплементације класе.
- Енкапсулација подразумева скривање података од спољних приступа.
- Примена енкапсулације је изузетно битна, поготово код великих пројеката.
- Поштовањем правила енкапсулације **обезбеђујемо** да објекти имају строго контролисане **улазе** и **излазе**, а самим тим смањујемо могућност грешке, логичке недоследности или грешака у програму.
- Шта се налази унутра !?





Објектно оријентисано програмирање

Наслеђивање (енгл. *inheritance*)

- Наслеђивање представља један од камена темељца ООП-а јер омогућава израду хијерархијске класификације.
- Применом наслеђивања можете да направите општу класу која дефинише заједничке карактеристике за скуп сродних елемената.
- Ту општу класу онда могу да наследе друге класе, при чему свака од њих додаје оно што је за њу јединствено.
- У Јавиној терминологији наслеђена класа се назива **наткласа** (енгл. *superclass*), а класа која је наслеђује се назива **поткласа** (енгл. *subclass*).
- Такође, овај однос је познат и као однос родитељ – дете.
- Поткласа је специјализована врста наткласе, која **наслеђује све чланове** који су дефинисани у наткласи **и додаје им** сопствене јединствене елементе.



Објектно оријентисано програмирање

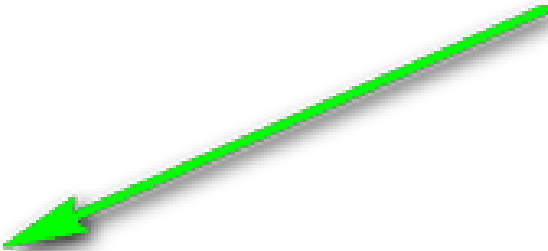
Језичке недоумице:

https://www.boske.rs/stranice/jednacenje_suglasnika_po_zvucnosti.html

поткровље или подкровље

поткровље или подкровље

Пише се **поткровље**.

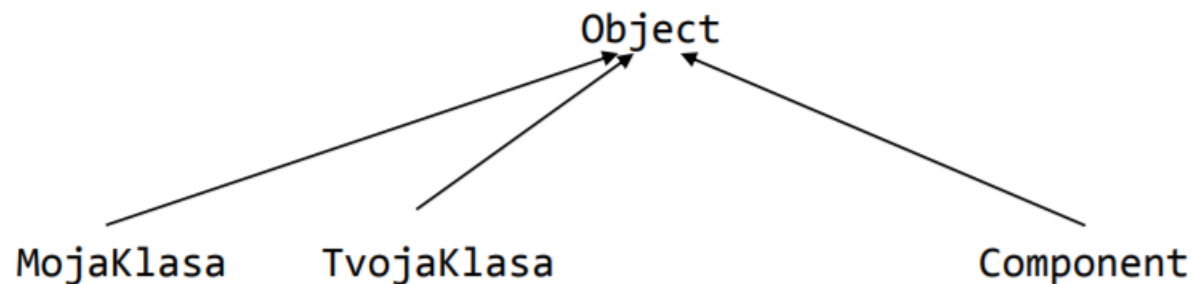


Префикс **под-** испред **к** даје **пот-** јер звучно **д** испред безвучног **к** прелази у свој безвучни пар **т**: **потковица**, **поткивач**, **потколеница**, **поткожна**, **поткошуља**, **потконтинент**, **поткрадати**, **поткратити**, **поткресати**, **поткупити**, **поткачити**, **Поткозарје**.



Објектно оријентисано програмирање

- Свака класа може да има неограничен број поткласа.
- Поткласе нису ограничене на променљиве, конструкторе и методе класе које наслеђују од своје родитељске класе.
- Поткласе могу додати и неке друге променљиве и методе или предефинисати старе методе.
- **У Јави класа може да има само једну наткласу, родитељску класу.**
- Дакле, за разлику од неких других програмских језика, **Јава не подржава вишеструко наслеђивање.**
- Све класе (и системске и наше) у Јави су директно или индиректно изведене из класе **Object**.

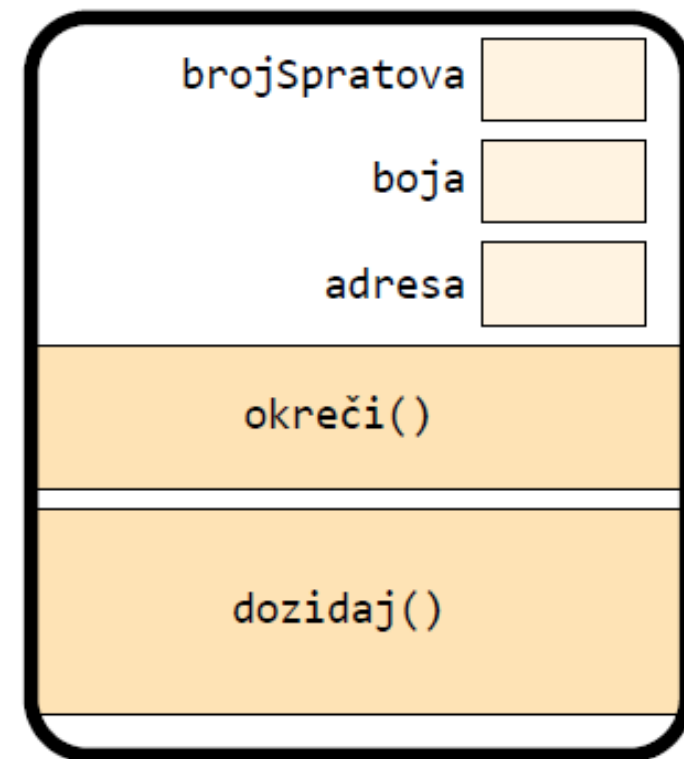




Објектно оријентисано програмирање

- Узећемо пример класе "Зграда", који смо помињали на ранијим предавањима:

```
public class Zgrada {  
    private int brojSpratova;  
    private String boja;  
    private String adresa;  
  
    public void okreči() {  
        System.out.println("Okreči zgradu u " + boja + " boju.");  
    }  
  
    public void dozidaј() {  
        System.out.println("Zgrada sada ima: " +  
            brojSpratova + 1 + " spratova");  
    }  
}
```

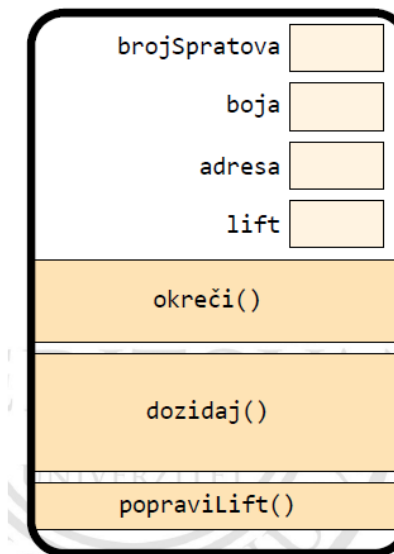




Објектно оријентисано програмирање

- Класу "Зграда" могу да наследе нпр. "Стамбена", "Пословна", ... зграда.

```
public class StambenaZgrada extends Zgrada {  
    private boolean liftRadi;  
  
    public void popraviLift() {  
        if (liftRadi == false) {  
            System.out.println("Pozovi majstora za lift.");  
        }  
    }  
}
```

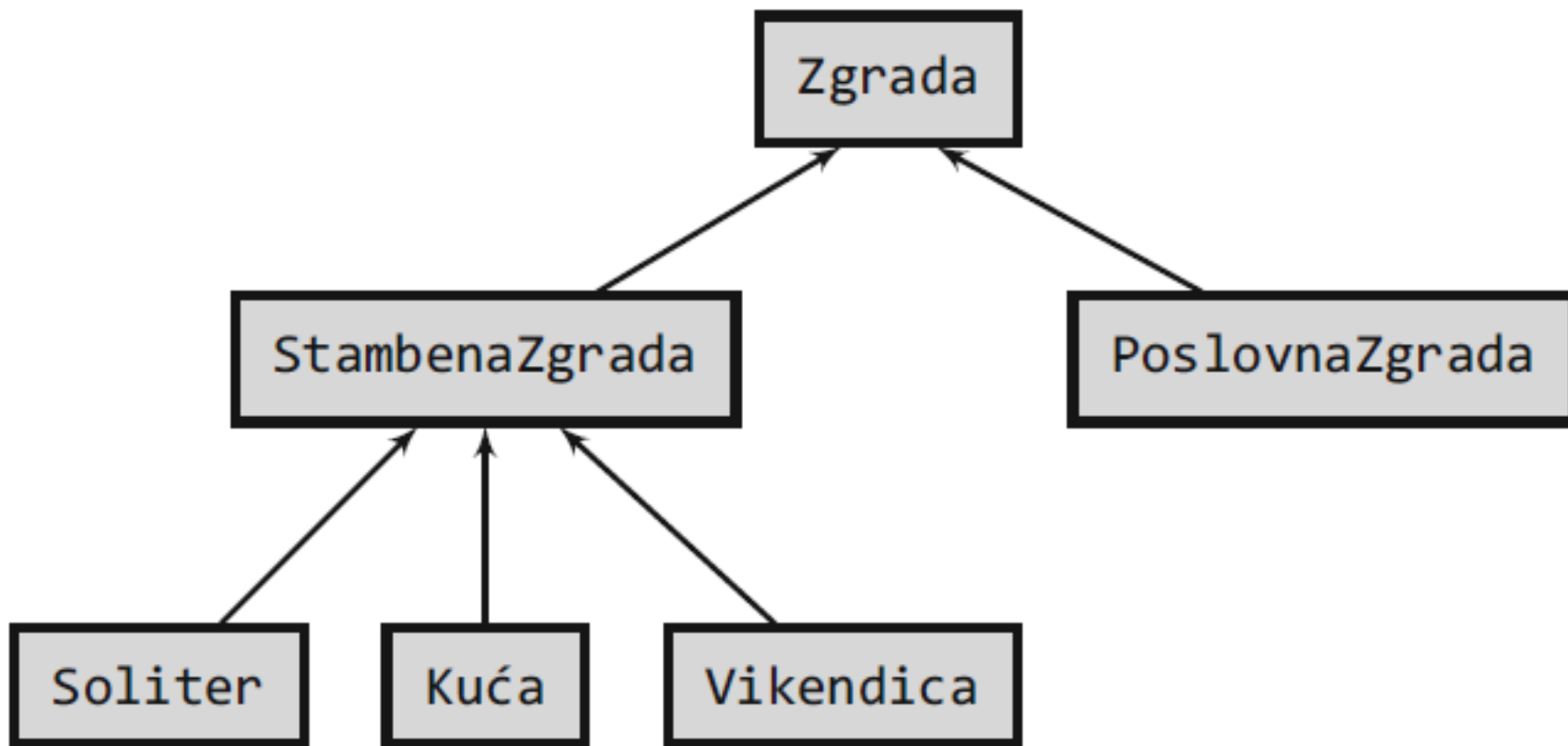


- Дакле, класа "Стамбена зграда" је наследила све што је дефинисано у класи "Зграда" и у њој је додатата једна њена карактеристика – лифт.
- Поткласа мора да користи кључну реч **extends** како би се декларисала као поткласа класе "Зграда".



Објектно оријентисано програмирање

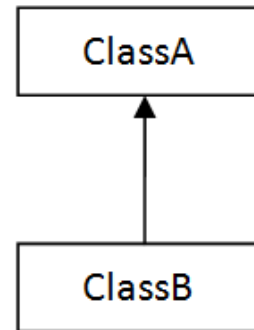
- Хијерархијска структура наших класа на примеру "Зграде" могла би да изгледа овако:



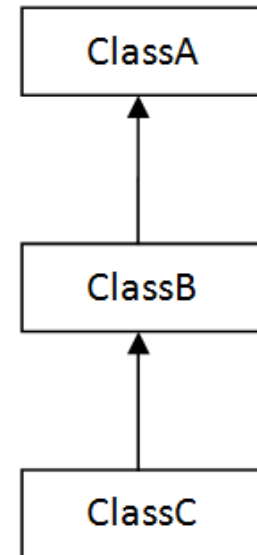


Објектно оријентисано програмирање

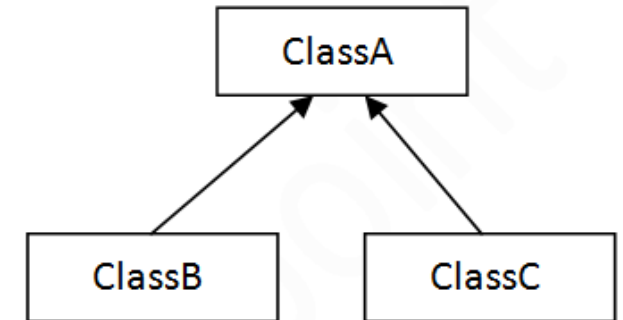
- Наслеђивање класа одговара релацији "јесте".
- "Стамбена зграда" **јесте** "Зграда".
- "Солитер" **јесте** "Стамбена зграда".
- На основу класе, у Јави могу постојати три врсте наслеђивања: **једноструко, вишеструко и хијерархијско.**



1) Single



2) Multilevel



3) Hierarchical



Објектно оријентисано програмирање

- Иако поткласа садржи све чланове своје наткласе, она не може да приступи члановима своје наткласе који су означени модификатором приступа **private**.

```
GlavnaKlasa.java x
1 public class GlavnaKlasa {
2     public static void main(String[] args) {
3         StambenaZgrada mojaZgrada = new StambenaZgrada();
4         mojaZgrada.
5     }
6 }
7
```

popraviLift()	void
dozidaj()	void
okreći()	void
equals(Object obj)	boolean
hashCode()	int
toString()	String



Објектно оријентисано програмирање

- Приступ вредностима атрибута који су означени модификатором приступа **private**, нашој поткласи **можемо обезбедити** помоћу конструктора (и он се такође наслеђује) или помоћу метода "гет" и "сет".

```

Zgrada.java x  StambenaZgrada.java x  GlavnaKlasa.java x
1  ► public class GlavnaKlasa {
2  ►     public static void main(String[] args) {
3      StambenaZgrada mojaZgrada = new StambenaZgrada();
4  mojaZgrada.
5  }
6  }
7

  ► popraviLift() void
  ► dozidaj() void
  ► getAdresa() String
  ► getBoja() String
  ► getBrojSpratova() int
  ► okreči() void
  ► setAdresa(String adr... void
  ► setBoja(String boja) void
  ► setBrojSpratova(int ... void
  ► equals(Object obi) boolean
```



Објектно оријентисано програмирање

- **Правило подтипа:** променљива класног типа А може садржати референце на објекте класе А, као и референце на објекте свих класа које наслеђују класу А.
- Наш пример: Објекат класе "Зграда" – "зграда" и објекат класе "Стамбена зграда" – "моја зграда". **Обрнути случај не важи:**

```
Zgrada.java x  StambenaZgrada.java x  GlavnaKlasa.java x
1  public class GlavnaKlasa {
2      public static void main(String[] args) {
3          Zgrada zgrada = new Zgrada();
4          StambenaZgrada mojaZgrada = new StambenaZgrada();
5          zgrada = mojaZgrada;
6          mojaZgrada = zgrada;
7      }
8  }
9
```

Incompatible types.
Required: **StambenaZgrada**
Found: **Zgrada**



Објектно оријентисано програмирање

- Пошто је енкапсулација такође један од главних принципа ООП-а, видели смо да се помоћу "гет" и "сет" метода може обезбедити приступ приватним члановима наткласе.
- Међутим, када год поткласа треба да се "обрати" својој непосредној наткласи, она то може да "уради" помоћу кључне речи **super**.
- Кључна или резервисана реч **super** има два општа облика употребе.
- Први начин позива конструктор класе.
- Други начин се користи за приступање члану наткласе који је сакривен чланом поткласе.
- Заклоњени (сакривени) чланови базне класе **не могу се директно користити** у проширеној класи.
- Ово ће бити јасније уз пример – **пример 1 (на крају документа)**.



Објектно оријентисано програмирање

- Са **super** позивамо **надјачану** методу из базне класе, а додатним наредбама проширујемо њену функционалност.
- Редифинисани метод може да врати вредност која је подтип вредности коју враћа метод из надкласе што може додатно да створи конфузију приликом програмирања и интензивног наслеђивања класа.
- Због овога се препоручује употреба посебне анотације (информације које нису део самог програма) **@Override**.
- Анотација **@Override** јасно указује на то да се метод редифинише али је додатно олакшица и за компајлер јер он може лакше да открије проблеме у наслеђивању и пријави адекватне грешке (нпр. ако је име редифинисаног метода погрешно наведено или није адекватан број и тип параметара).



Објектно оријентисано програмирање

- Већ је напоменуто да све класе (и системске и наше) у Јави су директно или индиректно изведене из класе **Object**.
- Класа **Object** дефинише основне могућности које имају сви објекти.
- Класа **Object** садржи методе које наслеђују све класе:
 - **equals()**
 - **hashCode()**
 - **toString()**
 - **clone()**
 - ...
- Свака класа која није експлицитно дефинисана као поткласа друге класе, постаје аутоматски поткласа класе **Object**.
- Променљива типа **Object** може садржати референцу на објекат било ког типа – ово некада зна да буде веома корисно.



Објектно оријентисано програмирање

- Композиције и агрегације омогућују конструисање комплексних класа које инкорпорирају (сједињују, придружују, ...) друге објекте.
- Они успостављају релацију **ИМА** између класа.
- На пример: "Аутомобил" **ИМА** "Мотор", "Купац" **ИМА** "Платну картицу".
- У компонованом објекту, објекти компоненте од којих је направљена композиција постоје и користе се једино преко компонованог објекта.
- Ако се компоновани објекат уништи, онда се уништавају и објекти компоненте.
- На пример, објекат "Запослени" **ИМА** име које је реализовано као "Стринг" објекат.
- Нема потребе да тај објекат остане у меморији уколико се уништава објекат "Запослени".



Објектно оријентисано програмирање

- У агрегацији, објекти од којих је направљена агрегација могу (али није неопходно) да егзистирају независно од њихове употребе у агрегираном објекту.
- У зависности од структуре агрегације, уништавање агрегације може, али и не мора да значи и уништавање њених компоненти (агрегираних објеката).
- На пример, објекат "Корисник" **ИМА** у оквиру себе објекат "Банковни рачун".
- Међутим, "Банковни рачун" може да представља заједнички рачун неколико корисника (мужа и жене, родитеља и детета).
- У том случају, није добро брисати објекат "Банковни рачун" само зато што је један објекат типа "Корисник" избрисан из система.



Литература

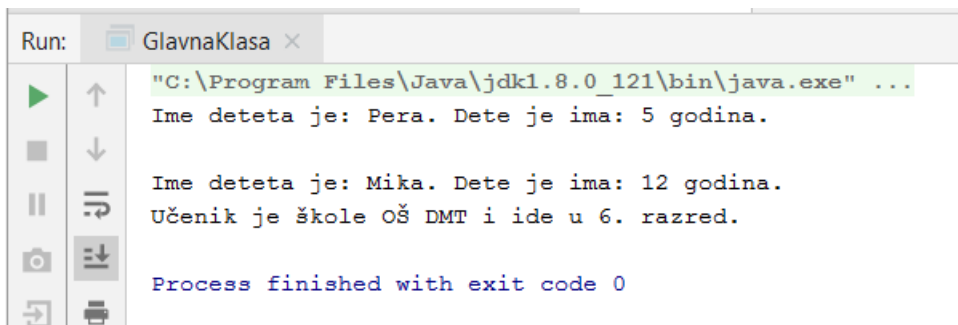
- Одабрана поглавља из књиге: [Јава JDK9: Комплетан приручник](#)
 - Аутор: *Herbert Schildt* (може и старије издање JDK7).
- Књига: [Објектно оријентисани начин мишљења](#)
 - Аутор: *Matt Weisfeld*
- Не морате да купујете наведене књиге.

ПРИМЕР 1

```
public class Dete {  
  
    private String ime;  
    private int uzrast;  
  
    public Dete(String ime, int uzrast) {  
        this.ime = ime;  
        this.uzrast = uzrast;  
    }  
  
    public void predstaviSe() {  
        System.out.print("Ime deteta je: " + ime + ". ");  
        System.out.println("Dete je ima: " + uzrast + " godina.");  
    }  
}  
  
public class Učenik extends Dete {  
  
    private String škola;  
    private int razred;  
  
    public Učenik(String ime, int uzrast, String škola, int razred) {  
        super(ime, uzrast);  
        this.škola = škola;  
        this.razred = razred;  
    }  
  
    @Override  
    public void predstaviSe() {
```

```
        super.predstaviSe();  
        System.out.print("Učenik je škole " + škola + " ");  
        System.out.println("i ide u " + razred + ". razred.");  
    }  
}
```

```
public class GlavnaKlasa {  
  
    public static void main(String[] args) {  
  
        Dete dete = new Dete("Pera", 5);  
        Učenik učenik = new Učenik("Mika", 12, "OŠ DMT", 6);  
  
        dete.predstaviSe();  
        System.out.println();  
        učenik.predstaviSe();  
    }  
}
```



```
Run: GlavnaKlasa x  
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...  
Ime deteta je: Pera. Dete je ima: 5 godina.  
  
Ime deteta je: Mika. Dete je ima: 12 godina.  
Učenik je škole OŠ DMT i ide u 6. razred.  
  
Process finished with exit code 0
```

ПРИМЕР 2

Definisati klasu **Vozilo** koja sadrži:

- **Javne** attribute **marka** i **model** i **privatni** atribut **maxBrzina**.
- **Konstruktor** sa parametrom koji **inicijalizuje** **privatni atribut**.
- Metode **GET** i **SET** za **privatni atribut maxBrzina**.
- Metod **infoVozilo** koji će da ispiše tekst "Vozilo ...", (... predstavljaju **poziv na sve attribute**).

Definisati klasu **Automobil** koja nasleđuje klasu **Vozilo** i koja sadrži:

- **Javni** atribut **brSedišta** i **privatni** atribut **zapreminaMotora**.
- **Konstruktor** koji pored **nasleđenog** atributa, **inicijalizuje** i **privatni atribut** iz klase **Automobil**.
- Metode **GET** i **SET** za **privatni atribut zapreminaMotora**.
- Metod **pokreniVozilo** koji će da ispiše tekst "Vozilo ...", (... predstavljaju poziv **na sve attribute – nasleđene i svoje**).

U glavnoj klasi napraviti **objekte**:

- **nekoVozilo** iz klase **Vozilo** koji ima definisane vrednosti **za sve attribute**.
- **mojAuto** iz klase **Automobil** koji ima definisane vrednosti **za sve attribute**.

Oba objekta treba da "**pozovu**" odgovarajuće **metode**.

Nakon toga, potrebno je **promeniti vrednosti svih atributa** za objekt iz klase **Automobil** i **ponoviti postupak "pozivanja"** odgovarajućeg **metoda**.

```
package zadatak_2;

public class Vozilo {
    public String marka;
    public String model;
    private int maxBrzina;

    public Vozilo(int maxBrzina) {
        this.maxBrzina = maxBrzina;
    }

    public int getMaxBrzina() {
        return maxBrzina;
    }
}
```

```

    public void setMaxBrzina(int maxBrzina) {
        this.maxBrzina = maxBrzina;
    }

    public void infoVozilo(){
        System.out.print("Vozlo " + marka + " " + model + " ");
        System.out.println("ima max brzinu od " + maxBrzina + " km/h.");
    }
}

```

```

package zadatak_2;

public class Automobil extends Vozilo {
    public int brSedišta;
    private double zapreminaMotora;

    public Automobil(int maxBrzina, double zapreminaMotora) {
        super(maxBrzina);
        this.zapreminaMotora = zapreminaMotora;
    }

    public double getZapreminaMotora() {
        return zapreminaMotora;
    }

    public void setZapreminaMotora(double zapreminaMotora) {
        this.zapreminaMotora = zapreminaMotora;
    }

    @Override
    public void infoVozilo() {
        super.infoVozilo();
        System.out.print("Auto ima " + brSedišta + " sedišta.");
        System.out.println(" Zapremina motora je " + zapreminaMotora + " l.");
    }
}

```

```
package zadatak_2;

public class G1_Zadatak_2 {

    public static void main(String[] args) {

        Vozilo nekoVozilo = new Vozilo(180);
        nekoVozilo.marka = "Ford";
        nekoVozilo.model = "Fiesta";

        Automobil mojAuto = new Automobil(260, 2.2);
        mojAuto.brSedišta = 5;
        mojAuto.marka = "BMW";
        mojAuto.model = "X5";

        nekoVozilo.infoVozilo();
        mojAuto.infoVozilo();

        System.out.println();

        mojAuto.model = "TT";
        mojAuto.marka = "Audi";
        mojAuto.brSedišta = 2;
        mojAuto.setZapreminaMotora(1.75);
        mojAuto.setMaxBrzina(240);

        mojAuto.infoVozilo();

    }
}
```

Run: Main x

▶

⬆

■

⏸

📷

📄

📁

🚀

⬆

⬇

↺

📄

🖨

🗑

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java.exe" ...  
Vozlo Ford Fiesta ima max brzinu od 180 km/h.  
Vozlo BMW X5 ima max brzinu od 260 km/h.  
Auto ima 5 sedišta. Zapremina motora je 2.2 l.  
  
Vozlo Audi TT ima max brzinu od 240 km/h.  
Auto ima 2 sedišta. Zapremina motora je 1.75 l.  
  
Process finished with exit code 0
```