



Академија струковних
студија Шумадија
одсек у Крагујевцу

Увод у програмирање

Презентација 12

Академија струковних студија Шумадија

Одсек у Крагујевцу

Студијски програм Информатика

Крагујевац, 2020. година



Објектно оријентисано програмирање

- Претходно предавање смо завршили са примером једноставне хијерархије класа, која се састојала од једне наткласе и једне поткласе.
- Наравно, ви можете да правите хијерархије које садрже произвољан број нивоа наслеђивања.
- Нпр. Ако имамо три класе, **A**, **B** и **C**, класа **C** може бити поткласа класе **B**, а класа **B** поткласа класе **A**.
- У оваквој хијерархији свака поткласа наслеђује све особине својих наткласа.
- У наведеном примеру класа **C** наслеђује све аспекте класе **B** и **A**.
- Да се подсетимо, једно од основних правила ОО пројектовања је да се наслеђивање класа представља помоћу односа "**јесте**".
- Уз помоћу примера 1 (дат у прилогу ове презентације) посматраћемо објектни модел хијерархије класе "Животиња".



Објектно оријентисано програмирање

- У примеру 1, почели смо са једном класом под именом "Животиња" у којој смо издвојили неке заједничке особине свих животиња.
- Затим смо у поткласама издвојили још неке заједничке особине одређених врста, а ако идемо још ниже по хијерархијској линији, издвојићемо и неке заједничке особине одређених раса.
- Овај концепт, који се понакад у литератури назива *генерализација-специјализација*, представља још један важан аспект **коришћења наслеђивања**.
- Наиме, што се више спуштамо низ "стабло наслеђивања" класе постају све специфичније. Који појам ће се користити зависи од "угла посматрања" на класе.
- Ако "ствари" посматрамо из угла једне од поткласа које дефинишу конкретну расу, класа одређене врсте је општа, односно "генералнија", па се релација између њих назива **генерализација**.



Објектно оријентисано програмирање

- Супротно претходном, ако "ствари" посматрамо из угла класе "Животиња", поткласе одређених врста су у односу на њу специфичније, па се релација између њих назива **специјализација**.
- Ако кренемо да пишемо код за овако представљени пример, почећемо са писањем од врха хијерархије, тј. класе "Животиња".
- Заправо, идеја наслеђивања јесте да се крене од општих особина ка специфичним и да се у том поступку издвајају заједничке особине.
- Један од основних **мотива** наслеђивања је избегавање редундантног кода – **кода који се понавља**.
- Код који је написан у наткласи не мора поново да се пише у поткласи, већ се исти аутоматски преноси (наслеђује) из наткласе.
- Увођењем претераног наслеђивања наш код може постати веома сложен, тј. што се више заједничких особина изваја наш код постаје сложенији.



Објектно оријентисано програмирање

- Како се крећемо уз хијерархију наслеђивања, класе постају све општије и све апстрактније.
- У неком тренутку наткласа постаје у тој мери општа да више представља основу за друге класе него класу чије конкретне примерке желимо да користимо.
- Сада долазимо до појма **апстрактне класе** које се не користе за креирање објеката, него као **основа за проширивање**.
- Апстрактна класа изражава **заједничка својства свих класа** које је проширују.
- Прављење инстанци апстрактне класе оператором **new** није могуће. Ако бисмо то покушали, Јава преводилац би пријавио грешку.
- Апстрактна класа најчешће садржи у себи имплементацију заједничких делова свих класа у хијерархији, док се имплементација специфичности појединих класа наследница **оставља за касније**.



Објектно оријентисано програмирање

- Апстрактна класа **може да садржи** апстрактне методе, **али и не мора**.
- Ако нека класа **садржи бар један апстрактни метод**, тада она такође мора бити **апстрактна**.
- Декларација апстрактног метода се састоји само од речи резервисане речи **abstract**, заглавља метода (немају тело) и тачке-зареза.
- Свака **неапстрактна класа** која је поткласа апстрактне класе са апстрактним методима мора садржати **имплементацију тих метода**.
- Апстрактна класа **може имати конструктор** (не може се креирати објекат помоћу њега), којим дефинише сопствена поља, а тај конструктор се потом позива од стране конструктора конкретне поткласе.
- Можемо да извучемо закључак да је наша класа "Животиња" добар пример да је представимо као апстрактну класу.



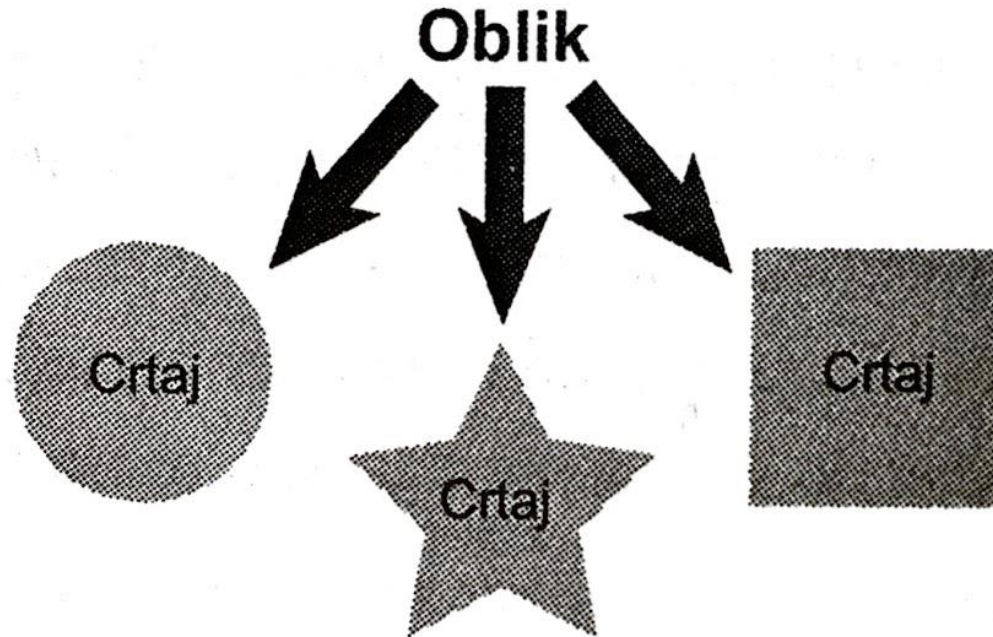
Објектно оријентисано програмирање

- Пре него што прикажемо реализацију примера 1, на једном једноставнијем примеру приказаћемо појам апстрактне класе.
- У литератури се често као пример за објашњавање појма апстрактне класе користи пример класе "Облик" и класама које је наслеђују: "Круг", "Квадрат", "Троугао", ...
- Наша релација **"јесте"** потврђује да свака изведена класа **"јесте"** заправо неки облик: круг јесете облик, квадрат јесте облик, ...
- Свака од изведених класа наслеђује све што има апстрактна класа "Облик", с тим да свака од њих има неке своје додатне карактеристике.
- Пошто је класа "Облик" представљена као апстрактна класа, начешће се у њој дефинише један апстрактан метод "цртај".
- Све изведене класе морају да имају имплементацију методе "цртај" и свака од њих је "одговорна" како ће да се поменути метод имплементира.



Објектно оријентисано програмирање

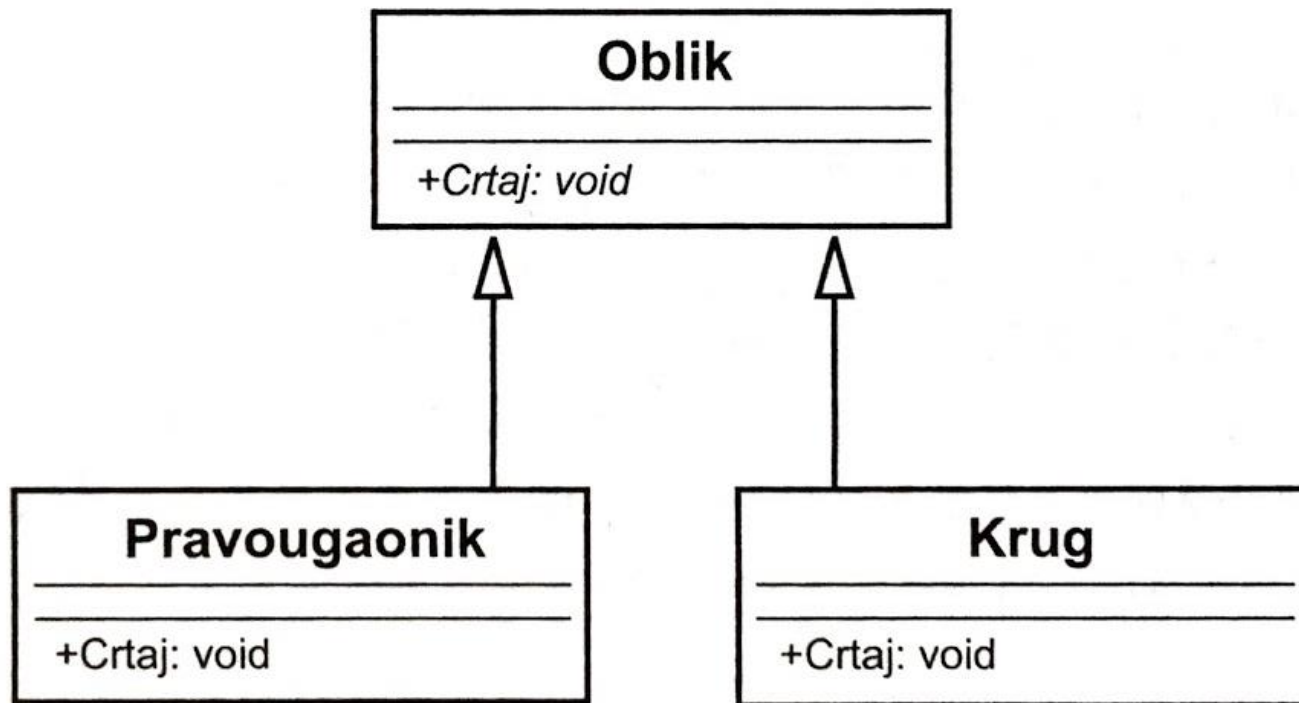
- У поменутом примеру, сваки појединачни објекат је одговоран за своје цртање:





Објектно оријентисано програмирање

- То можемо да представимо и дијаграмом класа:



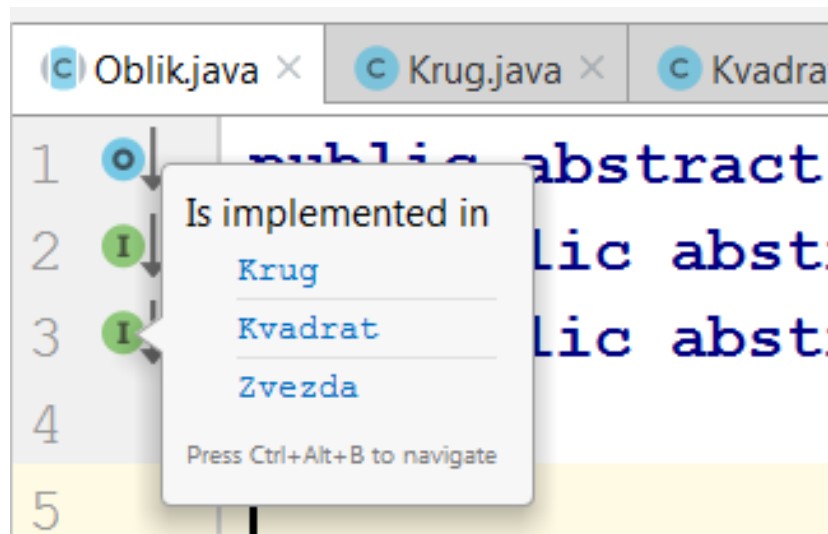
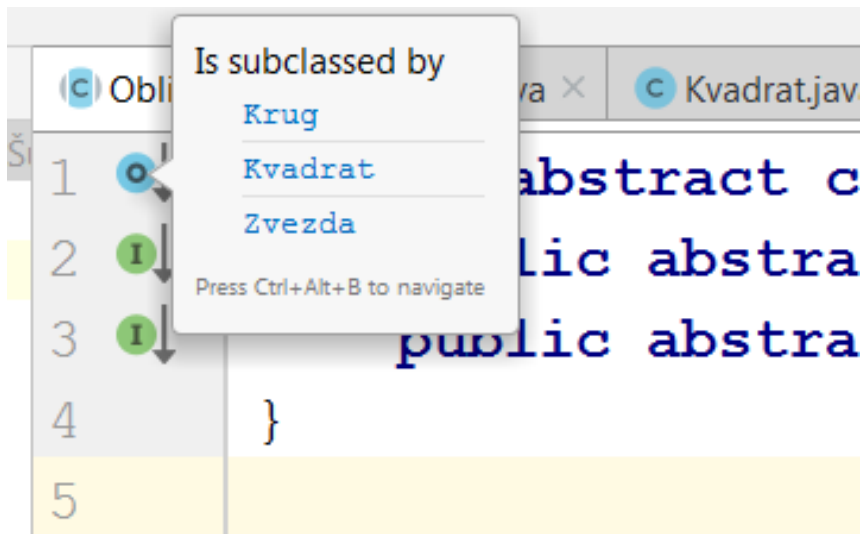
- Ако некоме кажете (мисли се на човека, људско биће) да нацрта неки облик, одмах ће уследити питање: "Који облик да нацртам?".
- Међутим, ако му кажете да нацрта круг, све је јасно, јер је круг одређен и познат облик – сви знају како се црта круг.



Објектно оријентисано програмирање

- На апстрактном нивоу, знамо да можемо нацртати сваки облик.
- Овим примером смо заправо "начели" тему полиморфизам, међутим, то ћемо на неком од наредних предавања да учимо.

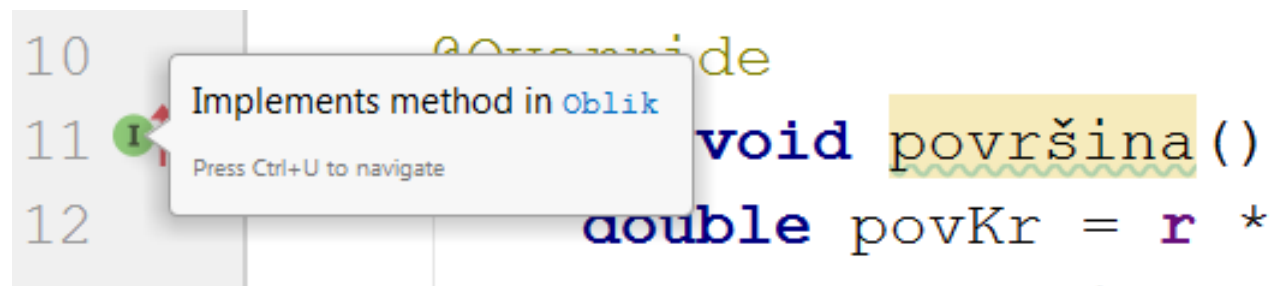
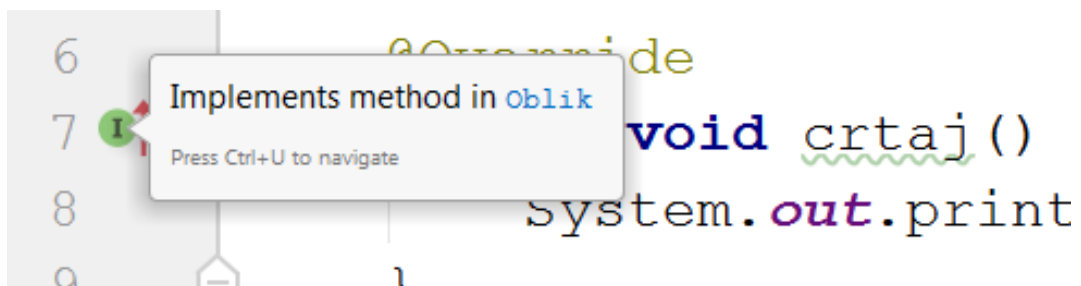
```
public abstract class Oblik {  
    public abstract void crtaj();  
    public abstract void površina();  
}
```





Објектно оријентисано програмирање

```
public class Krug extends Oblik {  
    private int r;  
    public Krug(int r) {  
        this.r = r;  
    }  
    @Override  
    public void crtaj() {  
        System.out.println("Iscrtava se krug.");  
    }  
    @Override  
    public void površina() {  
        double povKr = r * r * Math.PI;  
        System.out.println("Površina kruga je: " + povKr);  
    }  
}
```





Објектно оријентисано програмирање

```
public class Kvadrat extends Oblik {  
    private int a;  
  
    public Kvadrat(int a) {  
        this.a = a;  
    }  
  
    @Override  
    public void crtaj() {  
        System.out.println("Iscrtava se kvadrat.");  
    }  
  
    @Override  
    public void površina() {  
        int povKv = a * a;  
        System.out.println("Površina kvadrata je: " + povKv);  
    }  
}
```



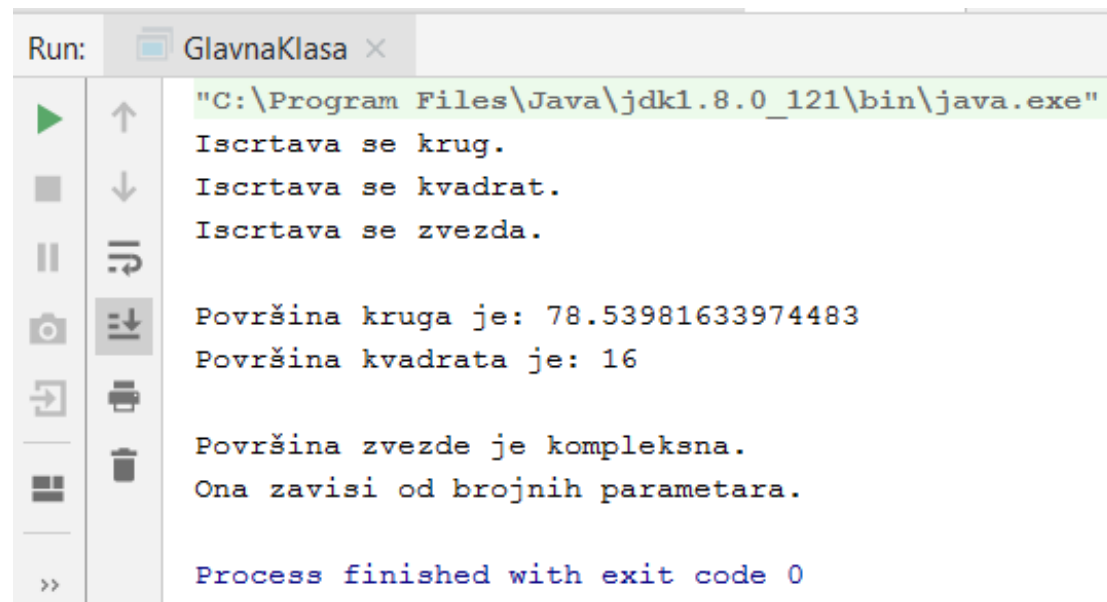
Објектно оријентисано програмирање

```
public class Zvezda extends Oblik {  
  
    @Override  
    public void crtaj() {  
        System.out.println("Iscrtava se zvezda.");  
    }  
  
    @Override  
    public void površina() {  
        System.out.println("\nPovršina zvezde je kompleksna.");  
        System.out.println("Ona zavisi od brojnih parametara.");  
    }  
}
```



Објектно оријентисано програмирање

```
public class GlavnaKlasa {  
  
    public static void main(String[] args) {  
  
        Krug kr = new Krug(5);  
        Kvadrat kv = new Kvadrat(4);  
        Zvezda z = new Zvezda();  
  
        kr.crtaj();  
        kv.crtaj();  
        z.crtaj();  
  
        System.out.println();  
  
        kr.površina();  
        kv.površina();  
        z.površina();  
    }  
}
```



Run: GlavnaKlasa x

"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe"

Iscrtava se krug.
Iscrtava se kvadrat.
Iscrtava se zvezda.

Površina kruga je: 78.53981633974483
Površina kvadrata je: 16

Površina zvezde je kompleksna.
Ona zavisi od brojnih parametara.

Process finished with exit code 0



Објектно оријентисано програмирање

- Иако се концепт апстрактних класа ослања на апстрактне методе, не постоји никаква препрека да "Облик" има извесну реализацију.
- Нпр., класа "Облик" је могла да има атрибут за боју и метод који задаје (има реализацију) ту боју.
- Метода којом се задаје боја је конкретна реализација, коју наслеђују класе "Круг", "Квадрат" и "Звезда".
- Једине методе које поткласе "Круг", "Квадрат" и "Звезда" морају да реализују се апстрактне методе декларисане у апстрактно класи "Облик".
- Сада ћемо да реализујемо Пример 1:
 - Креираћемо програм који садржи апстрактну класу Животиња коју наслеђују класе Птица, Пас и Мачка.
 - Класа Животиња треба да садржи метод оглашавање, такође класа Животиња може али и не мора да садржи атрибуте.
 - Ради показног примера, класа Животиња ће имати атрибут "име".



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

```
package rs.edu.asss.uup;

public abstract class Životinja {

    private String ime;

    public Životinja(String ime) {
        this.ime = ime;
    }

    public String getIme() {
        return ime;
    }

    public abstract void oglašavanje();
}
```

```
package rs.edu.asss.uup;

public class Pas extends Životinja {

    private String rasa;

    public Pas(String ime, String rasa) {
        super(ime);
        this.rasa = rasa;
    }
}
```

```
@Override
public void oglašavanje() {
    System.out.println("Pas " + rasa + " " + getIme() + " laje.\n");
}
}
```

```
package rs.edu.asss.uup;
```

```
public class Mačka extends Životinja {
```

```
    private String rasa;
```

```
    public Mačka(String ime, String rasa) {
        super(ime);
        this.rasa = rasa;
    }
}
```

```
@Override
public void oglašavanje() {
    System.out.println(rasa + " mačka " + getIme() + " mjauče.\n");
}
}
```

```
package rs.edu.asss.uup;

public class Ptica extends Životinja {

    private String vrsta;

    public Ptica(String ime, String vrsta) {
        super(ime);
        this.vrsta = vrsta;
    }

    // kreiramo set metodu kako bi smo mogli da promenimo vrstu ptice
    public void setVrsta(String vrsta) {
        this.vrsta = vrsta;
    }

    @Override
    public void oglašavanje() {
        if (vrsta.equalsIgnoreCase("pingvin") || vrsta.equalsIgnoreCase("noj")) {
            System.out.println("Ovo je posebna vrsta ptica koje ne lete.");
        } else {
            System.out.println("Ptica " + vrsta + " " + getIme() + " peva ili krešti, "
                + "zavisi kako na to svako od nas \n\"gleda\".\n");
        }
    }
}
```

```
package rs.edu.asss.uup;

public class GlavnaKlasa {

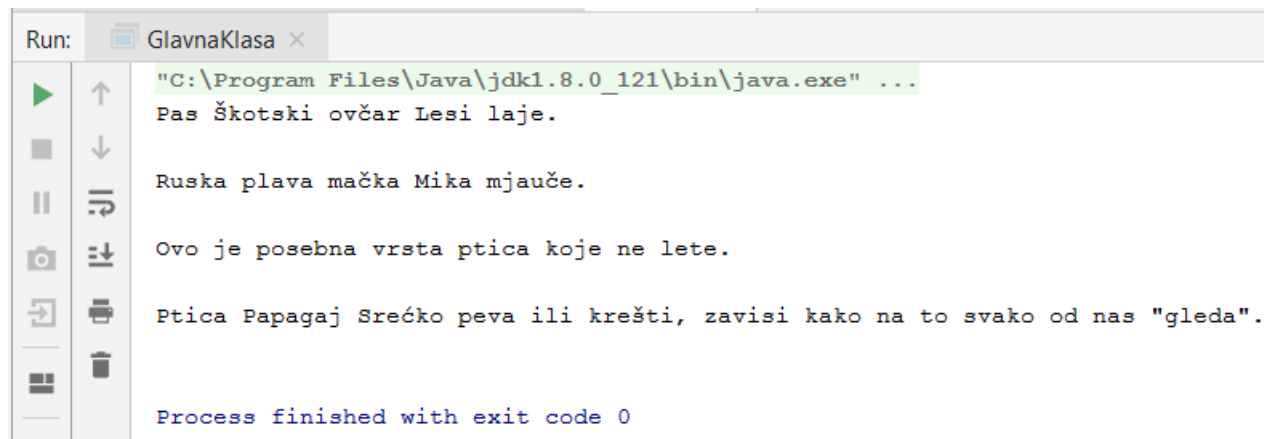
    public static void main(String[] args) {

        Pas mojPas = new Pas("Lesi", "Škotski ovčar");
        Mačka mojaMačka = new Mačka("Mika", "Ruska plava");
        Ptica mojaPtica = new Ptica("Srećko", "NOJ");

        mojPas.oglašavanje();
        mojaMačka.oglašavanje();
        mojaPtica.oglašavanje();

        System.out.println();

        mojaPtica.setVrsta("Papagaj");
        mojaPtica.oglašavanje();
    }
}
```



The screenshot shows a Java IDE's Run console window. The title bar reads "Run: GlavnaKlasa x". The console output is as follows:

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...
Pas Škotski ovčar Lesi laje.

Ruska plava mačka Mika mjauče.

Ovo je posebna vrsta ptica koje ne lete.

Ptica Papagaj Srećko peva ili krešti, zavisi kako na to svako od nas "gleda".

Process finished with exit code 0
```

On the left side of the console, there is a vertical toolbar with icons for running (a green play button), stepping through code (up and down arrows), debugging (a camera icon), and other IDE functions.