```
        // Mnoze se i ispisuje se proizvod
        System.out.println("Proizvod je: " + a.pomnozi(b));

        // Unosimo n i racunamo n!
        System.out.print("Unesite broj ciji faktorijel zelite: ");
        int n = sc.nextInt();

System.out.println(n + "! staticka verzija: "
                        + Broj.faktorijel(n));
        System.out.println(n + "! rekurzivna verz.: "
                            + (new Broj(n)).faktorijel());

        // Unosimo n i racunamo n-ti Fibonacijev broj na oba nacina
        System.out.print("Unesite indeks Fibonacijevog niza: ");
        n = sc.nextInt();

        long pocetak = System.nanoTime();
        Broj fibR = Broj.fibonaciR(n);
        long kraj = System.nanoTime();
        System.out.println("Rekurzija : " + n + ". Fibonacijev broj " + fibR
                                          + "\t vreme: " + (kraj - pocetak) + "ns"
                           );

        pocetak = System.nanoTime();
        Broj fib = Broj.fibonaci(n);
        kraj = System.nanoTime();
        System.out.println("Dinamicko programiranje: " + n + ". Fibonacijev broj: "
                                 + fib + "\t vreme: " + (kraj - pocetak) + "ns"
                         );



    }

}
```

12. *Животиње – наслеђивање.* Написати дефиницију класе *Zivotinja*. Сваку животињу карактерише њена животињска врста. Из класе *Zivotinja* извести класу *Pas*. За пса је познато његово име и раса. Ако се приликом креирања пса не наведе његова раса, поставити је на "бернардинац". Написати тест-класу.

Објашњење:

Пример илуструје важан концепт објектно оријентисаног програмирања – наслеђивање.

Наслеђивање је поступак којим се из постојећих изводе нове класе. Постојећа класа назива се базном или суперкласом, док се нова назива изведеном класом. Између базне и изведене класе постоји специфичан однос. Објекат изведене класе је специјализација објекта базне. У Јави је могуће само једноструко наслеђивање, што значи да свака изведена класа има тачно једну директну суперкласу. У Јавиној библиотеци постоји једна универзална суперкласа, *Object*, која је директна или индиректна суперкласа свих осталих класа Јавине библиотеке, али и сваке од класа коју сами напишемо. Даље, објекат изведене класе у себи увек садржи читав подобјекат базне класе. Међутим, нису сви чланови тог подобјекта базне класе директно доступни објекту изведене класе, у смислу да унутар изведене класе не можемо приступати свим члановима базне класе просто навођењем њиховог имена. Чланови базне класе који јесу директно доступни објекту изведене класе називају се наслеђеним члановима базне класе. Који чланови се наслеђују? *public* и *protected* чланови се наслеђују готово увек. Изузетак представљају једино конструктори базне класе који се никада не наслеђују. Чланови са пакетним правом приступа наслеђују се у изведеним класама које се налазе у истом пакету у коме је и базна класа, док се

иначе не наслеђују. Чланови који су у базној класи декларисани са *private* приступним атрибутом никада се не наслеђују.

Потребно је посветити посебну пажњу писању конструктора изведене класе. Наиме, неопходно је извршити иницијализацију инстанцних променљивих подобјекта базне класе што се чини позивом конструктора базне класе. Тај позив, ако је присутан, мора бити прва линија у телу конструктора изведене класе, при чему се користи кључна реч *super* као име метода. Уколико се позив конструктора базне класе изостави у конструктору изведене класе, компајлер ће имплицитно уметнути позив подразумеваног конструктора базне класе (*super()*), што може довести до грешке при компајлирању у случају да у базној класи не постоји подразумевани конструктор. Може се десити да компајлер за неку класу имплицитно генерише подразумевани конструктор са празним телом. То се дешава само ако програмер није написао ниједан конструктор у тој класи.

Допуштено је да постоји наслеђени члан базне класе који има исто име као и неки члан изведене класе. У том случају, просто навођењем имена члана обраћамо се члану изведене класе, док, да бисмо се обратили истоименом члану базне класе, морамо га квалификовати кључном речју *super*, што ћемо видети на примеру метода *toString()*.

Класа *Zivotinja* је базна. Има једну инстанцну променљиву, *vrsta*, типа *String*. Имплементирани су подразумевани и конструктор са једним аргументом. Такође, имплементиран је и метод *toString()* који враћа *String*-репрезентацију објекта ове класе.

Класа *Pas* је изведена из класе *Zivotinja* (кључна реч *extends* у првом реду дефиниције класе *Pas*). Инстанцне променљиве класе су *ime* и *rasa*, типа *String*. У телима конструктора се, као прва линија, појављује позив конструктора базне класе који иницијализује инстанцну променљиву *vrsta* класе *Zivotinja* вредношћу "пас". Типично, у методу *toString()* изведене класе најпре се позива истоимени наслеђени метод базне класе, при чему се мора користити кључна реч *super*, а затим се надовежу и преостале информације које карактеришу текући објекат изведене класе.

У тест-класи креирамо два објекта класе *Pas*, користећи обе варијанте конструктора, а затим исписујемо њихове *String*-репрезентације. Том приликом имплицитно се позива метод *toString()* класе *Pas*.

Решење:

Zivotinja.java

```
package nasledjivanje;

public class Zivotinja
{
   private String vrsta;

   public Zivotinja()
   {
   }

   public Zivotinja(String vrsta)
   {
      this.vrsta = new String(vrsta);
   }

   public String toString()
   {
      return "Ovo je " + vrsta;
```

```
   }

}
```

Pas.java

```
package nasledjivanje;

public class Pas extends Zivotinja
{
   private String ime;
   private String rasa;

   public Pas(String ime)
   {
      super("pas");
      this.ime = ime;
      rasa = "bernardinac";
   }

   public Pas(String ime, String rasa)
   {
      super("pas");
      this.ime = ime;
      this.rasa = rasa;

   }

   public String toString()
   {
      return super.toString() + " " + ime + ", " + rasa + ".";
   }

}
```

TestNasledjivanje.java

```
package nasledjivanje;

public class TestNasledjivanje
{

   public static void main(String[] args)
   {
      Pas pas = new Pas("Mange", "nemacki ovcar");
      Pas poznatPas = new Pas("Betoven");

      System.out.println(pas);
      System.out.println(poznatPas);
   }

}
```

13. *Животиње – полиморфизам*. Написати апстрактну класу *Zivotinja*, при чему сваку животињу карактерише њена врста. Класа треба да садржи метод за оглашавање животиње и метод за оглашавање животиње у љутини, као и метод који описује начин кретања животиње. Дефинисати класе *Pas, Riba* и *Pingvin* које наслеђују класу *Zivotinja* и класу *ZlatniRetriver* која наслеђује класу *Pas*. Све конкретне врсте животиња карактеришу се својим именом, за пса је позната и његова раса, а за рибу подврста. У тест-класи направити низ животиња који садржи по један примерак сваке животињске врсте, а затим 4 пута случајно изабрати елемент низа, исписати податке о изабраној животињи, као и начине њеног кретања и оглашавања.

Објашњење:

Полиморфизам означава могућност да се један исти позив метода понаша различито у зависности од типа објекта над којим се метод примењује.
Полиморфизам ради са објектима изведене класе.
Могуће је да се референца на објекат изведене класе чува у промењивој типа базне класе (директне или индиректне). Ово је обавезно да би имали могућност полиморфизма.
Који ће метод бити позван, зависи од *типа објекта* над којим се метод позива, а не од *типа промењиве* која садржи референцу на објекат.
За полиморфизам је неопходно следеће:

- Позив метода над објектом изведене класе врши се преко промењиве базне класе
- Метод који се позива мора бити декларисан у базној класи
- Метод који се позива мора бити декларисан у изведеној класи
- Потпис метода у базној и изведеној класи мора бити исти
- Повратни тип метода у изведеној класи мора бити *исти* као и у базној класи или типа који је његова *потклаcа*

Апстрактна класа је класа која има један или више метода који су декларисани, а нису дефинисани. Називамо их апстрактним методима.
Апстрактни метод не може бити *private*, јер *private* метод не може бити наслеђен, а самим тим ни предефинисан у потклаcи.
Не може се инстанцирати објекат апстрактне класе, али је могуће декларисати промењиву типа апстрактне класе.

```
Zivotinja ljubimac = null;
```

Ова промењива се касније може користити да сачува референцу на конкретан објекат изведене класе.
Уколико нису сви апстрактни методи базне класе дефинисани у изведеној класи, она ће такође бити апстрактна класа.
Ако је класа апстрактна, мора се користити кључна реч *abstract* приликом њене дефиниције.

U apstraktnoj klasi *Zivotinja* definisane su statičke konstante HODA, SKACE, PLIVA, GNJURI, koje opisuju mogуće načine kretanja raznih vrsta životinja. Binarna repрezentacija svake od njih sastoji se od jedne jedinice i nula, pri čemu su sve repрezentacije međusobno različite. Na ovaj način se omogućava njihovo jednostavno kombinovanje. Takođe, iz date kombinacije jednostavno se zaključuje koji načini kretanja čine kombinaciju.
Instancna članica ove klase predstavlja vrstu životinje.
Klasa sadrži sledeće metode:

- Konstruktor sa datom vrstom
- Metod *toString()* koji ispisuje vrstu
- Apstraktni metodi za oglašavanje i kretanje
- Statički metod *ispisiKratanje()*. Koji sve načini kretanja čine datu kombinaciju kretanja, zaključuje se primenom bitske konjunkcije na kombinaciju i konkretan način kretanja.

Klase *Pas*, *Riba* i *Pingvin* nasleđuju klasu *Zivotinja* i imaju još instancne promenljive koje su za njih karakteristične.
Definisani su apstraktni metodi za oglašavanje i kretanje, nasleđeni iz bazne klase za svaku od vrsta životinja.
Način kretanja se definiše kao kombinacija konstanti primenom bitske disjunkcije.

Rešenje:

Zivotinja.java

```
package zivotinjePolimorfizam;
```

```
public abstract class Zivotinja
{
   public static final int HODA = 1 << 1;
   public static final int SKACE = 1 << 2;
   public static final int PLIVA = 1 << 3;
   public static final int GNJURI = 1 << 4;

   // Clanica je deklarisana kao private, pa se ne nasledjuje
   private String vrsta;

   public Zivotinja(String vrsta)
   {
      this.vrsta = vrsta;
   }

   public String toString()
   {
      return "Ovo je " + vrsta;
   }

   // Metodi koji ce biti predefinisan u izvedenim klasama
   public abstract void oglasiSe();

   public abstract void oglasiSeULjutini();

   public abstract int kretanje();

   public static void ispisiKretanja(int naciniKretanja)
   {
      if((naciniKretanja & HODA) == HODA)
         System.out.print("hoda ");
      if((naciniKretanja & SKACE) == SKACE)
         System.out.print("skace ");
      if((naciniKretanja & PLIVA) == PLIVA)
         System.out.print("pliva ");
      if((naciniKretanja & GNJURI) == GNJURI)
         System.out.print("gnjura ");
      System.out.println();
   }

}
```

Pas.java

```
package zivotinjePolimorfizam;

public class Pas extends Zivotinja
{
   private String ime;
   private String rasa;

   public Pas(String ime)
   {
      super("Pas");
      this.ime = ime;
      rasa = "Nepoznata";
   }

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

   public String toString()
```

```
   {
      return super.toString() + "\nZove se " + ime + ", a rasa je " + rasa;
   }

   public void oglasiSe()
   {
      System.out.println("AV AV");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("AV AV AV AV AV");
   }

   public int kretanje()
   {
      return HODA | SKACE;
   }
}
```

Riba.java

```
package zivotinjePolimorfizam;

public class Riba extends Zivotinja
{
   private String ime;
   private String podvrsta;

   public Riba(String ime)
   {
      super("Riba");
      this.ime = ime;
      podvrsta = "Nepoznata";
   }

   public Riba(String ime, String podvrsta)
   {
      super("Riba");
      this.ime = ime;
      this.podvrsta = podvrsta;

   }

   public String toString()
   {
      return super.toString() + "\nZove se " + ime + ", a podvrsta je " +
              podvrsta;
   }

   public void oglasiSe()
   {
      System.out.println("Riba se ne oglasava.");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("Riba se ne oglasava u ljutini");
   }

   public int kretanje()
   {
      return PLIVA;
   }
}
```

Pingvin.java

```
package zivotinjePolimorfizam;

public class Pingvin extends Zivotinja
{
   private String ime;

   public Pingvin(String ime)
   {
      super("Pingvin");
      this.ime = ime;
   }

   public Pingvin(String ime, String podvrsta)
   {
      super("Pingvin");
      this.ime = ime;
   }

   public String toString()
   {
      return super.toString() + "\nZove se " + ime;
   }

   public int kretanje()
   {
      return HODA | PLIVA | GNJURI;
   }

   public void oglasiSe()
   {
      System.out.println("Specificno oglasavanje.");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("Specificno oglasavanje u ljutini.");
   }

}
```

ZlatniRetriver.java

```
package zivotinjePolimorfizam;

public class ZlatniRetriver extends Pas
{
   public ZlatniRetriver(String imePsa)
   {
      super(imePsa, "ZlatniRetriver");
   }
}
```

TestPolimorfizam.java

```
package zivotinjePolimorfizam;

import java.util.Random;

public class TestPolimorfizam
{
   public static void main(String[] args)
   {
```

```
      Zivotinja zivotinje[] = {
           new Pas("Ajk", "dalmatinac"),
           new Riba("Nemo", "riba klovn"),
           new Pingvin("Tux"),
           new ZlatniRetriver("Zak")
      };

      Random random = new Random();

      for(int i = 0; i < 4; i++)
      {
         Zivotinja zivotinja =
                 zivotinje[random.nextInt(zivotinje.length)];
         System.out.print((i+1) + ". " + zivotinja + ": ");
         Zivotinja.ispisiKretanja(zivotinja.kretanje());
         zivotinja.oglasiSe();
         zivotinja.oglasiSeULjutini();
         System.out.println();
      }
   }
}
```

14. *Животиње – интерфејси.* Написати Јава-програм за рад са животињама. Од животињских врста треба описати пса, мачку, рибу, жабу, кокошку, делфина и пингвина. Свака животиња има своје име; за пса, мачку и кокошку позната је и њихова раса, а за рибу врста. Имплементирати интерфејсе *Oglasavanje* (описује оглашавање животиње, као и оглашавање животиње у љутини) и *Kretanje* (описује начин кретања животиње). Интерфејс *Kretanje* имплементирати за све наведене животињске врсте, а интерфејс *Oglasavanje* за све осим рибе, делфина и пингвина. Написати потом и тест-класу. У тест-класи направити низ животиња који садржи по један примерак сваке животињске врсте, а затим 5 пута случајно изабрати елемент низа и исписати податке о изабраној животињи, као и начине њеног кретања. Уколико животиња има могућност оглашавања, исписати најпре њено уобичајено, а потом и оглашавање у љутини.

Објашњење:

Интерфејс је колекција повезаних константи и/или апстрактних метода и у већини случајева садржи само методе. Интерфејс не дефинише како метод ради. Он само дефинише његов облик – име, параметре, повратни тип, тако да су по дефиницији методи у интерфејсу апстрактни.

Коришћење интерфејса подразумева постојање бар једне класе која имплементира тај интерфејс. Када класа имплементира интерфејс, директно су јој доступни сви чланови интерфејса, управо као да су наслеђени од базне класе. Да би било могуће креирање конкретних објеката класе која имплементира интерфејс, неопходно је да се у дефиницији класе налази имплементација сваког од метода декларисаних у интерфејсу. Иначе, класа од интерфејса "наслеђује" бар један апстрактан метод, те је и сама апстрактна, што је неопходно навести у њеној дефиницији. Ако се то не учини, јавља се грешка при компајлирању.

Методи у интерфејсу су увек *public* и *abstract*, па не морамо то експлицитно да наводимо. Не могу бити статички. Константе у интерфејсу су увек *public*, *static* и *final*, па ни за њих нема потребе наводити експлицитно ове кључне речи.

Интерфејс који декларише методе дефинише стандардни скуп операција. Разне класе могу додати такав стандардни интерфејс имплементирајући га. Тако објекти разних класа могу да деле заједнички скуп операција. Наравно, дата операција у једној класи може бити имплементирана посве различито од начина на који је имплементирана у другој класи. Али, начин на који се позива операција је исти за објекте свих класа које имплементирају интерфејс.

Најважнија употреба интерфејса је: коришћење полиморфизма кроз скуп класа које имплементирају исти интерфејс.

Није могуће креирати објекте типа интерфејса, али могуће је декларисати променљиву типа интерфејса. Ту променљиву можемо користити за смештање референце на објекат произвољне класе која имплементира интерфејс.
Класа може имплементирати и више од једног интерфејса.

За сваки могући начин кретања (ходање, скакање, пливање, гњурање, летење) животињских врста које су нам од интереса у интерфејсу *Kretanje* дефинисана је по једна константа типа *int*. Константе су изабране тако да су све међусобно различите и у битовској репрезентацији сваке од њих тачно један бит је јединица, док су сви преостали нуле. Овакав избор је погодан јер се коришћењем битовског оператора | (или) ове константе једноставно комбинују, а резултат је поново вредност типа *int*. Такође, ако имамо неку комбинацију константи добијену коришћењем оператора |, применом оператора & (и) могуће је испитати да ли је одређена константа укључена у ту комбинацију.

Метод *kreciSe()* из интерфејса *Kretanje* враћа комбинацију свих могућих начина кретања одређене животињске врсте као вредност типа *int*, па је његова имплементација у класама прилично једноставна.

У тест-класи имамо помоћни метод *ispisiKretanja()* који, на основу дате комбинације, исписује све начине кретања који су укључени у ту комбинацију.

Методе из интерфејса ћемо полиморфно позивати. Из тог разлога неопходно је референце на објекте класа чувати у променљивама типа интерфејса. Све класе имплементирају интерфејс *Kretanje*, па је низовска променљива *zivotinje* декларисана тако да буде типа овог интерфејса.

Након креирања низа, у *for*-петљи се 5 пута случајно бира његов индекс коришћењем објекта *random* класе *Random* и референца на елемент са тим индексом памти у променљивој *zivotinja* типа *Kretanje*. Следи полиморфно позивање метода *kreciSe()*.
Да бисмо полиморфно позвали методе интерфејса *Oglasavanje*, неопходно је да се ради о објекту класе која имплементира тај интерфејс (што се проверава оператором *instanceof*), а ако то јесте случај, пошто имамо референцу на објекат у променљивој типа *Kretanje*, неопходно је пре позива метода извршити експлицитно кастовање референце у тип *Oglasavanje* чији метод желимо да позовемо.

Решење:

Pas.java

```
package interfejsi;

public class Pas implements Oglasavanje, Kretanje
{
   private String ime;
   private String rasa;

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

   public void oglasiSe()
   {
      System.out.println("Av, av.");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("Avavavauuuu!");
   }
```

```
   /*
 * metod vraca kombinaciju svih mogucih nacina kretanja psa
    * kao vrednost tipa int
    */
   public int kreciSe()
   {
      return HODA | SKACE;
   }

   public String toString()
   {
      return "Pas " + ime + ", " + rasa;
   }

}
```

Macka.java

```
package interfejsi;

public class Macka implements Oglasavanje, Kretanje
{
   private String ime;
   private String rasa;

   public Macka(String ime, String rasa)
{
      this.ime = ime;
      this.rasa = rasa;
   }

   public void oglasiSe()
   {
      System.out.println("Mijau.");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("Shhuuu!");
   }

   /*
 * metod vraca kombinaciju svih mogucih nacina kretanja macke
    * kao vrednost tipa int
    */
   public int kreciSe()
   {
      return HODA | SKACE;
   }

   public String toString()
   {
      return "Macka " + ime + ", " + rasa;
   }


}
```

Riba.java

```
package interfejsi;

public class Riba implements Kretanje
{
   private String ime;
   private String vrsta;
```

```
   public Riba(String ime, String vrsta)
   {
      this.ime = ime;
      this.vrsta = vrsta;
   }

   /*
 * metod vraca nacin kretanja ribe kao vrednost
    * tipa int
    */
   public int kreciSe()
   {
      return PLIVA;
   }

   public String toString()
   {
      return "Riba " + ime + ", " + vrsta;
   }

}
```

Zaba.java

```
package interfejsi;

public class Zaba implements Oglasavanje, Kretanje
{
   private String ime;

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

   public void oglasiSe()
   {
      System.out.println("Kre, kre.");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("Kre-kre, kre-kre!");
   }

   /*
 * metod vraca kombinaciju svih mogucih nacina kretanja zabe
    * kao vrednost tipa int
    */
   public int kreciSe()
   {
      return SKACE | PLIVA;
   }

   public String toString()
   {
      return "Zaba " + ime;
   }

}
```

Kokoska.java

```
package interfejsi;

public class Kokoska implements Oglasavanje, Kretanje
```

```
{
   private String ime;
   private String rasa;

   public Kokoska(String ime, String rasa)
   {
      this.ime = ime;
      this.rasa = rasa;
   }

   public void oglasiSe()
   {
      System.out.println("Kokoda.");
   }

   public void oglasiSeULjutini()
   {
      System.out.println("Kokokokokokodaaaa!");
   }

   /*
 * metod vraca kombinaciju svih mogucih nacina kretanja kokoske
    * kao vrednost tipa int
    */
   public int kreciSe()
   {
      return HODA | LETI;
   }

   public String toString()
   {
      return "Kokoska " + ime + ", " + rasa;
   }

}
```

Delfin.java

```
package interfejsi;

public class Delfin implements Kretanje
{
   private String ime;

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

   /*
    * metod vraca kombinaciju svih mogucih nacina kretanja delfina
    * kao vrednost tipa int
    */
   public int kreciSe()
   {
      return PLIVA | SKACE | GNJURA;
   }

   public String toString()
   {
      return "Delfin " + ime;
   }

}
```

Pingvin.java

```
package interfejsi;

public class Pingvin implements Kretanje
{
   private String ime;

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

   /*
 * metod vraca kombinaciju svih mogucih nacina kretanja pingvina
    * kao vrednost tipa int
    */
   public int kreciSe()
   {
      return PLIVA | HODA | GNJURA;
   }

   public String toString()
   {
      return "Pingvin " + ime;
   }

}
```

Kretanje.java

```
package interfejsi;

public interface Kretanje
{
   int HODA = 1 << 1;
   int SKACE = 1 << 2;
   int PLIVA = 1 << 3;
   int GNJURA = 1 << 4;
   int LETI = 1 << 5;

   int kreciSe();
}
```

Oglasavanje.java

```
package interfejsi;

public interface Oglasavanje
{
   void oglasiSe();
   void oglasiSeULjutini();
}
```

TestInterfejsi.java

```
package interfejsi;

import java.util.Random;

public class TestInterfejsi
{

   private static void ispisiKretanja(int naciniKretanja)
   {
      if ((naciniKretanja & Kretanje.HODA) == Kretanje.HODA)
         System.out.print("hoda ");
      if ((naciniKretanja & Kretanje.SKACE) == Kretanje.SKACE)
         System.out.print("skace ");
```

```
      if ((naciniKretanja & Kretanje.PLIVA) == Kretanje.PLIVA)
         System.out.print("pliva ");
      if ((naciniKretanja & Kretanje.GNJURA) == Kretanje.GNJURA)
         System.out.print("gnjura ");
      if ((naciniKretanja & Kretanje.LETI) == Kretanje.LETI)
         System.out.print("leti ");
      System.out.println();
   }

   public static void main(String[] args)
   {
      Kretanje zivotinje[] = { new Pas("Mange", "nemacki ovcar"),
          new Macka("Silvester", "persijski macak"),
          new Riba("Nemo", "riba klovn"), new Zaba("Kermit"),
          new Kokoska("Mica", "domaca kokos"), new Delfin("Joca"),
          new Pingvin("Tux") };

      Random random = new Random();

      for (int i = 0; i < 5; i++)
      {
         Kretanje zivotinja = zivotinje[random.nextInt(zivotinje.length)];
         System.out.print((i + 1) + ". " + zivotinja + ": ");
         ispisiKretanja(zivotinja.kreciSe());
         if (zivotinja instanceof Oglasavanje)
         {
            ((Oglasavanje) zivotinja).oglasiSe();
            ((Oglasavanje) zivotinja).oglasiSeULjutini();
         }

      }

   }

}
```

15. *Угњеждене класе*. Написати класу *MagicniSesir* која представља магични шешир. Магични шешир садржи променљив број зечева који су дефинисани класом *Zec*. Она је угњеждена у класи *MagicniSesir*.
Магични шешир се карактерише својим називом и низом зечева који су у њему. Оно што је заједничко за све магичне шешире јесте максималан дозвољени број зечева и дозвољена имена зечева. Обезбедити начин да се зечеви са истим именом међусобно разликују. Написати конструктор и метод за генерисање *String*-репрезентације магичног шешира, коју чине назив шешира и имена зечева који су у њему.
Сваки зец карактерише се својим именом, које је једно од дозвољених имена и мора бити генерисано као јединствено. Обезбедити и метод за генерисање *String*-репрезентације зеца која садржи његово име.

Објашњење:

Угњеждена класа је класа чија се дефиниција налази унутар друге класе. Угњеждена класа је чланица спољашње класе и додељују јој се права приступа као и било којој другој чланици.

```
public class Spoljasnja {
   public class Unutrasnja {
}
}
```