Владимир Филиповић
Биљана Стојановић
Марија Милановић
Сташа Вујичић Станковић

Задаци из Објектно-орјентисаног програмирања (Скрипта)

Београд, 2012.

1. *Аритметичка средина*. Написати апликацију у којој се са стандардног улаза најпре учитава број елемената низа реалних бројева, а потом и сами елементи низа и рачуна и исписује збир и аритметичка средина учитаних бројева.

Објашњење:

Објекат класе *java.util.Scanner* користи се као парсер стандардног улаза.
Класом *java.util.Scanner* имплементира се једноставни парсер који парсира примитивне типове и стрингове користећи регуларне изразе.
Улаз може бити фајл или ток, укључујући и стандардни улазни ток *System.in*.
Улаз се дели на токене користећи као делимитер размак (‘ ‘).
Добијени токени се могу конвертовати у вредности различитих типова одговарајућим *next...* методом за сваки тип.

У решењу се може уочити у великој мери стандардна процедура рада са низовима која обухвата следеће кораке:

- *Учитавање димензије низа* (уколико није претходно задата)
  ```
  int brojEl = skener.nextInt();
  ```
  Методом *nextInt()* објекта за парсирање учитава се цео број са улаза који представља димензију низа.
- *Креирање низа*, тј. алоцирање потребног меморијског простора за елементе низа
  ```
  int[] nizBrojeva = new int[brojEl];
  ```
  Лева страна наредбе доделе је само декларација низа целих бројева. Да би низ био креиран, тј. да би се алоцирао потребан меморијски простор за елементе низа, на десној страни наредбе доделе неопходна је кључна реч *new* праћена типом и бројем елемената низа између [] заграда.
- *Учитавање елемената низа*
  У петљи се редом учитавају елементи низа. У овом случају се за учитавање користи метод *nextInt()* објекта за парсирање, пошто се ради о низу целих бројева.
- *Обрада*
  Део обраде може се извршити и приликом учитавања елемената. Овде је сабирање елемената низа извршено у фази учитавања. Користи се помоћна променљива *zbir* која иницијално има вредност 0. Вредност сваког новоучитаног елемента додаје се на текућу вредност ове променљиве.
  Друга фаза обраде је рачунање аритметичке средине као количника добијене суме елемената низа и њиховог броја.

Решење:

AritmetickaSredina.java

```
package aritmetickaSredina;

import java.util.Scanner;

public class AritmetickaSredina
{
   public static void main(String[] args)
   {
     // Parser standardnog ulaza
     Scanner skener = new Scanner(System.in);
     // Ucitavanje broja elemenata niza i kreiranje niza
     System.out.println("Unesite broj elemenata niza:");
     // Broj elemenata niza
     int brojEl = skener.nextInt();
     // Kreiranje niza date velicine
     int[] nizBrojeva = new int[brojEl];
```

```
     // Promenljiva u kojoj se racuna zbir elemenata
     int zbir = 0;

     // Ucitavaju se elementi niza i istovremeno se racuna njihov zbir
     for(int i = 0; i < brojEl; i++)
     {
        System.out.print((i + 1) + ". broj: ");
        nizBrojeva[i] = skener.nextInt();
        zbir += nizBrojeva[i];
     }

     // Ispis zbira elemenata niza
     System.out.println("Zbir brojeva je: " + zbir);

     // Racunanje i ispis aritmeticke sredine elemenata niza
     double aritmSredina = (double) zbir / brojEl;
     System.out.println("Aritmeticka sredina je: " + aritmSredina);
   }
}
```

2. *Бројање речи.* Одредити број појављивања речи "све" и "је" у задатом тексту.

Најзначајнији коришћени методи:

```
int indexOf (String str);
int indexOf (String str, int fromIndex);
int lastIndexOf (String str);
int lastIndexOf (String str, int fromIndex);
```

класе *String* (из пакета *java.lang*).

Ако је стринг *str* подстринг текућег стринга, метод *indexOf(str)* враћа индекс прве појаве стринга *str* у текућем стрингу. Иначе, метод враћа вредност -1.

Метод *indexOf(str, fromIndex)* враћа индекс прве појаве стринга *str* у текућем стрингу, али почев од позиције *fromIndex*, ако таква појава постоји. Иначе, метод враћа вредност -1.

Ако је стринг *str* подстринг текућег стринга, метод *lastIndexOf(str)* враћа индекс последње појаве стринга *str* у текућем стрингу. Иначе, метод враћа вредност -1.

Метод *lastIndexOf(str, fromIndex)* враћа индекс последње појаве стринга *str* у текућем стрингу, али почев од позиције *fromIndex*, ако таква појава постоји. Иначе, метод враћа вредност -1.

Примери:

```
String tekst = "abrakadabra";
String str = "bra";
int prva = tekst.indexOf(str);    // prva = 1 (indeksi se broje od 0)
int neka = tekst.indexOf(str, 6);  // neka = 8
int nema = tekst.indexOf(str, 10); // nema = -1
int poslednja = tekst.lastIndexOf(str); // poslednja = 8
int neka2 = tekst.lastIndexOf(str, 5);  // neka2 = 1
int nema2 = tekst.lastIndexOf(str, 0);   // nema2 = -1
```

Објашњење:

Појављивања речи "све" тражићемо од почетка текста ка његовом крају користећи метод *indexOf()*, док ћемо појављивања речи "је" тражити у супротном смеру, тј. од краја ка почетку текста коришћењем метода *lastIndexOf()*.

За сваку од две тражене речи уводи се по један бројач (*sveBr*, *jeBr*) који се иницијализује нулом, а затим инкрементира приликом проналажења сваке појаве одговарајуће речи у датом тексту.

Реч "све" тражимо од почетка текста, па за добијање индекса прве појаве те речи позивамо метод *indexOf()* који прима један аргумент (*String* који тражимо). У случају да је тражена реч пронађена, повратна вредност метода је *≥ 0* (у случају неуспеха метод враћа вредност -1), па се улази у *while*-петљу којом се обрађују и преостале појаве речи "све", ако их има у тексту. Како се услов уласка у тело петље (*index >= 0*) сваки пут проверава након позива метода *indexOf()* било са једним, било са два аргумента, улазак у тело петље значи да је управо пронађена једна појава тражене речи. Зато се одговарајући бројач увећава, а затим се врши припрема за тражење следеће појаве те речи у тексту. Реч коју тражимо, "све", је таква да, ако је пронађена њена појава на некој позицији, *index*, следећа појава сигурно се не може наћи пре позиције непосредно иза краја ове појаве (што нпр. не би био случај да тражимо реч "ааа", где би наредна појава евентуално могла да почне већ од позиције *index+1*). Индекс премештамо на ту позицију (*indeks += sveStr.length()*) и настављамо потрагу одатле. По изласку из петље, у променљивој *sveBr* налази се број појављивања речи "све" у датом тексту.

Појављивања речи "је" тражимо крећући се унатраг кроз текст, па индекс последње појаве покушавамо добити позивом метода *lastIndexOf()* са једним аргументом. Логика је иста као и код тражења унапред. Једина је разлика у припреми индекса за следећи позив метода *lastIndexOf()*. Наиме, ако је претходни позив метода *lastIndexOf()* за реч "је" вратио нпр. индекс *index*, и ако постоји још нека појава ове речи на позицији мањој од *index*, прва позиција од које има смисла тражити претходну појаву речи "је" добија се када се од позиције *index* одмакнемо за дужину тражене речи (*indeks –= jeStr.length()*), јер та реч је таква да није могуће да се две њене узастопне појаве преклопе.

Решење:

BrojanjeReci.java

```
package brojanjeReci;

public class BrojanjeReci
{

   public static void main(String[] args)
   {
      // Tekst koji se analizira
      String text = "U svetu postoji jedno carstvo "
                  + "u njemu caruje drugarstvo "
+ "u njemu je sve lepo "
                    + "u njemu je sve nezno "
+ "u njemu se sve raduje.";

      int sveBr = 0;  // brojac pojava reci "sve"
      int jeBr = 0;      // brojac pojava reci "je"
      int indeks = -1; // pozicija u tekstu koji se analizira
      String sveStr = "sve";
      String jeStr = "je";

      // Trazi se pojavljivanje reci "sve"
      indeks = text.indexOf(sveStr); // Nalazi se prva pojava reci "sve"
      while (indeks >= 0)
      {
         ++sveBr;
         indeks += sveStr.length();   // Postavljanje na poziciju iza
                                      // pojave tekuceg "sve"
```

```
         indeks = text.indexOf(sveStr, indeks);
      }

      // Trazi se unazad pojavljivanje reci "je"
      indeks = text.lastIndexOf(jeStr);   // Nalazi se poslednja pojava
                                          // reci "je"
      while (indeks >= 0)
      {
         ++jeBr;
         indeks -= jeStr.length();  // Postavljanje na poziciju ispred
                                    // pojave tekuceg "je"
         indeks = text.lastIndexOf(jeStr, indeks);
      }
      System.out.println("Zadati tekst sadrzi " + sveBr +
                         " pojave reci \"sve\"\n" +
                         "i " + jeBr + " pojava reci \"je\".");

   }

}
```

3. *Факторијел – рекурзија.* Написати рекурзивну функцију *faktorijel()* за рачунање факторијела датог ненегативног целог броја. Написати функцију за тестирање функције *faktorijel()*.

Објашњење:

Факторијел датог ненегативног целог броја n дефинише се рекурзивно са:

n! = 1, n = 0 (излаз из рекурзије)
n! = n * (n - 1)!, n > 0 (рекурзивни корак)

Дефиниција проблема је рекурзивна ако се решавање проблема своди на решавање истог проблема, али мање димензије. У нашем случају, n! се рачуна преко (n - 1)!. Битно је и да постоји један или већи број случајева који се (релативно) лако решавају и који служе као излазак из рекурзије.

Када за неки проблем постоји рекурзивна дефиниција, најлакше је написати рекурзивну функцију за његово решавање. Функција је рекурзивна ако саму себе позива.

Рекурзивна функција за рачунање факторијела прати рекурзивну дефиницију овог проблема. Функцију смо декларисали као статичку да бисмо могли да је позивамо без прављења конкретних објеката класе у чијој дефиницији се налази дефиниција наше рекурзивне функције.

Решење:

Faktorijel.java

```
package faktorijel;

import java.util.Scanner;

public class Faktorijel
{

   public static long faktorijel(int n)
   {
      // izlaz iz rekurzije
      if (n == 0)
         return 1;

      // rekurzivni korak
```

```
      return n * faktorijel(n - 1);
   }

   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);
      int n;
      System.out.println("Unesite broj ciji faktorijel trazite " +
"(negativan za kraj)");
      while ((n = sc.nextInt()) >= 0)
      {
         System.out.println(n + "! = " + faktorijel(n));
         System.out.println("Unesite broj ciji faktorijel trazite " +
"(negativan za kraj)");
      }
      System.out.println("Kraj!");

   }

}
```

4. *Фибоначијеви бројеви*. Написати рекурзивни и итеративни метод који за дати природан број *n* рачуна *n*-ти Фибоначијев број. Апликација треба да, за природан број *n* учитан са стандардног улаза, одреди и испише *n*-ти Фибоначијев број.

Објашњење:

Проблем је рекурзивне природе уколико се поступак његовог решавања може свести на поступак решавања истог проблема, али мање димензије. Обично се успоставља функционална веза која омогућује свођење на проблем мање димензије, што представља тзв. *рекурзивни корак*. Неопходно је обезбедити окончање процеса рекурзивног израчунавања, тј. обезбедити тзв. излаз из рекурзије, што се постиже једним или већим бројем случајева који се најчешће једноставно решавају.

Функционална веза у рекурзивној дефиницији проблема омогућује дефинисање рекурзивне функције за решавање проблема.
Рекурзивна функција позива саму себе за решавање проблема мање димензије.

Рекурентна формула за рачунање *n*-тог члана Фибоначијевог низа:

$F(0) = 1, F(1) = 1$ (1)

$F(n) = F(n-1) + F(n-2)$ (2)

Искази *(1)* представљају излаз из рекурзије, док исказ *(2)* представља рекурзивни корак.

Дефиниција и рекурзивног и итеративног метода за рачунање *n*-тог члана Фибоначијевог низа произилази из горње рекурентне формуле. Разликује се начин на који се формула примењује.

Рекурзивни метод *fibonaciRekurzivno()* позива самог себе за рачунање Фибоначијевих бројева мањих од *n*. Проблем са овим начином је што се не памте резултати рекурзивних позива, те се проблем исте димензије израчунава више пута.
Пример: позив *fibonaciRekurzivno(4)* имплицира позиве *fibonaciRekurzivno(3)* и *fibonaciRekurzivno(2)*. Они даље условљавају позиве *fibonaciRekurzivno(2)* и *fibonaciRekurzivno(1)*, односно *fibonaciRekurzivno(1)* и *fibonaciRekurzivno(0)*. Позив *fibonaciRekurzivno(2)* даље условљава позиве *fibonaciRekurzivno(1)* и *fibonaciRekurzivno(0)*.
Може се закључити да у рекурзивном поступку има доста поновљених израчунавања, чији се број повећава са повећањем димензије проблема.

Итеративни метод *fibonaciIterativno()* користи идеју *динамичког програмирања* која такође своди решавање проблема одговарајуће димензије на проблеме мање димензије, али се резултати тих израчунавања памте, тако да нема вишеструких израчунавања.
Израчунате вредности се памте у низу. Израчунавање почиње од првог елемента Фибоначијевог низа. Приликом рачунања произвољног члана Фибоначијевог низа, сви претходни чланови су већ срачунати и њихове вредности су сачуване у низу на одговарајућим позицијама које претходе позицији елемента који се рачуна. Потребна су само два претходна елемента за његово рачунање према рекурентној формули.

Дефиниције оба метода су статичке, што омогућава позив сваког метода без претходног креирања објекта класе *Fibonaci* која садржи те дефиниције.

У методу *main()* рачуна се *n*-ти члан Фибоначијевог низа за дати ненегативни број *n* и рекурзивним и итеративним поступком, исписују се резултати, као и време потребно за рачуње у оба случаја. Рачуна се време у наносекундама. Код већих вредности броја *n*, може се уочити драстична разлика у времену израчунавања ова два приступа решавању.

Решење:

Fibonaci.java

```
package fibonaci;

import java.util.Scanner;

public class Fibonaci
{
   // Rekurzivna definicija metoda
   public static int fibonaciRekurzivno(int n)
   {
      // izlaz iz rekurzije
      if(n == 0 || n == 1)
        return 1;
      // rekurzivni korak
      else
      {
        int n1 = fibonaciRekurzivno(n - 1);
        int n2 = fibonaciRekurzivno(n - 2);
        return n1 + n2;
      }
   }

   // Iterativna definicija metoda koriscenjem
   // principa dinamickog programiranja
   public static int fibonaciIterativno(int n)
   {
      // niz za cuvanje prvih n+1 Fibonacijevih brojeva
      // jer indeksi pocinju od 0.
      int fibonaci[] = new int [n+1];

      // nulti element niza se inicijalizuje na 1
      fibonaci[0] = 1;

      // ako je n bar 1, inicijalizuje se i prvi element na 1
      if(n >= 1)
        fibonaci[1] = 1;

      // za svako i, 2 <= i <= n, primenjuje se
      // rekurentna formula
      for(int i = 2; i < n+1; i++)
        fibonaci[i] = fibonaci[i-1] + fibonaci[i-2];
```

```
      // n-ti clan Fibonacijevog niza je
      // element fibonaci[n] dobijenog niza
      return fibonaci[n];

   }

   public static void main(String[] args)
   {
      // Parser standardnog ulaza
      Scanner skener = new Scanner(System.in);
      // Broj za koji se racuna n-ti clan
      // Fibonacijevog niza
      int n;
      System.out.print("Unesite ceo nenegativan broj " +
           " - negativan za kraj: ");

      while((n = skener.nextInt()) >= 0)
      {
         // Rekurzivna varijanta
         long vremePocetak = System.nanoTime();
         int fibonaciR = Fibonaci.fibonaciRekurzivno(n);
         long vremeKraj = System.nanoTime();

         System.out.println("Rekurzivna varijanta:");
         System.out.println("Fibonaci(" + n + ") = " + fibonaciR);
         System.out.println("Vreme: " + (vremeKraj - vremePocetak) + "ns");

         vremePocetak = System.nanoTime();
         int fibonaciI = Fibonaci.fibonaciIterativno(n);
         vremeKraj = System.nanoTime();

         System.out.println("Iterativna varijanta:");
         System.out.println("Fibonaci(" + n + ") = " + fibonaciI);
         System.out.println("Vreme: " + (vremeKraj - vremePocetak) + "ns");

         System.out.print("Unesite ceo nenegativan broj " +
            " - negativan za kraj: ");
         }
   }
}
```

5. *Сфера.* Написати класу *Sfera* која описује сфере у *3D*. Од метода имплементирати: подразумевани конструктор (креира јединичну сферу са центром у координатном почетку), конструктор са три аргумента (координате центра сфере; креира јединичну сферу са центром у тачки чије су координате дате као аргументи конструктора), конструктор који креира сферу када су познати сви неопходни подаци, метод за рачунање запремине сфере, као и метод за промену полупречника сфере. Такође, обезбедити да је у сваком тренутку могуће добити број до тада креираних сфера. Написати потом и тест-класу.

Објашњење:

Од инстанцних променљивих, класа *Sfera* поседује *x*, *y* и *z* координату центра сфере, као и полупречник сфере, све типа *double*. Ове променљиве карактеришу сваку појединачну сферу и свака креирана сфера имаће свој примерак сваке од инстанцних променљивих. Такође, класа *Sfera* поседује и једну статичку целобројну променљиву, *brojac*, која одговара броју креираних сфера. Променљива је статичка, што значи да постоји чак и када није креиран ниједан конкретан објекат класе (тада има вредност 0, што је у реду) и постоји само један примерак ове променљиве у меморији. Право место за увећавање ове променљиве је у телима конструктора, јер се увек приликом креирања објекта класе врши позив одговарајућег конструктора.
Подразумевани конструктор (конструктор без аргумената) поставља вредност полупречника на 1 и увећава бројач креираних сфера. Како су инстанцне променљиве *x*, *y* и *z* нумеричког типа (реалне), аутоматски ће бити иницијализоване нулама, па то нема потребе експлицитно писати.

Конструктор који прима три аргумента (координате центра сфере) позива подразумевани конструктор (*this()* у првој линији тела конструктора), а затим поставља вредности координата центра на задате вредности.
Конструктор који прима сва 4 неопходна податка најпре позове конструктор са 3 аргумента (који направи јединичну сферу са центром у датој тачки и увећа бројач креираних сфера), а затим промени вредност полупречника на задату вредност. Када постоје инстанцна променљива класе и истоимени аргумент метода (у овом случају *poluprecnik*), инстанцној променљивој се приступа коришћењем кључне речи *this* (*this.poluprecnik*), док, ако се изостави кључна реч *this*, врши се приступ истоименом аргументу метода (*poluprecnik*).

Запремина сфере рачуна се по формули: $\frac{4}{3}R^3\pi$ (где је *R* полупречник сфере).

Као π користимо вредност константе *PI* из класе *Math* пакета *java.lang*. Ако у фајл `Sfera.java` додамо статичку *import*-декларацију овог члана класе *Math*, унутар класе *Sfera* можемо му приступати просто навођењем његовог имена (*PI*) без квалификовања именом класе (*Math.PI*). Такође, да би се извршило реално дељење, неопходно је да бар један од операнада оператора / буде реалан број. Израз 4/3 у Јави има вредност 1 типа *int* јер се врши целобројно дељење, док је резултат израза 4./3 реалан број који има вредност приближно једнаку 1.333...

За дохватање вредности статичке променљиве *brojac*, имплементиран је, такође статички, метод *getBrojac()*.

У свакој својој класи писаћемо метод са прототипом:
```
public String toString();
```
који враћа *String*-репрезентацију објекта класе у форми коју ми желимо. Овај метод се имплицитно позива у ситуацијама типа:
```
System.out.println(lopta);
```
(као да смо написали: `System.out.println(lopta.toString());`)
као и
```
lopta + ", zapremina = "  // "sabiranje" objekta i String-a
```
(као да смо написали: `lopta.toString() + ", zapremina = "`)

У тест-класи, пре креирања било које сфере, исписујемо вредност бројача (која је тада 0), а затим креирамо сферу са центром у координатном почетку полупречника 4 и исписујемо њену *String*-репрезентацију и запремину. Креирање објекта класе врши се навођењем кључне речи *new* за којом следи позив одговарајућег конструктора. Поново исписујемо вредност бројача (која је у том тренутку 1 јер је креирана једна сфера). Поступак се понавља за сферу која има центар у тачки (1,1,1), а полупречника је 12 и јединичну сферу са центром у тачки (0,1,0).

Решење:

`Sfera.java`

```
package sfera;

import static java.lang.Math.PI;

public class Sfera
{
   private double xCentar;
   private double yCentar;
```

```
   private double zCentar;
   private double poluprecnik;

   private static int brojac = 0;

   public Sfera()
   {
      poluprecnik = 1;
      brojac++;
   }

   public Sfera(double x, double y, double z)
   {
      this();
      xCentar = x;
      yCentar = y;
      zCentar = z;
   }

   public Sfera(double x, double y, double z, double poluprecnik)
   {
      this(x, y, z);
      this.poluprecnik = poluprecnik;
   }

   public double zapremina()
   {
      return 4. / 3 * poluprecnik * poluprecnik * poluprecnik * PI;
   }

   public void setPoluprecnik(double poluprecnik)
   {
      this.poluprecnik = poluprecnik;
   }

   public static int getBrojac()
   {
      return brojac;
   }

   public String toString()
   {
      return "[(" + xCentar + ", " + yCentar + ", " + zCentar +
             ") R = " + poluprecnik + "]";
   }
}
```

TestSfera.java

```
package sfera;

public class TestSfera
{

   public static void main(String[] args)
   {
      System.out.println("Broj kreiranih sfera: " + Sfera.getBrojac());

      Sfera lopta = new Sfera(0.0, 0.0, 0.0, 4.0);
      System.out.println("lopta: " + lopta + ", zapremina = " +
                         lopta.zapremina());
      System.out.println("Broj kreiranih sfera: " + Sfera.getBrojac());

      Sfera globus = new Sfera(1.0, 1.0, 1.0, 12.0);
      System.out.println("globus: " + globus + ", zapremina = " +
                         globus.zapremina());
```

```
     System.out.println("Broj kreiranih sfera: " + Sfera.getBrojac());

Sfera loptica = new Sfera(0.0, 1.0, 0.0);
     System.out.println("loptica: " + loptica + ", zapremina = " +
                        loptica.zapremina());
     System.out.println("Broj kreiranih sfera: " + Sfera.getBrojac());

   }

}
```

6. Геометрија. Апликација за рад са тачкама и дужима у 2D.
Класа *Tacka* представља тачку која се карактерише *x* и *y* координатом. Од метода садржи одговарајуће конструкторе, констуктор копије, метод за померање тачке дуж *x* и *y* осе, метод *toString(),* као и *get()* и *set()* методе.
Класа *Duz* представља дуж са датом почетном и крајњом тачком. Од метода садржи конструктор са задатом почетном и крајњом тачком као и конструктор са задатим координата почетне и крајње тачке. Потом, метод за рачунање дужине дужи, метод за рачунање координата пресечне тачке правих одређених двема дужима и метод *toString().*
Класа *TestGeometrija* тестира рад са тачкама и дужима.

Објашњење:
Класа *Tacka* има две инстанцне променљиве, *x* и *y* координату тачке у равни, типа *double*.
Садржи следеће методе:

- подразумевани конструктор који поставља тачку у координатни почетак. Тело конструктора је празно. Подразумевано се врши иницијализација инстанцних променљивих нулама јер се ради о нумеричким вредностима.
- конструктор са датим вредностима за координате тачке. Врши се иницијализација координата тим вредностима.
- конструктор копије који креира нови објекат класе *Tacka* на основу постојећег. Референца на постојећи објекат прослеђује се као аргумент конструктору. Кључна реч *final* испред назива класе наглашава да објекат неће бити модификован, већ да се само приступа вредностима његових инстанцних променљивих.
- метод *pomeri()* помера тачку дуж *x* и *y* осе. Величина помераја дуж обе осе задаје се као аргумент. Метод модификује тренутне вредности координата тачке.
- Метод *rastojanje()* рачуна растојање до задате тачке по познатој формули, при чему се операција квадратног кореновања врши позивом метода *Math.sqrt()* из пакета *java.lang*.
- *get()* и *set()* методе за инстанцне чланице
- Метод *toString()* за генерисање *String*-репрезентације тачке у облику *(x, y)*.
- Приликом исписивања објекта класе *Tacka* није неопходно позивати метод *toString()* над објектом, он ће подразумевано бити позван. Само се методу за испис прослеђује референца на објекат.

Класа *Duz* има две инстанцне променљиве, *pocetak* и *kraj*, типа *Tacka*, које представљају почетну и крајњу тачку дужи.
Садржи следеће методе:

- Конструктор на основу задате почетне и крајње тачке. Конструктор се може дефинисати на два начина.
  Први начин је да инстанцне променљиве *pocetak* и *kraj* нове дужи реферишу на идентичне копије тачака *pocetak* и *kraj* које су прослеђене као аргументи конструктору. Промена неке од ових тачака неће утицати на инстанцне променљиве *pocetak* и *kraj* нове дужи.
  ```
  this.pocetak = new Tacka(pocetak);
  this.kraj = new Tacka(kraj);
  ```
  Други начин је да инстанцне променљиве *pocetak* и *kraj* реферишу на постојеће тачке *pocetak* и *kraj*. Промена неке од ових тачака условиће промену инстанцних променљивих нове дужи.
  ```
  this.pocetak = pocetak;
  ```

```
this.kraj = kraj;
```

- Конструктор на основу задатих координата почетне и крајње тачке.
- Метод *duzina()* рачуна дужину дужи. Дужина дужи је растојање између почетне и крајње тачке.
- Метод *presek()* одређује тачке пресека правих одређених двема дужима, при чему су праве задате параметарски.

Параметарска једначина праве одређене првом дужи је:

$$x = x\_pocetak_1 + (x\_kraj_1 - x\_pocetak_1) * t \quad (1)$$

$$y = y\_pocetak_1 + (y\_kraj_1 - y\_pocetak_1) * t \quad (2)$$

Параметарска једначина праве одређене другом дужи је:

$$x = x\_pocetak_2 + (x\_kraj_2 - x\_pocetak_2) * s \quad (3)$$

$$y = y\_pocetak_2 + (y\_kraj_2 - y\_pocetak_2) * s \quad (4)$$

Изједначавањем десних страна једанкости (1) и (3), може се добити израз за параметар *s*:

$$s=(x\_pocetak_1-x\_pocetak_2+(x\_kraj_1-x\_pocetak_1)*t) \; / \; (x\_kraj_2-x\_pocetak_2) \quad (5)$$

Изједначавањем десних страна једнакости (2) и (4) и заменом параметра *s* изразом (5), добија се израз за параметар *t*:

$$t=((x\_pocetak_2-x\_pocetak_1) * (y\_kraj_2-y\_pocetak_2) - (x\_kraj_2x\_pocetak_2)*(y\_pocetak_2-y\_pocetak_1)) \; / \; ((x\_kraj_1-x\_pocetak_1)*(y\_kraj_2-y\_pocetak_2) - (x\_kraj_2-x\_pocetak_2)*(y\_kraj_1y\_pocetak_1)) \quad (6)$$

Уколико је именилац израза (6) једнак 0, праве су паралелне, пресек не постоји и метод враћа *null*.

Иначе, рачунају се *x* и *y* координата пресечне тачке, тако што се у једначине (1) и (2) уместо параметра *t* уврсти израз (6).

- Метод *toString()* за формирање *String*-репрезентације дужи у облику
  (pocetak.x, pocetak.y) : (kraj.x, kraj.y)
- Приликом исписивања објекта класе *Duz* на излаз, није неопходно позивати метод *toString()* над објектом, он ће подразумевано бити позван. Само се методу за испис прослеђује референца на објекат.

Класа *TestGeometrija* тестира рад са тачкама и дужима.
Најпре се креирају две тачке *pocetak* и *kraj*. На основу њих се креира дуж *duz1*. Још једна дуж, *duz2*, креира се задавањем координата почетне и крајње тачке. Исписују се обе дужи и рачуна њихов пресек.
Потом се тестира како померање тачке *kraj* утиче на прву дуж. Резултат зависи од начина на који је дефинисан конструктор класе *Duz*. Ако је коришћен први начин, дуж *duz1* неће бити промењена померањем тачке *kraj*, у супротном хоће.

Решење:

Tacka.java

```
package geometrija;

public class Tacka
{
   // Instancne promenljive
   private double x;
   private double y;

   // Podrazumevani konstruktor postavlja tacku u koordinatni pocetak
   public Tacka()
   {}
   //Konstruktor sa datim vrednostima za koordinate tacke
   public Tacka(double x, double y)
   {
      this.x = x;
      this.y = y;
   }
```

```
   // Copy-konstruktor
   public Tacka(final Tacka tacka)
   {
      this(tacka.x, tacka.y); // x = tacka.x; y = tacka.y;
   }

   // Metod za pomeranje tacke duz x i y ose
   public void pomeri(double x_pomeraj, double y_pomeraj)
   {
      x += x_pomeraj;
      y += y_pomeraj;
   }

   //Metod za racunanje rastojanja do zadate tacke
   public double rastojanje(final Tacka tacka)
   {
      return Math.sqrt( (x - tacka.x) * (x - tacka.x)
                + (y - tacka.y) * (y - tacka.y));
   }


   //Metod koji vraca vrednost x koordinate tacke
   public double getX()
   {
      return x;
   }

   //Metod koji vraca vrednost y koordinate tacke
   public double getY()
   {
      return y;
   }

   //Metod koji postavlja vrednost x koordinate tacke na datu vrednost
   public void setX(double x)
   {
      this.x = x;
   }

   //Metod koji postavlja vrednost y koordinate tacke na datu vrednost
   public void setY(double y)
   {
      this.y = y;
   }

   //Metod za generisanje String reprezentacije tacke
   public String toString()
   {
      return "(" + x + ", " + y + ")";
      // return "(" + Double.toString(x) + ", " + Double.toString(y) + ")";
      // return "(" + String.valueOf(x) + ", " + String.valueOf(y) + ")";
   }
}
```

Duz.java

```
package geometrija;

public class Duz
{
   // Instancne promenljive: dve tacke - pocetak i kraj duzi
   private Tacka pocetak;
   private Tacka kraj;

   // Konstruktor na osnovu zadate pocetne i krajnje tacke
   public Duz(final Tacka pocetak, final Tacka kraj)
   {
```

```
      // Prvi nacin
      this.pocetak = new Tacka(pocetak);
      this.kraj = new Tacka(kraj);

      // Drugi nacin
      // this.pocetak = pocetak;
      // this.kraj = kraj;
   }

   // Konstruktor na osnovu zadatih koordinata pocetne i kranje tacke
   public Duz(double x_pocetak, double y_pocetak, double x_kraj, double y_kraj)
   {
      // Kreiramo pocetnu i krajnu tacku duzi na osnovu zadatih koordinata
      pocetak = new Tacka(x_pocetak, y_pocetak);
      kraj = new Tacka(x_kraj, y_kraj);
   }

   // Metod za racunanje duzine duzi
   public double duzina()
   {
      return pocetak.rastojanje(kraj); // return kraj.rastojanje(pocetak);
   }

   // Metod za odredjivanje tacke preseka dveju duzi
   public Tacka presek(final Duz duz2)
   {
      Tacka presek = new Tacka(0, 0);

      // Racunaju se brojilac i imenilac parametra t dobijenog za presecnu tacku
      double brojilac =
        (duz2.pocetak.getX() - this.pocetak.getX()) *
                      (duz2.kraj.getY() - duz2.pocetak.getY())
        - (duz2.kraj.getX() - duz2.pocetak.getX()) *
                      (duz2.pocetak.getY() - this.pocetak.getY());

      double imenilac =
        (this.kraj.getX() - this.pocetak.getX()) *
                      (duz2.kraj.getY() - duz2.pocetak.getY())
        - (this.kraj.getY() - this.pocetak.getY()) *
                      (duz2.kraj.getX() - duz2.pocetak.getX());

      // Ukoliko je imenilac jednak 0, prave koje su odredjene duzima su
      // paralelne, te presek ne postoji i vraca se null
      if(imenilac == 0)
        return null;

      // Inace, racunaju se x i y koordinata presecne tacke
      presek.setX(
        this.pocetak.getX() + (this.kraj.getX() -
                        this.pocetak.getX()) * brojilac / imenilac );
      presek.setY(
        this.pocetak.getY() + (this.kraj.getY() -
                       this.pocetak.getY()) * brojilac / imenilac );
      return presek;
   }

   public String toString()
   {
      return pocetak + ":" + kraj;
      // return pocetak.toString() + ":" + kraj.toString();
   }

}
```

TestGeometrija.java

```
package geometrija;
```

```
public class TestGeometrija
{
   public static void main(String[] args)
   {
      // Kreiraju se dve tacke.
      Tacka pocetak = new Tacka(0.0, 2.0);
      Tacka kraj = new Tacka(5.0, 1.0);

      System.out.println("Kreirane su dve tacke " + pocetak + " i " + kraj);
      Duz duz1 = new Duz(pocetak, kraj);
      // Kreira se druga duz sa zadatim koordinatama pocetne i krajnje tacke.
      Duz duz2 = new Duz(1.0, 1.0, 4.0, 4.0);

      System.out.println("Kreirane su duzi " + duz1 + " i " + duz2);
      // Odradjivanje i ispis preseka pravih koje su odredjene datim duzima
      System.out.println("Presek je: " + duz2.presek(duz1));
      // Pomeranje tacke kraj.
      kraj.pomeri(1.0, -4.0);

      System.out.println("Posle pomeranja imamo duzi " + duz1 + " i " + duz2);
      System.out.println("Novi presek je " + duz1.presek(duz2));
   }
}
```

7. Матрице. Написати програм за рад са квадратним матрицама. Од операција обезбедити: креирање јединичне матрице задате димензије, креирање нула-матрице задате димензије, креирање матрице када су познати њени елементи, учитавање елемената матрице са стандардног улаза, одређивање димензије матрице, одређивање: транспоноване матрице, детерминанте матрице, збира, разлике и производа двеју матрица, као и испис матрице на стандардни излаз у квадратном облику.

Објашњење:

Класа *Matrica* има инстанцну променљиву *m* која представља дводимензиони низ елемената матрице. Класа садржи следеће методе:

- Конструктор који прима један аргумент, *n*, типа *int*, креира јединичну матрицу димензије *n*. Прва линија у телу овог конструктора креира дводимензиони низ одговарајућих димензија чији се елементи аутоматски иницијализују нулама због своје нумеричке природе (у питању су реални бројеви типа *double*). Потом се елементи главне дијагонале (чији је индекс врсте једнак индексу колоне) постављају на јединице.
- Конструктор који као аргумент прима дводимензиони низ дефинише дводимензиони низ истих димензија које има и задати дводимензиони низ, а потом, копира елемент по елемент из задатог у креирани низ.
- Копи-конструктор само позива претходну верзију конструктора преносећи му дводимензиони низ матрице која се копира као аргумент.
- Статички метод *ucitaj()* најпре захтева од корисника да унесе димензију матрице, дефинише дводимензиони низ одговарајућих димензија, а потом, у двострукој петљи, елементе које корисник унесе са стандардног улаза смешта на одговарајуће локације у дводимензионом низу. Коначно, од формираног дводимензионог низа креира се матрица која представља повратну вредност метода.
- Статички метод *nula()* креира и враћа нула-матрицу задате димензије.
- Метод *dimenzija()* враћа димензију текуће квадратне матрице.
- Метод *transponovana()* рачуна и враћа транспоновану матрицу текуће матрице. Транспонована матрица дате матрице је матрица добијена заменом места свих врста задате матрице одговарајућим колонама или обрнуто.
- Метод *plus()* рачуна и враћа збир текуће и задате матрице. Матрице које се сабирају морају бити једнаких димензија да би сабирање било могуће. Ако то није случај, метод избацује изузетак типа *RuntimeException*. За изузетке овог типа је карактеристично да код који може довести до њиховог избацивања није неопходно ограђивати одговарајућим *try-catch* блоком, тј. нисмо дужни да их хватамо. Збир двају матрица је матрица димензије једнаке димензији сабирака чији је сваки елемент једнак збиру одговарајућих елемената матрица сабирака.
- Метод *minus()* рачуна и враћа разлику текуће и задате матрице. Матрице морају бити једнаких димензија да би операција била могућа. Као и претходни метод, метод *minus()* у случају да овај услов није испуњен избацује изузетак типа *RuntimeException*. Разлика двају матрица је матрица димензије једнаке димензији матрица операнада чији је сваки елемент једнак разлици одговарајућих елемената матрице умањеника и матрице умањиоца.
- Метод *puta()* рачуна и враћа производ текуће и задате матрице. Да би матрице могле да се множе, број колона прве матрице мора бити једнак броју врста друге матрице. Ако то није случај, метод избацује изузетак типа *RuntimeException*. Производ двају матрица димензија $m \times p$ и $p \times n$ је матрица димензије $m \times n$ чији се произвољни елемент, нпр. елемент који се налази у *i*-тој врсти и *j*-тој колони, образује тако што се елементи *i*-те врсте прве матрице помноже одговарајућим елементима *j*-те колоне друге матрице и добијени производи саберу.

Свакој квадратној матрици придружује се реални број који се назива детерминантом те матрице.

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \qquad D=\det(A)=\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

Детерминанта која се добија из дате детерминанте D одбацивањем *i*-те врсте и *j*-те колоне назива се минором елемента a_{ij} и обележава се са M_{ij}. Развојем по елементима *i*-те врсте матрице A, њена детерминанта се може израчунати по формули:

$$D=a_{i1}A_{1}+a_{i2}A_{12}+\cdots+a_{in}A_{in}=\sum_{k=1}^{n}a_{ik}A_{ik},\ i=1,2,\cdots,n$$

где је A_{ik} тзв. кофактор елемента a_{ij}, дефинисан са:

$$A_{ij}=(-1)^{i+j}M_{ij}.$$

- Метод *minorMat()* рачуна матрицу минора елемента чији су индекси врсте и колоне задати као аргументи метода, односно матрицу из које су изостављене врста и колона одређене аргументима метода. Резултујућа матрица је квадратна и димензије за један мање од полазне матрице. Метод се креће кроз полазну матрицу, врста по врста, и преписује све изузев елемената који припадају задатој врсти, односно колони. Елементи задате врсте се прескачу наредбом *continue* која спречава копирање елемената те врсте и увећање индекса врсте резултујуће матрице, док се, из свих осталих врста, у резултујућу матрицу копирају само елементи полазне матрице који не припадају задатој колони (индекс колоне текућег елемента различит је од индекса задате колоне). Под коментаром је алтернативни начин за добијање матрице минора елемента одређеног аргументима метода. Наиме, елементи који се у полазној матрици налазе "изнад" задате врсте и "лево" од задате колоне, у матрици минора имају исте индексе врсте и колоне као и у полазној матрици; елементи који се у полазној матрици налазе "изнад" задате врсте и "десно" од задате колоне, у матрици минора имају исти индекс врсте као у полазној матрици, док им је индекс колоне за један мањи; елементи који се у полазној матрици налазе "испод" задате врсте и "лево" од задате колоне, у матрици минора имају исти индекс колоне као у полазној матрици, док им је индекс врсте за један мањи; најзад, елементи који се у полазној матрици налазе "испод" задате врсте и "десно" од задате колоне, у матрици минора имају индексе врсте и колоне који су, оба, за по један мањи од одговарајућих индекса које имају у полазној матрици. Елементи који се налазе баш у задатој врсти или задатој колони полазне матрице немају никакве индексе у резултујућој матрици јер се и не налазе у њој.
- Метод *determinanta()* рекурзивно рачуна детерминанту текуће матрице на следећи начин: ако је матрица димензије 1, њена детерминанта једнака је јединому елементу матрице. Иначе, детерминанта се рачуна развојем по елементима 0-те врсте, при чему се за одређивање матрице одговарајућег минора позива метод *minorMat()*, док се за рачунање самог минора рекурзивно позива метод *determinanta()* над добијеном матрицом минора.
- Метод *equals()* тестира једнакост текуће и задате матрице. Две матрице су једнаке акко су истих димензија и сви парови одговарајућих елемената су им једнаки.
- Метод *toString()* враћа *String*-репрезентацију матрице у жељеном квадратном облику. Елементи исте врсте смештају се у један ред раздвојени по једном белином, при чему се сваки елемент записује у пољу ширине 9 са 4 места иза децималне тачке.

У тест-класи креирају се нула-матрица и јединична матрица задатих димензија, а потом се са стандардног улаза уносе две матрице и тестирају имплементиране операције над њима.

Решење:

Matrica.java

```
package matrice;

import java.util.Scanner;

public class Matrica
{
   private static Scanner sc = new Scanner(System.in);
   private double m[][];

   // Kreira jedinicnu matricu
   public Matrica(int n)
   {
      m = new double[n][n];
      for (int i = 0; i < n; i++)
         m[i][i] = 1;
   }

   public Matrica(double m[][])
   {
```

```
   this.m = new double[m.length][m.length];
   for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m.length; j++)
         this.m[i][j] = m[i][j];

}

public Matrica(final Matrica mat)
{
   this(mat.m);
}

public static Matrica ucitaj()
{
   System.out.print("Unesite dimenziju matrice: ");
   int n = sc.nextInt();
   double m[][] = new double[n][n];
   System.out.print("Elementi: ");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         m[i][j] = sc.nextDouble();
   return new Matrica(m);
}

public static Matrica nula(int n)
{
   return new Matrica(new double[n][n]);
}

public int dimenzija()
{
   return m.length;
}

public Matrica transponovana()
{
   double m[][] = new double[this.m.length][this.m.length];
   for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m.length; j++)
         m[i][j] = this.m[j][i];
   return new Matrica(m);
}

public Matrica plus(Matrica B)
{
   if (B.m.length != m.length)
      throw new RuntimeException("Neodgovarajuce dimenzije matrica.");
   double[][] zbir = new double[m.length][m.length];
   for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m.length; j++)
         zbir[i][j] = m[i][j] + B.m[i][j];
   return new Matrica(zbir);
}

public Matrica minus(Matrica B)
{
   if (B.m.length != m.length)
      throw new RuntimeException("Neodgovarajuce dimenzije matrica.");
   double[][] razlika = new double[m.length][m.length];
   for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m.length; j++)
         razlika[i][j] = m[i][j] - B.m[i][j];
   return new Matrica(razlika);
}

public Matrica puta(Matrica B)
{
```

```
    if (m.length != B.m.length)
      throw new RuntimeException("Neodgovarajuce dimenzije matrica");
    double[][] proizvod = new double[m.length][m.length];
    for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m.length; j++)
        for (int k = 0; k < m.length; k++)
          proizvod[i][j] += m[i][k] * B.m[k][j];
    return new Matrica(proizvod);
  }

  public Matrica minorMat(int k, int p)
  {
    if (m.length == 1)
      return null;
    double m[][] = new double[this.m.length - 1][this.m.length - 1];
    // s je indeks tekuce vrste matrice minora
    for (int i = 0, s = 0; i < this.m.length; i++)
    {
      if (i == k)
        continue; // cela ta vrsta se preskace
      // t je indeks tekuce kolone matrice minora
      for (int j = 0, t = 0; j < this.m.length; j++)
        if (j != p)
          m[s][t++] = this.m[i][j];
      s++;
    }

    return new Matrica(m);

  }

  /*
   * public Matrica minorMat(int k, int p)
* {
*   if (m.length == 1)
   *      return null;
   *
   *   double m[][] = new double[this.m.length - 1][this.m.length - 1];
*   for (int i = 0; i < this.m.length; i++)
*      for (int j = 0; j < this.m.length; j++)
*        if (i < k && j < p)
   *          m[i][j] = this.m[i][j];
*        else if (i < k && j > p)
   *          m[i][j - 1] = this.m[i][j];
*        else if (i > k && j < p)
*          m[i - 1][j] = this.m[i][j];
*        else if (i > k && j > p)
   *          m[i - 1][j - 1] = this.m[i][j];
*   return new Matrica(m);
   * }
   */
  public double determinanta()
  {
    if (m.length == 1)
      return m[0][0];

    double det = 0;
    int sign = 1;
    for (int i = 0; i < m.length; i++)
    {
      det += sign * m[0][i] * minorMat(0, i).determinanta();
      sign = -sign;
    }
    return det;
  }

  public boolean equals(Object o)
```

```
   {
      if (!(o instanceof Matrica))
         return false;
      Matrica B = (Matrica) o;
      if (B.m.length != m.length)
         return false;
      for (int i = 0; i < m.length; i++)
         for (int j = 0; j < m.length; j++)
            if (m[i][j] != B.m[i][j])
               return false;
      return true;
   }

   public String toString()
   {
      StringBuffer str = new StringBuffer();
      for (int i = 0; i < m.length; i++)
      {
         for (int j = 0; j < m.length; j++)
            str.append(String.format("%9.4f", m[i][j]) + " ");
         str.append("\n");
      }
      return str.toString();
   }
}
```

TestMatrica.java

```
package matrice;

import java.util.Scanner;

public class TestMatrica
{
   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);
      System.out.print("Unesite dimenziju nula-matrice: ");
      int n = sc.nextInt();
      Matrica Z = Matrica.nula(n);
      System.out.println("Z:");
      System.out.println(Z);

      System.out.print("Unesite dimenziju jedinicne matrice: ");
      n = sc.nextInt();
      Matrica I = new Matrica(n);
      System.out.println("I:");
      System.out.println(I);

      System.out.println("Matrica A: ");
      Matrica A = Matrica.ucitaj();
      System.out.println("A:");
      System.out.println(A);

      System.out.println("det(A) = " + A.determinanta());

      System.out.println("Transponovana matrica A: ");
      System.out.println(A.transponovana());

      System.out.println("Matrica B: ");
      Matrica B = Matrica.ucitaj();
      System.out.println("B:");
      System.out.println(B);

      System.out.println("det(B) = " + B.determinanta());

      System.out.println("A + B = \n" + A.plus(B));
```

```
      System.out.println("A - B = \n" + A.minus(B));
      System.out.println("A * B = \n" + A.puta(B));

   }

}
```

Коментар: Сиромашнија имплементација класе Matrica; нема све методе као претходна.

8. *Matrice.* Написати класу *Matrica* којом се дефинише квадратна матрица реалних бројева. Обезбедити конструктор, конструктор копије, рекурзиван метод за рачунање детерминанте матрице, као и метод за учитавање матрице са стандардног улаза. Написати тест класу.

Објашњење:
Класа *Matrica* садржи као инстанцну промењљиву дводимензиони низ *elementi* елемената типа *double.* Садржи следеће методе:

- Конструктор на основу датих елемената.
  Прво се иницијализује чланица *elementi:*
  ```
  elementi = new double[clanovi.length][clanovi.length];
  ```
  Овим је у меморији резервисан простор за одговарајући број елемената.
  Затим се у петљи иницијализују елементи матрице датим вредностима.
- Помоћни метод *element(i, j)* враћа елемент на позицији [i,j] у матрици и помоћни метод *dimenzija()* који враћа димензију матрице
- Метод *determinanta()* рачуна детерминанту матрице рекурзивно помоћу минора.

Излаз из рекурзије је случај када је димензија матрице једнака 1 или 2 и када детерминанта може директно да се срачуна. Могуће је то урадити и за димензију 3, али је у решењу то стављено у рекурзивни корак.
Рекурзивни корак подразумева рачунање детерминанте коришћењем развоја по кофакторима на првој врсти матрице по формули:

$$\sum_{j=0}^{dim-1} A_{0,j} C_{0,j} = \sum_{j=0}^{dim-1} A_{0,j} (-1)^j M_{0,j}$$

где $C_{0,j}$ представља матрицу кофактора, тј. $C_{0,j} = (-1)^j M_{0,j}$, а $M_{0,j}$ је минор који је детерминанта матрице која се добије кад се из матрице A уклони *0*-та врста и *j*-та колона.
Напомена: оптималније је да се развој врши по врсти или колони која садржи највећи број нула, али то захтева додатне провере.

- Помоћни метод *podmatrica()* генерише подматрицу без елемената *i*-те врсте и *j*-те колоне полазне матрице. Димензија подматрице је за један мања од димензије полазне матрице. Индекси *k* и *l* служе за пролаз кроз врсте, односно колоне полазне матрице, искључујући *i*-ту врсту и *j*-ту колону. Индекси *m* і *n* су индекси елемената подматрице.
- Метод *ucitajMatricu()* учитава матрицу са стандардног улаза. Аргумент метода је референца на објекат за парсирање улаза. Формат улаза је такав да је у првој линији димензија матрице, а затим у свакој новој линији елементи врста матрице.

У класи *TestMatrice* се најпре креира објекат *skener* за парсирање улаза, а потом се учитава матрица. На стандардни излаз се, поред матрице, исписују и подматрице коришћене за рачунање детерминанте, као и детерминанта матрице.

Решење:

Matrica.java

```
package matrice;

package matrice;

import java.util.Scanner;

public class Matrica
{
   private double[][] elementi;
```

```
// Konstruktor na osnovu datih elemenata matrice
public Matrica(double[][] clanovi)
{
   elementi = new double[clanovi.length][clanovi.length];
   for(int i = 0; i < clanovi.length; i++)
      for(int j = 0; j < clanovi.length; j++)
         elementi[i][j] = clanovi[i][j];
}

// Konstruktor kopije
public Matrica(final Matrica matrica)
{
   int dimenzija = matrica.elementi.length;
   elementi = new double[dimenzija][dimenzija];
   for(int i = 0; i < dimenzija; i++)
      for(int j = 0; j < dimenzija; j++)
         elementi[i][j] = matrica.element(i, j);
}

// Metod vraca element matrice na poziciji [i,j]
public double element(int i, int j)
{
   return elementi[i][j];
}

// Dimenzija matrice
public int dimenzija()
{
   return elementi.length;
}

// Metod koji racuna determinantu matrice rekurzivno
public double determinanta()
{
   if(dimenzija() == 1)
      return element(0, 0);
   if(dimenzija() == 2)
      return element(0, 0)*element(1, 1) - element(0, 1)*element(1, 0);
   double determinanta = 0.0;
   for(int j = 0; j < dimenzija(); j++)
      determinanta += Math.pow(-1, j) * element(0, j) *
             podmatrica(0, j).determinanta();
   return determinanta;
}

// Metod koji za datu matricu generise podmatricu bez elemenata
// i-te vrste i j-te kolone matrice
public Matrica podmatrica(int i, int j)
{
   int dimenzija = dimenzija();
   double[][] elementiPodmatrice = new double[dimenzija-1][dimenzija-1];
   for(int k = 0, m = 0; k < dimenzija && m < dimenzija-1; k++)
   {
      if(k != i)
      {
         for(int l = 0, n = 0; l < dimenzija && n < dimenzija-1; l++)
         {
            if(l != j)
               elementiPodmatrice[m][n++] = element(k, l);
         }
         m++;
      }
   }
   return new Matrica(elementiPodmatrice);
}
```

```
   // Metod koji ucitava matricu.
   public static Matrica ucitajMatricu(final Scanner skener)
   {
      System.out.print("Unesite dimenziju matrice: ");
      int dim = skener.nextInt();
      System.out.println("Unesite elemente matrice:");
      double[][] elementi = new double[dim][dim];
      for(int i = 0; i < dim; i++)
         for(int j = 0; j < dim; j++)
            elementi[i][j] = skener.nextDouble();
      return new Matrica(elementi);
   }

   public String toString()
   {
      StringBuffer matrica = new StringBuffer();
      for(int i = 0; i < dimenzija(); i++)
      {
         for(int j = 0; j < dimenzija(); j++)
            matrica.append(element(i, j) + "  ");
         matrica.append("\n");
      }
      return matrica.toString();
   }
}
```

TestMatrice.java

```
package matrice;

import java.util.Scanner;

public class TestMatrice
{
   public static void main(String[] args)
   {
      // Parser ulaza
      Scanner skener = new Scanner(System.in);
      // Ucitavanje matrice
      Matrica m = Matrica.ucitajMatricu(skener);
      skener.close();
      System.out.println(m);
      // Ispis podmatrica koje su koriscene za racunanje
      // determinante u datoteku
      for(int j = 0; j < m.dimenzija(); j++)
         System.out.println(m.podmatrica(0, j));

      // Ipis vrednosti determinante u datoteku
      System.out.println("Determinanta matrice je: " + m.determinanta());
   }
}
```

9. ***Релација.*** Написати апликацију у којој се испитује да ли је релација рефлексивна, симетрична и транзитивна. Релација је представљена квадратном матрицом. Користити класу *Matrica* из задатка ???. У матрици се на позицији [i,j] налази *1*, ако су *i* и *j* у релацији, иначе *0*. Димензија и елементи матрице уносе се из датотеке. Исписати одговарајуће поруке на стандардни излаз.

Објашњење:

Релација R над доменом D је:

- рефлексивна уколико за сваки елемент x из D важи R(x), тј. уколико је сваки елемент у релацији R са самим собом.
- симетрична уколико за свака два елемента x и y из D важи R(x, y) => R(y, x).
- транзитивна уколико за сваку тројку елемената x, y и z из D важи R(x, y) ∧ R(y, z) => R(x, z).

Релација је представљена квадратном матрицом. Класа *Matrica*, дефинисана у задатку ???, користи се као репрезентација квадратне матрице, при чему је тип елемената промењен у *int*. Промењен је и метод *ucitajMatricu()*, у смислу да је додата провера да ли су елементи матрице релације једнаки 0 или 1. Уколико услов није задовољен, метод враћа *null*.
Класа *Relacija*, поред метода *main()* за тестирање рада са релацијама, може да садржи и методе за испитивање рефлексивности, симетричности и транзитивности квадратне матрице која се прослеђује као њихов аргумент. Сваки метод враћа вредност типа *boolean*. Методи су статички, јер нема потребе за креирањем објеката класе *Relacija* у методу *main()*.
Други начин је да методи за проверу рефлексивности, симетричности и транзитивности буду дефинисани у класи *Matrica*, у ком случају нису статички.
У решењу је примењен први начин.
У методу *main()* се најпре учитава квадратна матрице позивом статичког метода *ucitajMatricu()* класе *Matrica*. Ако метод врати *null*, формат елемената матрице није задовољавајући, тј. нису сви елементи једаки 0 или 1. У том случају се прекида даље извршавање програма позивом метода *System.exit(1)*. Аргумент метода је целобројна вредност која указује на статус при завршетку програма. Вредност 0 означава да је све у реду, док вредност 1 сигнализира некоректност у програму.
Испитивање рефлексивности, симетричности и транзитивности се виши позивом одговарајућих статичких метода.
Исписују се одговарајуће поруке, као и елементи саме матрице. Матрица се учитава са стандрадног улаза.

Решење:

Matrica.java

```
package relacija;

import java.util.Scanner;

public class Matrica
{
   private int[][] elementi;

   // Konstruktor na osnovu datih elemenata matrice
   public Matrica(int[][] clanovi)
   {
      elementi = new int[clanovi.length][clanovi.length];
      for(int i = 0; i < clanovi.length; i++)
         for(int j = 0; j < clanovi.length; j++)
            elementi[i][j] = clanovi[i][j];
   }

   // Konstruktor kopije
   public Matrica(final Matrica matrica)
   {
      int dimenzija = matrica.elementi.length;
      elementi = new int[dimenzija][dimenzija];
      for(int i = 0; i < dimenzija; i++)
         for(int j = 0; j < dimenzija; j++)
            elementi[i][j] = matrica.element(i, j);
   }

   // Metod vraca element matrice na poziciji [i,j]
```

```
public int element(int i, int j)
{
   return elementi[i][j];
}

// Dimenzija matrice
public int dimenzija()
{
   return elementi.length;
}

// Metod koji racuna determinantu matrice rekurzivno
public double determinanta()
{
   if(dimenzija() == 1)
      return element(0, 0);
   if(dimenzija() == 2)
      return element(0, 0)*element(1, 1) - element(0, 1)*element(1, 0);
   double determinanta = 0.0;
   for(int j = 0; j < dimenzija(); j++)
      determinanta += Math.pow(-1, j) * element(0, j) *
              podmatrica(0, j).determinanta();
   return determinanta;
}


// Metod koji za datu matricu generise podmatricu bez elemenata
// i-te vrste i j-te kolone matrice
public Matrica podmatrica(int i, int j)
{
   int dimenzija = dimenzija();
   int[][] elementiPodmatrice = new int[dimenzija-1][dimenzija-1];
   for(int k = 0, m = 0; k < dimenzija && m < dimenzija-1; k++)
   {
      if(k != i)
      {
         for(int l = 0, n = 0; l < dimenzija && n < dimenzija-1; l++)
         {
            if(l != j)
               elementiPodmatrice[m][n++] = element(k, l);
         }
         m++;
      }
   }
   return new Matrica(elementiPodmatrice);
}

// Metod koji ucitava matricu.
public static Matrica ucitajMatricu(final Scanner skener)
{
   System.out.print("Unesite dimenziju matrice:");
   int dim = skener.nextInt();
   System.out.println("Unesite elemente matrice:");
   int[][] elementi = new int[dim][dim];
   for(int i = 0; i < dim; i++)
      for(int j = 0; j < dim; j++)
      {
         elementi[i][j] = skener.nextInt();
         if(elementi[i][j] != 0 && elementi[i][j] != 1)
            return null;
      }
   return new Matrica(elementi);
}

public String toString()
{
   StringBuffer matrica = new StringBuffer();
```

```
      for(int i = 0; i < dimenzija(); i++)
      {
         for(int j = 0; j < dimenzija(); j++)
            matrica.append(element(i, j) + "  ");
         matrica.append("\n");
      }
      return matrica.toString();
   }
}
```

Relacija.java

```
package relacija;

import java.util.Scanner;

import matrice.Matrica;

public class Relacija
{
   // Metod koji ispituje da li je relacija refleksivna
   public static boolean refleksivna(Matrica rel)
   {
      for(int i = 0; i < rel.dimenzija(); i++)
         if(rel.element(i, i) == 0)
            return false;
      return true;
   }

   // Metod koji ispituje da li je relacija simetricna
   public static boolean simetricna(Matrica rel)
   {
      for(int i = 0; i < rel.dimenzija(); i++)
         for(int j = 0; j < i; j++)
            if(rel.element(i, j) != rel.element(j, i))
               return false;
      return true;
   }

   // Metod koji ispituje da li je relacija tranzitivna
   public static boolean tranzitivna(Matrica rel)
   {
      for(int i = 0; i < rel.dimenzija(); i++)
         for(int j = 0; j < rel.dimenzija(); j++)
            for(int k = 0; k < rel.dimenzija(); k++)
               if(i != j && j != k)
                  if(rel.element(i, j) == 1 && rel.element(j, k) == 1 &&
                     rel.element(i, k) == 0)
                    return false;
      return true;
   }

   public static void main(String[] args)
   {
         // Parser ulaza
         Scanner skener = new Scanner(System.in);
         // Ucitavanje matrice
         Matrica rel = Matrica.ucitajMatricu(skener);
         if(rel == null)
         {
            System.out.println("Elementi matrice moraju biti jednaki 0 ili 1");
            System.exit(1);
         }
         skener.close();
         System.out.println(rel);
```

```
        // Refleksivnost
        if(refleksivna(rel))
           System.out.println("Relacija je refleksivna");
        else
           System.out.println("Relacija nije refleksivna");

        // Simetricnost
        if(Relacija.simetricna(rel))
           System.out.println("Relacija je simetricna");
        else
           System.out.println("Relacija nije simetricna");

        // Tranzitivnost
        if(Relacija.tranzitivna(rel))
           System.out.println("Relacija je tranzitivna");
        else
           System.out.println("Relacija nije tranzitivna");

   }
}
```

10. *Многоугао*. Написати класу *Mnogougao* којом се описује многоугао помоћу низа својих темена. Теме је представљено објектом класе *Tacka*. Обезбедити конструктор на основу датог дводимензионалног низа координата темена, као и конструктор на основу датог низа темена многоугла. Написати следеће методе:

- Метод који враћа низ чији су елементи дужине страница многоугла
- Методе који рачунају обим и површину многоугла
- Метод који испитује да ли је многоугао конвексан
- Метод који испитује да ли је многоугао квадрат
- Метод који одређује број дијагонала многоугла
- Метод за добијање *String*-репрезентације многоугла коју чине координате темена многоугла и дужине страница многоугла
- Метод за учитавање многоугла из датотеке. У првој линији датотеке је број темена многоугла, а затим у свакој наредној координате појединачних темена.

Написати и тест-класу где прво треба учитати многоугао из датотеке, потом срачунати обим и површину, испитати да ли је многоугао конвексан и, ако јесте, да ли је квадрат. Резултате исписати такође у датотеку.

Објашњење:

За читање података из датотеке користе се следеће класе:

```
java.io.File;
java.io.FileNotFoundException;
java.util.Scanner;
```

Да би се креирао парсер улазног тока података, који долази из датотеке, а не са стандардног улаза, конструктору класе *Scanner* прослеђује се објекат класе *File*. Класа *File* представља апстрактну репрезентацију датотеке. Инстанца ове класе креира се тако што се конструктору проследи путања до датотеке између двоструких наводника. Уколико се датотека налази у истом директоријуму у коме је и пакет, довољно је навести само име датотеке. Ако се јаве проблеми при отварању датотеке, или датотека са датим именом не постоји на датој локацији, избацује се изузетак типа *FileNotFoundException.* Због тога је неопходно да део кода где се врши отварање датотеке буде унутар *try* блока. Обрада изузетка врши се у одговарајућем *catch* блоку.

Први начин:

```
Scanner sc=null;
try
{
   sc=new Scanner(new File("brojevi.txt"));
  //... Citanje i obrada podataka
```

```
}
catch(FileNotFoundException e)
{
          System.out.println(e.getMessage());
}
// ... Poslednji catch blok
catch(...)
{}
// kraj programa
```

Други начин:

```
Scanner sc=null;
try
{
   sc=new Scanner(new File("brojevi.txt"));
}
catch(FileNotFoundException e)
{
  System.out.println(e.getMessage());
   System.exit(1); // return;
}
// ... Citanje i obrada podataka
```

Преостали код, који обухвата читање података из датотеке и њихову обраду, може доћи након наредбе за отварање датотеке унутар *try* блока, или после последњег *catch* блока. У првом случају, како после *try – catch* блока нема других наредби, нема потребе за позивом метода *System.exit()* у *catch* блоку.

По завршетку рада са улазном датотеком, методом *close()* затвара се улазни ток.

За испис података у датотеку користе се следеће класе:

```
java.io.FileWriter;
java.io.IOException;
java.io.PrintWriter;
```

Класа *PrintWriter* дефинише објекат којим се врши испис форматизоване репрезентације објеката у излазни ток. Уколико се ради о карактерском излазном току, конструктору ове класе прослеђује се објекат класе *FileWriter*, мада је овај случај обухваћен и када се проследи објекат класе *File.*

Конструктору класе *FileWriter* такође треба проследити путању до улазне датотеке између двоструких наводника.

Ако је коришћена класа *FileWriter* уместо класе *File*, неопходно је обрадити изузетак типа *IOException.*

```
PrintWriter izlaz;
try
{
   izlaz = new PrintWriter(new FileWriter("izlaz.txt"));
}
catch(IOException IOe)
{
   System.out.println("Neuspela operacija pripreme za
             upis u fajl izlaz.txt");
   return;
}
// Ispis podataka u datoteku
```

Као и у случају читања података из датотеке, код за испис у датотеку може бити део *try* блока у коме се отвара датотека за испис, или може доћи после последњег *catch* блока.

По завршетку рада са излазном датотеком, методом *close()* затвара се излазни ток.

Класа *Mnogougao* има инстанцну промењиву *temena* која представља низ темена многоугла. Темена су дефинисана као објекти класе *Tacka.*

Садржи следеће методе:

- Конструктор на основу датог дводимензионалног низа координата темена
- Конструктор на основу датог низа темена
- Метод *stranice()* који одрађује низ чији су елементи дужине страница многоугла
- Метод *brojDijagonala()* који одређује број дијагонала многоугла
- Метод *obim()* за рачунање обима
- Метод *povrsina()* за рачунање површине. Користе се помоћни методи *kvadrat()* и *strIdijagTemeA1()*. Методом *kvadrat()* проверава се да ли је у питању квадрат и у том случају се површина рачуна по познатој формули. У супротном, многоугао се разложи на троуглове са заједничким теменом A_1 (прво у низу темена), а површина сваког се рачуна Хероновим обрасцем $\sqrt{s(s-a)(s-b)(s-c)}$, где је *s* полуобим троугла. Уколико је многоугао баш троугао, површина се израчунава након првог корака.
- Метод *strIdijagTemeA1()* одређује низ чији су елементи дужине страница и дијагонала из темена A_1, тј. растојања темена A_1 до преосталих темена многоугла.
- Метод *konveksan()* за проверу конвексности многоугла. За свака два суседна темена многоугла одреди се једначина праве која их садржи и уколико се сва преостала темена налазе са исте стране праве, многоугао је конвексан.
  За два суседна темена одреде се коефицијенти A, B и C једначине праве Ax+By+C=0 која их садржи. Потом се одреди положај следећег темена у односу на праву, тј. одреди се знак израза Ax+By+C када се уврсте координате тог темена. Уколико исти услов задовоље и преостала темена и то важи за сваки пар суседних темена, многоугао је конвексан.
- Метод *toString()* – *String*-репрезентација многоугла обухвата приказ координата његових темена и дужина страница.
- Статички метод *ucitajMnogougao()* учитава многоугао из датотеке. У првој линији датотеке је број темена, а свака следећа линија садржи координате појединачних темена. Аргумент метода је референца на објекат за парсирање улазне датотеке. Креира се низ темена многоугла на основу учитаних координата, а потом и многоугао на основу низа темена.

У класи *TestMnogougao* се сви потребни кораци – отварање улазне датотеке, креирање излазне датотеке и испис тражених резултата, врше у једном *try* блоку. Могућа су три типа изузетка – *FileNotFoundException, InputMismatchException* и *IOException*. Први тип се јавља ако улазна датотека са датом путањом не постоји, други тип уколико за координате темена нису унешени бројеви и трећи тип уколико постоји проблем при креирању излазне датотеке са задатом путањом или проблем при испису.

Решење:

Tacka.java

```
package mnogougao;

public class Tacka
{
   //Instancne promenljive - x i y koordinata tacke
   private double x;
   private double y;

   //Podrazumevani konstruktor postavlja tacku u koordinatni pocetak
   public Tacka()
   {}

   // Konstruktor sa datim vrednostima za koordinate tacke
   public Tacka(double x, double y)
   {
      this.x = x;
      this.y = y;
   }
```

```
   // Konstruktor kopija
   public Tacka(final Tacka tacka)
   {
      this(tacka.x, tacka.y); // x = tacka.x; y = tacka.y;
   }

   // Metod za racunanje rastojanja do zadate tacke
   public double rastojanje(final Tacka tacka)
   {
      return Math.sqrt( (x - tacka.x) * (x - tacka.x)
                + (y - tacka.y) * (y - tacka.y));
   }

   public double getX()
   {
      return x;
   }

   public double getY()
   {
      return y;
   }

   public void setX(double x)
   {
      this.x = x;
   }

   public void setY(double y)
   {
      this.y = y;
   }

   public String toString()
   {
      return "(" + x + ", " + y + ")";
   }
}
```

Mnogougao.java

```
package mnogougao;

import java.util.Scanner;

public class Mnogougao
{
   private Tacka[] temena;

   // Konstruktor na osnovu datog dvodimenzionalnog niza koordinata
   public Mnogougao(final double[][] koordinate)
   {
      temena = new Tacka[koordinate.length];
      int i = 0;
      for(double[] koord : koordinate)
         temena[i++] = new Tacka(koord[0], koord[1]);
   }

   // Konstruktor na osnovu datog niza temena
   public Mnogougao(final Tacka[] tacke)
   {
      temena = new Tacka[tacke.length];
      int i = 0;
      for(Tacka tacka : tacke)
         temena[i++] = new Tacka(tacka);
   }
```

```
// Odredjuje duzine stranica mnogougla
public double[] stranice()
{
   double[] stranice = new double[temena.length];
   for(int i = 0; i < temena.length-1; i++)
      stranice[i] = temena[i].rastojanje(temena[i+1]);
   stranice[stranice.length-1] =
        temena[temena.length-1].rastojanje(temena[0]);
   return stranice;
}

// Odradjuje broj dijagonala mnogougla
public int brojDijagonala()
{
   return temena.length * (temena.length - 3) / 2;
}

// Obim mnogougla
public double obim()
{
   double obim = 0;
   double[] stranice = stranice();
   for(int i = 0; i < stranice.length; i++)
      obim += stranice[i];
   return obim;
}

// Povrsina mnogougla
public double povrsina()
{
   double povrsina = 0;
   double[] stranice = stranice();
   if(Mnogougao.kvadrat(stranice))
      return stranice[0] * stranice[0];
   double[] strIdijag = strIdijagTemeA1();
   for(int i = 0 ; i < stranice.length-2; i++)
   {
      double a = strIdijag[i];
      double b = stranice[i+1];
      double c = strIdijag[i+1];
      double s = (a + b + c) / 2.0;
      povrsina += Math.sqrt(s*(s-a)*(s-b)*(s-c));
   }
   return povrsina;
}

// Duzine stranica i dijagonala iz prvog temena
public double[] strIdijagTemeA1()
{
   double[] strIdijag = new double[temena.length-1];
   for(int i = 0; i < temena.length-1; i++)
      strIdijag[i] = temena[0].rastojanje(temena[i+1]);
   return strIdijag;
}

// Staticki metod za ispitivanje da li je mnogougao kvadrat
public static boolean kvadrat(double[] stranice)
{
   if(stranice.length != 4)
      return false;
   for(int i = 0; i < stranice.length-1; i++)
      if(stranice[i] != stranice[i+1])
         return false;
   return true;
}
```

```
   // Provera konveksnosti
   public boolean konveksan()
   {
      int velicina = temena.length;
      boolean ind = false;
      for(int i = 0; i < velicina-1; i++)
      {
         // Koeficijenti A, B i C jednacine prave koju odredjuju
         // po dva uzastopna temena mnogougla
         double A = temena[i+1].getY() - temena[i].getY();
         double B = temena[i].getX() - temena[i+1].getX();
         double C = temena[i+1].getX()*temena[i].getY() -
              temena[i].getX()*temena[i+1].getY();

         // Odradjuje se polozaj sledeceg temena u odnosu na pravu
         int indeks = (i+2) % velicina;
         ind = A*temena[indeks].getX() + B*temena[indeks].getY() + C > 0;
         // Mnogougao je konveksan ukoliko i preostala temena imaju isti polozaj u
         // odnosu na pravu
         for(int j = 0; j < velicina; j++)
            if(j != i && j != i+1 && j != indeks &&
                 (A*temena[i].getX() + B*temena[i].getY() + C > 0) != ind)
               return false;
      }
      return true;
   }

   // String-reprezentacija mnogougla
   public String toString()
   {
      StringBuffer str = new StringBuffer("Temena mnogougla:\n");
      for(int i = 0; i < temena.length; i++)
         str.append(temena[i] + " ");
      str.append("\nStranice mnogougla:\n");
      double[] stranice = stranice();
      for(int i = 0; i < stranice.length; i++)
         str.append(stranice[i] + " ");
      return str.toString();
   }

   // Staticki metod koji ucitava mnogougao
   public static Mnogougao ucitajMnogougao(Scanner skener)
   {
      int brojTemena = skener.nextInt();
      Tacka[] temena = new Tacka[brojTemena];
      for(int i = 0; i < brojTemena; i++)
         temena[i] = new Tacka(skener.nextDouble(), skener.nextDouble());
      return new Mnogougao(temena);
   }

}
```

TestMnogougao.java

```
package mnogougao;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.InputMismatchException;
import java.util.Scanner;

public class TestMnogougao
{
```

```
   public static void main(String[] args)
   {
      try
      {
         // Parser ulaza, tj. datoteke
         Scanner skener = new Scanner(new File("mnogougao.txt"));
         // Ucitavanje mnogougla
         Mnogougao mnogougao = Mnogougao.ucitajMnogougao(skener);
         skener.close();
         // Objekat za ispis u datoteku
         PrintWriter pisac = new PrintWriter(new FileWriter("izlaz.txt"));
         // Ispis mnogougla u datoteku
         pisac.println(mnogougao);
         // Ispis preostalih podataka o mnogouglu.
         pisac.println("Obim je: " + mnogougao.obim() +
               "\tPovrsina je: " + mnogougao.povrsina());
         if(mnogougao.konveksan())
         {
            pisac.println("Mnogougao je konveksan");
           if(Mnogougao.kvadrat(mnogougao.stranice()))
              pisac.println("Mnogougao je kvadrat");
         }
         else
            pisac.println("Mnogougao je konkavan");

         pisac.close();

      }
      // Izuzetak koji se izbacuje ako ne postoji ulazni fajl
      catch(FileNotFoundException e1)
      {
         System.out.println(e1.getMessage());
      }
      // Izuzetak koji se javlja usled nekorektno unesenih koordinata
      // temena mnogougla
      catch(InputMismatchException e2)
      {
         System.out.println("Nekorektno unesene koordinate temena.");
      }
     // Izuzetak koji se izbacuje usled problema pri ispisu
      catch(IOException e3)
      {
         System.out.println(e3.getMessage());
      }
   }
}
```

11. *Велики бројеви*. Написати класу за рад са великим ненегативним целим бројевима.
Претпоставити да бројеви немају више од 2000 цифара.
Од операција имплементирати: унос "великог" броја са тастатуре, креирање "великог" броја од цифара задатог целог броја, сабирање "великих" бројева, множење "великих" бројева, рачунање факторијела ненегативног целог броја и рачунање задатог елемента Фибоначијевог низа.
Предефинисати методе *compareTo()* и *equals()* наслеђене од класе *Object* тако да раде на одговарајући начин са "великим" бројевима.
Написати потом и тест-класу.

Објашњење:

У класи *Broj* имамо константу *MAX_CIFRE* која представља максималан број цифара великог броја. За сваки велики број памтимо низ његових цифара – инстанцна променљива *cifre*, као и број цифара у том низу – инстанцна променљива *duzina*. Цифре се у низ смештају тако да на позицији 0 буде цифра најмање тежине броја.

Конструктор са једним целобројним аргументом креира велики број од цифара броја који му је прослеђен као аргумент. Цифра најмање тежине броја рачуна се као остатак при целобројном дељењу броја са 10. Издвајање једне по једне по једне цифре броја и њихово смештање у низ врши се у петљи на следећи начин: све док не обрадимо све цифре (број је различит од 0), очитавамо цифру најмање тежине броја и смештамо је на одговарајуће место у низу, а потом ту цифру "бришемо" са десног краја његовим целобројним дељењем са 10. Као бројачку променљиву петље користили смо баш члан *duzina* тако да је, по изласку из петље, вредност у њему коректна и представља број цифара великог броја.

Следе два помоћна метода, *obrisiVodeceNule()* и *obrniCifre()*, које користи статички метод *ucitajBroj()* за учитавање великог броја са стандардног улаза. Метод *ucitajBroj()* најпре учитава *String* са стандардног улаза, а затим издваја из њега велики број. Унети *String* се обрађује почев од првог карактера, а стаје се када се наиђе на први карактер који није цифра, успешно се обради читав *String* или се премаши максималан број цифара. Када нам је познато да се у неком карактеру налази запис цифре (што проверавамо статичким методом *isDigit()* класе *Character*), њену вредност добијамо одузимањем кода нуле, '0', од кода цифре, јер су кодови цифара у UNICODE-у поређани узастопно почев од нуле. Након што смо очитали цифре броја и сместили их у низ, неопходно је тај низ обрнути, јер смо цифре смештали у низ редоследом како смо их читали, тј. почев од цифре највеће тежине. Потом се, методом *obrisiVodeceNule()*, бришу евентуалне водеће нуле уз ажурирање броја цифара.

Метод *toString()* гради *String*-репрезентацију великог броја тако што дописује његове цифре слева надесно почев од цифре највеће тежине.

Метод *compareTo()* пореди текући и задати велики број. Ако бројеви имају различит број цифара, мањи је онај који има мањи број цифара. У случају да бројеви имају једнак број цифара, утврђивање односа међу њима врши се на следећи начин: пореди се цифра по цифра, почев од цифре највеће тежине. Први пар цифара на коме дође до неслагања одређује однос бројева. Бројеви су једнаки ако имају једнак број цифара и сви одговарајући парови цифара су једнаки.

Метод *equals()* само позива *compareTo()* да утврди да ли је текући велики број једнак задатом.

Метод *saberi()* сабира текући и задати велики број и враћа њихов збир. Сабирање се врши симулирањем поступка "ручног" сабирања: бројеви се потпишу један испод другог, а затим се сабира цифра по цифра, почев од цифре најмање тежине, све док се не обраде све цифре. Води се рачуна и о преносу на следећу позицију, који је у првом кораку 0. Када се исцрпе све цифре краћег броја, поступак се наставља, при чему се као њихова вредност узима вредност 0. У сваком кораку, саберу се одговарајуће цифре сабирака и пренос са претходне позиције. Одговарајућа цифра резултата је цифра најмање тежине тако добијеног збира, док се пренос на наредну позицију одређује као целобројни количник тог збира и броја 10. Ако, по завршетку овог поступка, постоји пренос, он се уписује као цифра највеће тежине резултата, уз ажурирање (инкрементирање) броја цифара. Иначе, дужина резултата једнака је дужини дужег од сабирака.

Множење текућег и задатог великог броја, имплементирано методом *pomnozi()*, такође је симулација "ручног" множења: узима се цифра по цифра множиоца, почев од цифре најмање тежине, множеник се множи текућом цифром множиоца и добијени међупроизводи се записују степенасто, сваки следећи померен за једно место улево. Резултат се добија сабирањем свих добијених међупроизвода. Из описа овог поступка видимо да се намеће потреба за множењем великог броја цифром, као и за "померањем улево" за одговарајући број места. Ове операције имплементиране су помоћним методима *pomnoziCifrom()* и *pomnoziSa10naK()* (померање броја улево за k места еквивалентно је његовом множењу са 10^k). Велики број се множи цифром тако што се задатом цифром множи цифра по цифра великог броја, почев од цифре најмање тежине. Води се рачуна и о преносу на следећу позицију, који је у првом кораку 0. У сваком кораку на производ задате цифре и текуће цифре броја

додаје се пренос са претходне позиције и врши одређивање текуће цифре резултата и преноса на наредну позицију на исти начин као код сабирања (као целобројни остатак и количник при дељењу добијеног збира бројем 10, респективно). Евентуални пренос са места највеће тежине уписује се као цифра највеће тежине резултата, уз ажурирање (инкрементирање) броја цифара. Множење броја са 10^k врши се дописивањем k нула на крај броја. Дакле, број цифара резултата је за k већи од полазног броја. Најпре се цифре броја, почев од цифре највеће тежине, препишу у низу на позиције за по k веће од полазних, а затим се на упражњених k места на почетку низа упишу нуле.

Статичка верзија метода *faktorijel()* рачуна факторијел датог ненегативног целог броја *n* и враћа резултат као велики број. Метод рачуна факторијел итеративно. Креће се од "великог" броја 1, који се затим у петљи узастопно множи "великим" бројевима 2, 3, ..., *n*.

За рекурзивну варијанту метода *faktorijel()*, која рачуна и враћа факторијел текућег великог броја, неопходна нам је операција декрементирања, која је имплементирана помоћним методом *dekrementiraj()*. Ако је цифра најмање тежине броја различита од 0, цифра најмање тежине декрементираног броја је за један мања од те цифре, док су све остале цифре једнаке одговарајућим цифрама полазног броја. Иначе, цифра најмање тежине декрементираног броја је 9 и имамо позајмицу на наредној позицији. Обрађују се и преостале цифре. Од текуће цифре броја покуша се одузимање позајмице. Ако би се као резултат тог одузимања добио негативан број, одговарајућа цифра резултата једнака је 9, и имамо позајмицу на наредној позицији, а иначе, цифра резултата једнака је разлици текуће цифре и позајмице и немамо позајмицу на наредној позицији, тј. позајмица има вредност 0. Декрементирани број је или исте дужине као и полазни, или за један краћи. Када имамо декрементирање, факторијел великог броја се рачуна рекурзивно на уобичајен начин, с тим да је потребно користити методе за баратање великим бројевима уместо стандардних оператора програмског језика Јава: метод *equals()* за поређење на једнакост (са "великом" нулом), *dekrementiraj()* за декрементирање великог броја и *pomnozi()* за множење два велика броја.

Рекурзивни метод *fibonaciR()* очекује као аргумент индекс Фибоначијевог низа, а враћа одговарајући елемент низа као "велики" број. Метод има потпуно исту структуру као и метод из задатка ???, с тим да се за сабирање великих бројева не може користити стандардни оператор +, већ се то чини одговарајућим методом класе *Broj*.

Метод *fibonaci()* рачуна одговарајући елемент Фибоначијевог низа итеративно уз коришћење технике познате као *динамичко програмирање*. Коришћењем низа уместо рекурзивних позива скраћује се време извршавања. Наиме, рекурзија "не памти" вредности које је претходно израчунала, па тако нпр. ако имамо позив *fibonaciR(5)*, он ће условити позиве *fibonaciR(4)* и *fibonaciR(3)*. Они ће, даље, звати *fibonaciR(3)*, *fibonaciR(2)* и *fibonaciR(2)*, *fibonaciR(1)*, редом. Коначно, *fibonaciR(3)* ће још по једном позвати *fibonaciR(2)* и *fibonaciR(1)*. Дакле, за рачунање 5. елемента Фибоначијевог низа, 2 пута се рачуна *fibonaciR(1)*, 3 пута *fibonaciR(2)*, 2 пута *fibonaciR(3)* и једном *fibonaciR(4)*. За већи индекс, већи је и број поновљених рачунања већ израчунатих вредности елемената са мањим индексима што знатно успорава читав процес. У методу *fibonaci()*, који користи приступ динамичког програмирања, нема редундантног рачуна: све израчунате вредности се памте у низу, а рачунање се врши почев од првог елемента, тако да, када је потребно израчунати произвољан елемент, у низу се већ налазе сви са индексима мањим од његовог, укључујући и 2 претходна помоћу којих се рачуна његова вредност, те је само потребно приступити им и сабрати их. У тест-класи се, поред вредности одговарајућег елемента Фибоначијевог низа, исписује и време извршавања позива функције којим је та вредност израчуната.

Динамичко програмирање

Основна идеја динамичког програмирања је да се решавање датог проблема сведе на решавање низа потпроблема. Низ потпроблема треба да буде такав да је сваки наредни потпроблем могуће решити комбиновањем решења једног или већег броја већ решених потпроблема. Динамичко програмирање користи приступ одоздо нагоре. Међурешења се чувају у низу како би се избегло понављање већ урађеног посла.

Решење:

Broj.java

```
package velikiBrojevi;

import java.util.Scanner;

public class Broj implements Comparable<Broj>
{
   public static final int MAX_CIFRE = 2000;

   private int[] cifre;
   private int duzina;

   private static Scanner sc = new Scanner(System.in);

   public Broj(int n)
   {
      // 0 je specijalan slucaj
      if (n == 0)
      {
         duzina = 1;
         cifre = new int[] { 0 };
         return;
      }

      cifre = new int[MAX_CIFRE];

      for (duzina = 0; n != 0; duzina++)
      {
         cifre[duzina] = n % 10;
         n /= 10;
      }

   }

   public Broj(int[] cifre, int duzina)
   {
      this.cifre = cifre;
      this.duzina = duzina;
   }

   private void obrisiVodeceNule()
   {
      /*
       * brisu se sve vodece nule, osim u slucaju da je broj sastavljen od svih
       * nula, tada se ostavlja samo jedna
       */
      while (duzina > 1 && cifre[duzina - 1] == 0)
         duzina--;
   }

   private void obrniCifre()
   {
      for (int i = 0, j = duzina - 1; i < j; i++, j--)
      {
         int pom = cifre[i];
         cifre[i] = cifre[j];
         cifre[j] = pom;
      }
   }

   public static Broj ucitajBroj()
```

```
  {
     int duzina;
     int[] cifre = new int[MAX_CIFRE];
     String unos = sc.next();

     for (duzina = 0; duzina < MAX_CIFRE && duzina < unos.length()
         && Character.isDigit(unos.charAt(duzina)); duzina++)
       cifre[duzina] = unos.charAt(duzina) - '0';

     Broj broj = new Broj(cifre, duzina);
     broj.obrniCifre();
     broj.obrisiVodeceNule();

     return broj;
  }

  public String toString()
  {
     StringBuffer rez = new StringBuffer();
     for (int i = duzina - 1; i >= 0; i--)
       rez.append(cifre[i]);
     return rez.toString();
  }

  public int compareTo(Broj b)
  {

     // Uporedjujemo duzine brojeva
     if (duzina > b.duzina)
       return 1;
     if (duzina < b.duzina)
       return -1;

     /*
      * U ovom trenutku znamo da su brojevi iste duzine, tako da prelazimo na
      * poredjenje cifra po cifra, pocevsi od cifre najvece tezine
      */
     for (int i = duzina - 1; i >= 0; i--)
     {
       if (cifre[i] > b.cifre[i])
         return 1;
       if (cifre[i] < b.cifre[i])
         return -1;
     }
     return 0;
  }

  public boolean equals(Object b)
  {
     return compareTo((Broj) b) == 0;
  }

  public Broj saberi(final Broj b)
  {
     int prenos = 0; // prenos sa prethodne pozicije
     int[] cifreRezultata = new int[MAX_CIFRE];
     int duzinaRezultata;

     for (duzinaRezultata = 0; duzinaRezultata < duzina
         || duzinaRezultata < b.duzina; duzinaRezultata++)
     {
       int cifraRezultata =
((duzinaRezultata < duzina) ? cifre[duzinaRezultata]: 0)
           + ((duzinaRezultata < b.duzina) ? b.cifre[duzinaRezultata] : 0)
           + prenos;

       cifreRezultata[duzinaRezultata] = cifraRezultata % 10;
```

```
      prenos = cifraRezultata / 10;
   }

   if (prenos != 0 && duzinaRezultata < MAX_CIFRE)
   {
      cifreRezultata[duzinaRezultata] = prenos;
      duzinaRezultata++;
   }

   return new Broj(cifreRezultata, duzinaRezultata);
}

private Broj pomnoziCifrom(int cifra)
{
   int prenos = 0;
   int duzinaRezultata;

   // Ako je cifra 0, rezultat je 0
   if (cifra == 0)
      return new Broj(0);

   int[] cifreRezultata = new int[MAX_CIFRE];
   for (duzinaRezultata = 0; duzinaRezultata < duzina; duzinaRezultata++)
   {
      int pom = cifre[duzinaRezultata] * cifra + prenos;

      cifreRezultata[duzinaRezultata] = pom % 10;
      prenos = pom / 10;
   }

   if (prenos != 0 && duzinaRezultata < MAX_CIFRE)
   {
      cifreRezultata[duzinaRezultata] = prenos;
      duzinaRezultata++;
   }

   return new Broj(cifreRezultata, duzinaRezultata);
}

/*
 * metod mnozi broj sa 10^k i menja tekuci broj rezultatom.
 * Koristi se za pojednostavljenje mnozenja.
 */
private void pomnoziSa10naK(int k)
{
   /*
    * nema svrhe nulu mnoziti sa 10^k jer bi to dalo kao rezultat niz od k+1
    * nule
    */
   if (duzina == 1 && cifre[0] == 0)
      return;

   duzina += k;

   for (int i = duzina - 1; i >= 0; i--)
      cifre[i] = i < k ? 0 : cifre[i - k];
}

public Broj pomnozi(final Broj b)
{

   /*
    * Broj pom ce sadrzati rezultat mnozenja tekuceg broja jednom po jednom
    * cifrom broja b, dok ce se na broj rezultat dodavati svaki put pom*(10^i)
    */

   Broj rezultat = new Broj(0);
```

```
   for (int i = 0; i < b.duzina; i++)
   {
      Broj pom = pomnoziCifrom(b.cifre[i]);
      pom.pomnoziSa10naK(i);
      rezultat = rezultat.saberi(pom);
   }

   return rezultat;

}

// metod izracunava faktorijel nenegativnog broja n kao veliki broj
public static Broj faktorijel(int n)
{
   Broj rezultat = new Broj(1);

   for (int i = 2; i <= n; i++)
   {
      Broj brojI = new Broj(i);
      rezultat = rezultat.pomnozi(brojI);
   }

   return rezultat;
}

private Broj dekrementiraj()
{
   int pozajmica = 0;
   int[] cifreRezultata = new int[MAX_CIFRE];

   if (cifre[0] == 0)
   {
      cifreRezultata[0] = 9;
      pozajmica = 1;

      for (int i = 1; i < duzina; i++)
      {
         if (cifre[i] - pozajmica < 0)
         {
            pozajmica = 1;
            cifreRezultata[i] = 9;
         } else
         {
            cifreRezultata[i] = cifre[i] - pozajmica;
            pozajmica = 0;
         }
      }

      int duzinaRezultata = cifreRezultata[duzina - 1] == 0 ? duzina - 1
          : duzina;

      return new Broj(cifreRezultata, duzinaRezultata);
   } else
   {
      cifreRezultata[0] = cifre[0] - 1;
      for (int i = 1; i < duzina; i++)
         cifreRezultata[i] = cifre[i];
      return new Broj(cifreRezultata, duzina);
   }
}

// metod racuna faktorijel tekuceg velikog broja
public Broj faktorijel()
{
   if (equals(new Broj(0)))
   {
```

```
         return new Broj(1);
      }

      Broj manje1 = dekrementiraj();

      return pomnozi(manje1.faktorijel());
   }

   // metod racuna n-ti clan Fibonacijevog niza, rekurzivna varijanta
   public static Broj fibonaciR(int n)
   {
      if (n == 1 || n == 2)
         return new Broj(1);

      return fibonaciR(n - 1).saberi(fibonaciR(n - 2));
   }

   /*
    * metod racuna n-ti clan Fibonacijevog niza, iterativna varijanta,
    * dinamicko programiranje
    */
   public static Broj fibonaci(int n)
   {
      Broj[] rez = new Broj[n];

      rez[0] = new Broj(1);

      if (n >= 2)
         rez[1] = new Broj(1);

      for (int i = 3; i <= n; i++)
         rez[i - 1] = rez[i - 2].saberi(rez[i - 3]);

      return rez[n - 1];

   }

}
```

TestVelikiBrojevi.java

```
package velikiBrojevi;

import java.util.Scanner;

public class TestVelikiBrojevi
{

   public static void main(String[] args)
   {

      Scanner sc = new Scanner(System.in);

      // Unos velikih brojeva
      System.out.print("Unesite prvi broj: ");
      Broj a = Broj.ucitajBroj();

      System.out.print("Unesite drugi broj: ");
      Broj b = Broj.ucitajBroj();

      // Prikaz velikih brojeva
      System.out.println("a je: " + a);
      System.out.println("b je: " + b);

      // Sabiraju se i ispisuje se zbir
      System.out.println("Zbir je: " + a.saberi(b));
```

```
        // 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 {
}
}
```

Постоји разлика у својствима угњеждених класа које су дефинисане са или без клаузуле *static.*

Нестатичке угњеждене класе имају важност само у контексту објекта спољашње класе, тј. не може се креирати објекат угњеждене класе без претходног креирања објекта спољашње класе.

```
        // Kreira se objekat klase Spoljasnja
Spoljasnja sp = new Spoljasnja();
/* Kreira se objekat klase Unutrasnja, pri cemu je tip ugnjezdene
 * klase kvalifikovan imenom spoljasnje klase. Naredba sp.new govori
 * da je objekat ugnjezdene klase u vezi sa objektom sp i ne moze da
 * postoji nezavisno.
 */
Spoljasnja.Unutrasnja un = sp.new Unutrasnja();
```

Статички методи спољашње класе не могу да креирају објекте нестатичке угњеждене класе.

Нестатичке угњеждене класе не могу да садрже статичке чланице, зато што не могу да се понашају као независне класе.

Статичке угњеждене класе имају својство да њихови објекти могу бити креирани независно од објеката спољашње класе, дакле, понашају се као самосталне класе. Стога, оне могу да садрже статичке чланице.

У решењу је класа *Zec* дефинисана као нестатичка угњеждена класа класе *MagicniSesir*.
Инстанцне чланице класе *MagicniSesir* су *imeSesira*, типа *String* и низ елемената *zecevi*, типа *Zec*, који представља низ зечева у шеширу.
Поред њих, класа садржи и статичке чланице:

- *int maksimumZeceva* - максимални дозвољени број зечева у шеширу
- *Random izbor* – генератор случајних бројева
- *String[] imenaZeceva* – низ дозвољених имена зечева
- *int[] brojImenaZeceva* – *i*-ти елемент овог низа одговара *i*-том елементу низа *imenaZeceva* и представља број зечева који се тако зову. Потребно је због генерисања јединствених имена зечева.

Напомена:
Последње две статичке чланице могу бити део класе *Zec*, јер представљају својства која се односе на зечеве, а не на магични шешир. То је могуће само у случају кад је класа *Zec* дефинисана као статичка угњеждена класа.
Методи класе *MagicniSesir*:

- Конструктор са једним аргументом типа *String*, на основу којег се иницијализује инстанцна променљива *nazivSesira*.
  Креира се низ *zecevi* тако да број елемената низа буде случајно изабран из интервала [*1, maksimumZeceva*]. То се постиже позивом метода *nextInt()* генератора случајних бројева *izbor* са аргументом *maksimumZeceva*. Метод генерише случајан цео број из интервала [*0, maksimumZeceva-1*], а након увећања за *1*, добија се број из интервала [*1, maksimumZecva*].
  ```
  zecevi = new Zec[1 + izbor.nextInt(maksimumZeceva)];
  ```
  Потом се креирају елементи овог низа, позивом конструктора *Zec()* класе *Zec*.
- Метод *toString()* генерише *String*-репрезентацију магичног шешира. Њу чине назив шешира и имена зечева који се у њему налазе.

Класа *Zec* има једну инстанцну променљиву, *ime*, типа *String*, која представља име зеца.
Методи класе *Zec*:

- Конструктор без аргумената у којем се случајно бира јединствено име зеца и том вредношћу се иницијализује инстанцна променљива. Пошто је чланица *izbor* у класи *MagicniSesir* декларисана као статичка, може јој се приступити из нестатичке угњеждене класе *Zec*. Помоћу ње се случајно бира име из низа дозвољених имена зечева. Изабраном имену додаје се цео број који се добија увећањем за један броја до сада креираних зечева са тим именом. Тиме се добија јединствено име зеца.
- Метод *toString()* за генерисање *String*-репрезентације зеца коју чини име зеца (алтернативно, име зеца праћено називом шешира у коме се зец налази).

У методу *main()* класе *TestMagicniSesir* креирају се две инстанце класе *MagicniSesir* („Crni kaubojski" и „Crni cilindar") и исписују се подаци о њима на стандардни излаз.
Потом се креира још једна инстанца класе *MagicniSesir* („Sombrero").
Креира се и објекат угњеждене класе *Zec*, с тим што не постоји могућност да то буде самостални објекат. Дакле, неопходан је објекат класе *MagicniSesir* у чијем контексту ће бити креиран објекат класе *Zec*. Нека је то управо креиран шешир *sombrero*. Име угњеждене класе мора бити квалификовано именом спољашње класе.

```
MagicniSesir.Zec zeka = sombrero.new Zec();
```

Напомена:
Веза која је успостављена између објеката *sombrero* и *zeka* не значи да објекат *sombrero* сада у низу зечева садржи и референцу на објекат *zeka*, већ да, ако се у унутрашњој класи користе чланице спољашње класе, у случају објекта *zeka*, то ће бити управо чланице објекта *sombrero*.
Потом се исписују подаци за шешир *sombrero* на стандардни излаз, чиме се може уочити да заиста зец *zeka* није у низу зечева овог шешира.
Затим се исписују и подаци за објекат *zeka*. Ако се поред имена зеца исписује и назив шешира коме зец припада, у овом случају ће бити исписано „Sombrero".

Решење:

MagicniSesir.java

```
package ZecIzSesira;

import java.util.Random;

public class MagicniSesir
{
   // Maksimalni broj zeceva u sesiru
   static int maksimumZeceva = 5;

   // Generator slucajnih brojeva
   static Random izbor = new Random();

   // Niz mogucih imenima zeceva
   static private String[] imenaZeceva =
      { "Lupko", "Traputalo", "Sarenko", "Garavko", "Dugousko" };

   // Koliko imamo zeceva svakog od imena;
   static private int[] brojImenaZeceva = new int[imenaZeceva.length];

   // Ime sesira
   private String nazivSesira;
   // Niz zeceva koji su u sesiru
   private Zec zecevi[];

   // Konstruktor sesira na osnovu njegovog naziva
   public MagicniSesir(final String nazivSesira)
   {
      // Postavlja se naziv sesira na zadati
      this.nazivSesira = nazivSesira;
```

```
      /*  Kreira se niz zeceva sa brojem elemenata
       *  iz intervala [1, maksimumZeceva]
       */
      zecevi = new Zec[1 + izbor.nextInt(maksimumZeceva)];

      // Kreiraju se objekti klase Zec
      for (int i = 0; i < zecevi.length; i++)
         zecevi[i] = new Zec();
   }

   // String-reprezentacija sesira
   public String toString()
   {
      String sesirString = "\n" + nazivSesira + " sadrzi:\n";
      for (int i = 0; i < zecevi.length; i++)
         sesirString += "\t" + zecevi[i] + " ";
      return sesirString;
   }

   // Ugnjezdena nestaticka klasa Zec
   class Zec
   {
      // Ime je jedno od mogucih imena iz niza imenaZeceva
      // za kojim sledi jedan ceo broj
      private String ime;

      // Konstruktor klase Zec
      public Zec()
      {
         // slucajno se bira ime iz niza dozvoljenih imena
         int indeks = izbor.nextInt(imenaZeceva.length);
         // ime prati jedan ceo broj
         ime = imenaZeceva[indeks] + (++brojImenaZeceva[indeks]);
      }

      // String-reprezentacija zeca
      public String toString()
      {
         return ime;
         // return ime + " sesir: " + nazivSesira;
      }
   }
}
```

TestMagicniSesir.java

```
package ZecIzSesira;

public class TestMagicniSesir
{
   static public void main(String[] args)
   {
      // Kreiraju se dve instance klase MagicniSesir i ispisuju se na izlaz
      System.out.println(new MagicniSesir("Crni kaubojski"));
      System.out.println(new MagicniSesir("Crni cilindar"));

      // Nestaticka implementacija klase Zec

      // Kreira se nova instanca klase MagicniSesir
      MagicniSesir sombrero = new MagicniSesir("Sombrero");

      MagicniSesir.Zec zeka = sombrero.new Zec();

      // Ispisuju se podaci o sesiru na standardni izlaz
      System.out.println(sombrero);
      // Ispisuju se i podaci o zecu
      System.out.println("\nNovi zec je: " + zeka);
```

```
    }
}
```

16. *Повезана листа, Многоугао*. Написати генеричку класу *PovezanaLista<T>* којом се описује двоструко повезана листа објеката произвољног типа *T*. Појединачни елементи листе дефинисани су класом *ElementListe*. Сваки елемент листе има свој садржај који је типа *T*. Дефинисати конструктор и метод за генерисање *String*-репрезентације елемента листе. У класи *PovezanaLista* дефинисати следеће методе:

- подразумевани конструктор који креира празну листу
- конструктор који креира листу од једног елемента
- конструктор који креира листу на основу низа објеката типа *T*
- Метод који враћа елемент листе са датим индексом у листи
- Метод који враћа садржај елемента са датим индексом у листи
- Метод за додавање елемента на крај листе, при чему је елемент задат својим садржајем
- Метод за додавање низа елемената на крај листе, при чему су елементи задати својим садржајима
- Метод који враћа садржај првог елемента листе
- Метод који враћа садржај последњег елемента листе
- Метод toString()

Модификовати класу *Mnogougao* из претходног задатка, тако да уместо низа темена садржи као инстанцну промењљиву повезану листу темена.
Написати и тест класу сличну тест класи из претходног задатка.

Објашњење:

Генерички (параметризовани) тип је класни или интерфејсни тип који има један или више параметара.

```
public class PovezanaLista<T> { ... }
```

Параметри се наводе између < > заграда. Генерички типови омогућавају дефинисање једне класе којом је представљена колекција објеката типа *T*.
Дефиниција конкретне класе или интерфејса из датог генеричког типа врши се навођењем конкретног типа уместо параметра *T*. Сва појављивања параметра *T* у дефиницији генеричког типа биће замењена датим типом.
Генерички тип, заправо, дефинише скуп типова, добијених за различите вредности параметра *T*. Конкретан тип који се наводи уместо параметра *T* може бити само класни или интерфејсни тип. Уместо примитивних типова користе се одговарајуће класе које их енкапсулирају.

Класа *ElementListe* дефинише појединачни елемент двоструко повезане листе. Садржи три инстанцне промењљиве: *sadrzaj*, типа *T* и референце *sledeci* и *prethodni*, типа *ElementListe*, на наредни, односно, претходни елемент повезане листе. Дефинисана је као статичка угњеждена класа класе *PovezanaLista*, те је неопходно наводити параметар иза имена класе у заградама, јер она може да се понаша као самостална класа.
Садржи следеће методе:

a) Конструктор са три аргумента којим се иницијализују инстанцне промењљиве
b) Метод *toString()*

Двоструко повезана листа представљена је главом листе и бројем елемената у листи.
Класа *PovezanaLista* има инстанцну промењљиву која је референца на главу листе. Глава листе је елемент листе чији је садржај једнак *null*. Иницијално се поставља да су и референце *sledeci* и *prethodni* једнаке *null*.

Друга инстанцна променљива класе *PovezanaLista* је величина листе, тј. број елемената и иницијално има вредност 0.
Методи:

a) Подразумевани конструктор креира празну листу. Референце *prethodni* и *sledeci* главе листе реферишу управо на главу листе када је листа празна.
b) Конструктор са једним аргументом типа *T* креира листу од једног елемента. Ако дати садржај типа *T* није *null*, креира се нови елемент листе са датим садржајем, а његове референце *sledeci* и *prethodni* реферишу на главу листе. Такође, референце *sledeci* и *prethodni* главе листе реферишу на креирани елемент.
c) Конструктор који креира листу на основу датог низа објеката типа *T*. Позива се претходни конструктор наредбом *this()*, а потом метод *dodajSve(),* који додаје на крај листе нове елементе чији су садржаји редом објекти датог низа.
d) Метод *vrati()* враћа садржај елемента са датим индексом у листи наредбом
```
return element(indeks).sadrzaj;
```
e) Метод *element()* враћа елемент листе са датим индексом у листи. Уколико индекс није у дозвољеним границама, избацује се изузетак типа *IndexOutOfBoundsException*. Претраживање листе почиње од главе и наставља се у једном или другом смеру кроз листу, у зависности од индекса елемента који се тражи. Ако је индекс у интервалу [0, *velicina*/2 – 1], претрага се врши од почетка ка средини листе, иначе, од краја ка средини.
f) Метод *dodajElement()* додаје елемент задат својим садржајем на крај листе. Додавање се врши позивом метода *dodajPre()* који додаје нови елемент на позицију пре главе листе, што је управо крај листе.
g) Метод *dodajPre()* додаје нови елемент са датим садржајем типа *T* непосредно пре датог елемента *e*. Најпре се креира нови елемент на основу датог садржаја. Његов следбеник у листи постаје управо елемент *e*, а претходник елемент који тренутно претходи елементу *e*. Нови елемент постаје нови претходник елемента *e* и нови следбеник елемента који је претходио елементу *e*. Повратна вредност метода је референца на додати елемент.
h) Метод *dodajSve()* додаје низ нових елемената у листу почев од позиције *indeks*. Поредак елемената при додавању одговара поретку у низу, а елементи који су били на тим позицијама у листи се померају удесно. Најмања дозвољена вредност за *indeks* је 0, када се нови елементи додају на почетак листе, а највећа је *vrednost*, када се елементи додају на крај, тј. непосредно пре главе листе. Ако је *indeks* ван овог опсега, избацује се изузетак типа *IndexOutOfBoundsException*. У случају да је дати низ објеката типа *T* празан, метод враћа *false*.
   Променљиве *sledbenik* и *prethodnik* представљају следбеника и претходника текућег елемента који се додаје.
   Ако је *indeks* једнак величини листе, текући следбеник је глава листе, иначе елемент са датим индексом. Текући претходник је претходник текућег следбеника. Сваки нови елемент који се додаје има истог следбеника.
   У петљи се креира један по један нови елемент, а следбеник и претходник су му текући следбеник и текући претходник. Потом, следбеник текућег претходника постаје нови елемент, а претходник наредног елемента који треба да се дода је управо додати елемент. Након завршетка петље, остаје још да последњи додати елемент постане претходник следбеника свих додатих објеката.
   На крају се увећава величина листе за број додатих елемената и враћа *true*.
i) Метод *vratiPrvi()* враћа садржај првог елемента листе. Ако је листа празна, избацује се изузатак типа *NoSuchElementException*, иначе се враћа садржај следбеника главе, што јесте први елемент.
j) Метод *vratiPoslednji()* враћа садржај последњег елемента, тј. елемента који претходи глави листе. И овај метод избацује изузетак истог типа као и претходни, ако је листа празна.
k) Метод *velicina()* враћа величину листе
l) Метод *toString()* генерише *String*-репрезентацију надовезивањем *String*-reprezentacija елемената листе.

У класи *Mnogougao* инстанцна променљива *temena* је сада типа *PovezanaLista<Tacka>*.
Модификовани су следећи методи:

a) Конструктор на основу датог дводимензионог низа координата темена. Након креирања низа темена, креира се повезана листа темена на основу добијеног низа.
b) Конструктор на основу датог низа темена. Само се позива конструктор класе *PovezanaLista*.
c) У методима *stranice(), strIdijagTemeA1()* и *konveksan(),* приступ *i*-том темену се сада врши позивом метода *temena.vrati(i)*
d) У методу *toString()* се *String*-репрезентација темена добија директно, јер је дефинисан метод *toString()* у класи *PovezanaLista*

Додати су методи:

- *dodajTeme(Tacka)*, који додаје дато теме на крај повезане листе темена, позивом метода *dodajElement()* над листом темена.
- *dodajTeme(double, double),* који додаје теме задато својим координатама на крај листе темена.
- *dodajTemena(Tacka[]),* који додаје низ темена на крај повезане листе, позивом метода *dodajSve()* над листом темена

Класа *TestMnogougao* из претходног задатка је модификована тако што се још врши и тестирање додавања новог темена многоуглу и посматра како се то рефлектује на његову конвексност. Координате темена које се додаје уносе се са стандардног улаза.

Решење:

Tacka.java

```
package povezanaLista;

public class Tacka
{
   //Instancne promenljive - x i y koordinata tacke
   private double x;
   private double y;

   //Podrazumevani konstruktor postavlja tacku u koordinatni pocetak
   public Tacka()
   {}

   // Konstruktor sa datim vrednostima za koordinate tacke
   public Tacka(double x, double y)
   {
      this.x = x;
      this.y = y;
   }

   // Konstruktor kopija
   public Tacka(final Tacka tacka)
   {
      this(tacka.x, tacka.y); // x = tacka.x; y = tacka.y;
   }

   // Metod za racunanje rastojanja do zadate tacke
   public double rastojanje(final Tacka tacka)
   {
      return Math.sqrt( (x - tacka.x) * (x - tacka.x)
                + (y - tacka.y) * (y - tacka.y));
   }

   public double getX()
   {
      return x;
   }

   public double getY()
   {
```

```
      return y;
   }

   public void setX(double x)
   {
      this.x = x;
   }

   public void setY(double y)
   {
      this.y = y;
   }

   public String toString()
   {
      return "(" + x + ", " + y + ")";
   }
}
```

PovezanaLista.java

```
package povezanaLista;

import java.util.NoSuchElementException;

public class PovezanaLista<T>
{
     private ElementListe<T> glava =
           new ElementListe<T>(null, null, null);   // Glava liste
     private int velicina = 0;

     // Podrazumevani konstruktor - kreira praznu listu
     public PovezanaLista()
     {
      glava.sledeci = glava.prethodni = glava;
     }

     // Konstruktor koji kreira listu od jednog elementa
     public PovezanaLista(T sadrzaj)
     {
        if(sadrzaj != null)
        {
         ElementListe<T> novi = new ElementListe<T>(sadrzaj, glava, glava);
         glava.sledeci = glava.prethodni = novi;
        }
      }

     // Konstruktor liste na osnovu datog niza objekata tipa T
     public PovezanaLista(T[] elementi)
     {
      this();
        dodajSve(elementi);
     }

     public T vrati(int indeks)
     {
      return element(indeks).sadrzaj;
     }

     public ElementListe<T> element(int indeks) throws IndexOutOfBoundsException
     {
      if(indeks < 0 || indeks >= velicina)
         throw new IndexOutOfBoundsException("Indeks: " + indeks +
                                             ", Velicina: " + velicina);
      ElementListe<T> tekuci = glava;
      if(indeks < (velicina >> 1))
```

```
{
   for(int i = 0; i <= indeks; i++)
      tekuci = tekuci.sledeci;
}
else
{
   for(int i = velicina; i > indeks; i--)
      tekuci = tekuci.prethodni;
}
return tekuci;
}

// Dodaje novi element na kraj liste
public boolean dodajElement(T sadrzaj)
{
dodajPre(sadrzaj, glava);
return true;
}

public ElementListe<T> dodajPre(T sadrzaj, ElementListe<T> e)
{
ElementListe<T> noviEl = new ElementListe<T>(sadrzaj, e, e.prethodni);
noviEl.sledeci.prethodni = noviEl;
noviEl.prethodni.sledeci = noviEl;
velicina++;
return noviEl;
}

public boolean dodajSve(T[] noviElementi)
{
return dodajSve(velicina, noviElementi);
}

public boolean dodajSve(int indeks, T[] noviElementi)
{
if (indeks < 0 || indeks > velicina)
   throw new IndexOutOfBoundsException("Indeks: " + indeks +
                                       ", Velicina: " + velicina);
int brojNovih = noviElementi.length;
if(brojNovih == 0)
   return false;

ElementListe<T> sledbenik = (indeks == velicina ?
                                  glava : element(indeks));
ElementListe<T> prethodnik = sledbenik.prethodni;
for (int i = 0; i < brojNovih; i++)
{
   ElementListe<T> el = new ElementListe<T>(
          (T)noviElementi[i], sledbenik, prethodnik);
   prethodnik.sledeci = el;
   prethodnik = el;
}
sledbenik.prethodni = prethodnik;

velicina += brojNovih;
return true;
}

// Vraca sadrzaj prvog elementa
public T vratiPrvi()
{
if (velicina == 0)
    throw new NoSuchElementException();
   return glava.sledeci.sadrzaj;
}

// Vraca sadrzaj poslednjeg elementa
```

```
    public T vratiPoslednji()
    {
     if (velicina == 0)
         throw new NoSuchElementException();
       return glava.prethodni.sadrzaj;
    }

    public int velicina()
    {
     return velicina;
    }

    public String toString()
    {
       StringBuffer str = new StringBuffer("");
       for(int i = 0; i < velicina; i++)
        str.append(element(i)+ " ");
       return str.toString();
    }

    // Ugnjezdena staticka klasa koja predstavlja element liste
    private static class ElementListe<T>
    {
       // Konstruktor
       public ElementListe(T sadrzaj, ElementListe<T> sledeci,
                                      ElementListe<T> prethodni)
       {
          this.sadrzaj = sadrzaj;
          this.sledeci = sledeci;
          this.prethodni = prethodni;
       }

       // String reprezentacija elementa liste
       public String toString()
       {
          return "" + sadrzaj ;
       }

       T sadrzaj;                // Sadrzaj elementa liste
       ElementListe<T> sledeci;  // Referenca na naredni element liste
       ElementListe<T> prethodni; // Referenca na prethodni element liste
    }
}
```

Mnogougao.java

```
package povezanaLista;

import java.util.Scanner;

public class Mnogougao
{
   private PovezanaLista<Tacka> temena;

   // Konstruktor na osnovu datog dvodimenzionalnog niza koordinata
   public Mnogougao(final double[][] koordinate)
   {
      Tacka[] tacke = new Tacka[koordinate.length];
      for(int i = 0; i < tacke.length; i++)
         tacke[i] = new Tacka(koordinate[i][0], koordinate[i][1]);

      temena = new PovezanaLista<Tacka>(tacke);
   }

   // Konstruktor na osnovu datog niza temena
   public Mnogougao(final Tacka[] tacke)
```

```
{
   temena = new PovezanaLista<Tacka>(tacke);
}

// Dodavanje temena na kraj liste temena
public boolean dodajTeme(final Tacka tacka)
{
   return temena.dodajElement(tacka);
}

// Dodavanje temena zadatog svojim koordinatama na kraj liste
public boolean dodajTeme(double x, double y)
{
   return temena.dodajElement(new Tacka(x, y));
}

public boolean dodajTemena(final Tacka[] novaTemena)
{
   return temena.dodajSve(novaTemena);
}

// Odredjuje duzine stranica mnogougla
public double[] stranice()
{
   int brojTemena = temena.velicina();
   double[] stranice = new double[brojTemena];
   for(int i = 0; i < brojTemena-1; i++)
           stranice[i] = temena.vrati(i).rastojanje(
                temena.vrati(i+1));
   stranice[brojTemena-1] =
        temena.vrati(brojTemena-1).rastojanje(
                temena.vrati(0));
   return stranice;
}

// Odredjuje broj dijagonala mnogougla
public int brojDijagonala()
{
   int velicina = temena.velicina();
   return velicina * (velicina - 3) / 2;
}

// Obim mnogougla
public double obim()
{
   double obim = 0;
   double[] stranice = stranice();
   for(int i = 0; i < stranice.length; i++)
      obim += stranice[i];
   return obim;
}

// Povrsina mnogougla
public double povrsina()
{
   double povrsina = 0;
   double[] stranice = stranice();
   if(Mnogougao.kvadrat(stranice))
      return stranice[0] * stranice[0];
   double[] strIdijag = strIdijagTemeA1();
   for(int i = 0 ; i < stranice.length-2; i++)
   {
      double a = strIdijag[i];
      double b = stranice[i+1];
      double c = strIdijag[i+1];
      double s = (a + b + c) / 2.0;
      povrsina += Math.sqrt(s*(s-a)*(s-b)*(s-c));
```

```
   }
   return povrsina;
}

// Duzine stranica i dijagonala iz prvog temena
public double[] strIdijagTemeA1()
{
   int velicina = temena.velicina();
   double[] strIdijag = new double[velicina-1];
   for(int i = 0; i < velicina-1; i++)
      strIdijag[i] = temena.vrati(0).rastojanje(
           temena.vrati(i+1));
   return strIdijag;
}

// Staticki metod za ispitivanje da li je mnogougao kvadrat
public static boolean kvadrat(double[] stranice)
{
   if(stranice.length != 4)
      return false;
   for(int i = 0; i < stranice.length-1; i++)
      if(stranice[i] != stranice[i+1])
         return false;
   return true;
}

// Provera konveksnosti
public boolean konveksan()
{
   int velicina = temena.velicina();
   boolean ind = true;
   for(int i = 0; i < velicina-1; i++)
   {
      // Koeficijenti A, B i C jednacine prave koju odredjuju
      // po dva uzastopna temena mnogogugla
      double A = temena.vrati(i+1).getY() - temena.vrati(i).getY();
      double B = temena.vrati(i).getX() - temena.vrati(i+1).getX();
      double C = temena.vrati(i+1).getX()*temena.vrati(i).getY() -
           temena.vrati(i).getX()*temena.vrati(i+1).getY();

      // Odradjuje se polozaj sledeceg temena u odnosu na pravu
      int indeks = (i+2) % velicina;
      ind = A*temena.vrati(indeks).getX() +
                B*temena.vrati(indeks).getY() + C > 0;
      // Mnogougao je konveksan ukoliko i preostala temena imaju isti polozaj u
      // odnosu na pravu
      for(int j = 0; j < velicina; j++)
         if(j != i && j != i+1 && j!= i+2 &&
            (A*temena.vrati(j).getX() + B*temena.vrati(j).getY() + C > 0) != ind)
              return false;
   }
   return true;
}

// String-reprezentacija mnogougla
public String toString()
{
   StringBuffer str = new StringBuffer("Temena mnogougla:\n");
   str.append(temena.toString());
   str.append("\nStranice mnogougla:\n");
   double[] stranice = stranice();
   for(int i = 0; i < stranice.length; i++)
      str.append(stranice[i] + " ");
   return str.toString();

}
```

```
   // Staticki metod koji ucitava mnogougao
   public static Mnogougao ucitajMnogougao(Scanner skener)
   {
      int brojTemena = skener.nextInt();
      Tacka[] temena = new Tacka[brojTemena];
      for(int i = 0; i < brojTemena; i++)
         temena[i] = new Tacka(skener.nextDouble(), skener.nextDouble());
      return new Mnogougao(temena);
   }

}
```

TestMnogougao.java

```
package povezanaLista;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.InputMismatchException;
import java.util.Scanner;

public class TestMnogougao
{
   public static void main(String[] args)
   {
      try
      {

         // Parser ulaza, tj. datoteke
         Scanner skener = new Scanner(new File("mnogougao.txt"));
         // Ucitavanje mnogougla
         Mnogougao mnogougao = Mnogougao.ucitajMnogougao(skener);
         skener.close();
         // Objekat za ispis u datoteku
         PrintWriter pisac = new PrintWriter(new FileWriter("izlaz.txt"));
         // Ispis mnogougla u datoteku
         pisac.println(mnogougao);
         // Ispis preostalih podataka o mnogouglu.
         pisac.println("Obim je: " + mnogougao.obim() +
              "\tPovrsina je: " + mnogougao.povrsina());
         if(mnogougao.konveksan())
         {
            pisac.println("Mnogougao je konveksan");
           if(Mnogougao.kvadrat(mnogougao.stranice()))
            pisac.println("Mnogougao je kvadrat");
         }
         else
            pisac.println("Mnogougao je konkavan");

         Tacka novoTeme;
         try
         {
              skener = new Scanner(System.in);
              System.out.print("Teme koje se dodaje: ");
              novoTeme = new Tacka(skener.nextDouble(), skener.nextDouble());
         }
         catch(NumberFormatException nfe)
         {
            nfe.printStackTrace();
            return;
         }

         mnogougao.dodajTeme(novoTeme);
```

```
            pisac.println(mnogougao);
            pisac.println("Obim je: " + mnogougao.obim() +
                 "\tPovrsina je: " + mnogougao.povrsina());

            if(mnogougao.konveksan())
            {
               pisac.println("Mnogougao je konveksan");
              if(Mnogougao.kvadrat(mnogougao.stranice()))
               pisac.println("Mnogougao je kvadrat");
            }
            else
               pisac.println("Mnogougao je konkavan");

            pisac.close();
         }
         // Izuzetak koji se izbacuje ako ne postoji ulazni fajl
         catch(FileNotFoundException e1)
         {
            System.out.println(e1.getMessage());
         }
         // Izuzetak koji se javlja usled nekorektno unesenih koordinata
         // temena mnogougla
         catch(InputMismatchException e2)
         {
            System.out.println("Nekorektno unesene koordinate temena.");
         }
        // Izuzetak koji se izbacuje usled problema pri ispisu
         catch(IOException e3)
         {
            System.out.println(e3.getMessage());
         }
         catch(IndexOutOfBoundsException e4)
         {
            e4.printStackTrace();
         }
      }
}
```

17. *LinkedList*, *Многоугао*. Дефинисати класу *Mnogougao* са истим својствима као у претходном задатку, само уместо класе *PovezanaLista*< >, користити класу *LinkedList*< > из пакета *java.util*. Тест-класа има исту форму као у претходном задатку.

Објашњење:

Генеричка класа *java.util.LinkedList* имплементира *java.util.List* интерфејс. Имплементиране су све операције над листом и допуштени су сви типови елемената (укључујући и *null*). Дефинише двоструко повезану листу која је представљена главом листе и бројем елемената у листи:

```
private Entry<E> header = new Entry<E>(null, null, null);
private int size = 0;
```

Појединачни елемент листе дефинисан је класом *Entry<E>* која је статичка угњеждена класа. Инстанцне чланице ове класе су:

```
E element;
Entry<E> next;
Entry<E> previous;
```

На исти начин је листа била дефинисана у класи *PovezanaLista*.

LinkedList<E>:

Конструктори:

- без аргумената, креира празну листу
- са аргументом *Collection<? extends E>*, креира *LinkedList<E>* објекат који садржи објекте колекције која се прослеђује као аргумент. Симбол ? у изразу *<? extends E>* означава било који тип који је изведен из типа E, или сам тип E.

Битнији методи:

- *add(E)* – додаје нови елемент са датим садржајем на крај листе
- *addAll(Collection<? extends E>* - додаје све елементе колекције на крај листе у поретку који важи у колекцији
- *addFirst(E)* – додаје елемент са датим садржајем на почетак листе
- *addLast(E)* – додаје елемент са датим садржајем на крај листе; има исти ефекат као и метод *add()*
- *get(int)* – враћа садржај елемента на датој позицији у листи
- *getFirst(), getLast()* – враћа садржај првог, односно, последњег елемента у листи
- *size()* – враћа број елемената у листи
- *removeFirst(), removeLast()* – уклања први, односно, последњи елемент из листе
- *remove(int)* - уклања из листе елемент на датој позицији
- *remove(Object)* – уклања прво појављивање датог елемента, ако такво постоји, док се елементи који следе померају у лево за једно место
- *entry(int)* – враћа елемент Entry<E> листе са датим индексом
- *addBefore(E, Entry<E>)* – додаје нови елемент са датим садржајем непосредно испред датог елемента

Објашњење:

Класа *Mnogougao* је незнатно модификована. На местима где су били позиви метода класе *PovezanaLista*, сада се налазе позиви одговарајућих метода класе *LinkedList*.
Оба конструктора имају потпуно другачију дефиницију. Креирање листе темена се сада врши постепеним додавањем једног по једног темена из датог низа.

Решење:

Tacka.java

```
package povezanaLista2;

public class Tacka
{
   //Instancne promenljive - x i y koordinata tacke
   private double x;
   private double y;

   //Podrazumevani konstruktor postavlja tacku u koordinatni pocetak
   public Tacka()
   {}

   // Konstruktor sa datim vrednostima za koordinate tacke
   public Tacka(double x, double y)
   {
      this.x = x;
      this.y = y;
   }

   // Konstruktor kopija
   public Tacka(final Tacka tacka)
   {
      this(tacka.x, tacka.y); // x = tacka.x; y = tacka.y;
```

```
   }

   // Metod za racunanje rastojanja do zadate tacke
   public double rastojanje(final Tacka tacka)
   {
      return Math.sqrt( (x - tacka.x) * (x - tacka.x)
                + (y - tacka.y) * (y - tacka.y));
   }

   public double getX()
   {
      return x;
   }

   public double getY()
   {
      return y;
   }

   public void setX(double x)
   {
      this.x = x;
   }

   public void setY(double y)
   {
      this.y = y;
   }

   public String toString()
   {
      return "(" + x + ", " + y + ")";
   }
}
```

Mnogougao.java

```
package povezanaLista2;

import java.util.LinkedList;
import java.util.Scanner;

public class Mnogougao
{
   private LinkedList<Tacka> temena;

   // Konstruktor na osnovu datog dvodimenzionalnog niza koordinata
   public Mnogougao(final double[][] koordinate)
   {
      for(double [] xy : koordinate)
         dodajTeme(new Tacka(xy[0], xy[1]));
   }

   // Konstruktor na osnovu datog niza temena
   public Mnogougao(final Tacka[] tacke)
   {
      for(Tacka tacka : tacke)
         dodajTeme(tacka);
   }

   // Dodavanje temena na kraj liste temena
   public void dodajTeme(Tacka tacka)
   {
      temena.add(tacka);
   }
```

```
   // Dodavanje temena zadatog svojim koordinatama na kraj liste
   public void dodajTeme(double x, double y)
   {
      temena.add(new Tacka(x, y));
   }

package povezanaLista2;

import java.util.LinkedList;
import java.util.Scanner;

public class Mnogougao
{
   private LinkedList<Tacka> temena;

   // Konstruktor na osnovu datog dvodimenzionalnog niza koordinata
   public Mnogougao(final double[][] koordinate)
   {
      for(double [] xy : koordinate)
         dodajTeme(new Tacka(xy[0], xy[1]));
   }

   // Konstruktor na osnovu datog niza temena
   public Mnogougao(final Tacka[] tacke)
   {
      for(Tacka tacka : tacke)
         dodajTeme(tacka);
   }

   // Dodavanje temena na kraj liste temena
   public void dodajTeme(Tacka tacka)
   {
      temena.add(tacka);
   }

   // Dodavanje temena zadatog svojim koordinatama na kraj liste
   public void dodajTeme(double x, double y)
   {
      temena.add(new Tacka(x, y));
   }

   // Odredjuje duzine stranica mnogougla
   public double[] stranice()
   {
      int brojTemena = temena.size();
      double[] stranice = new double[brojTemena];
      for(int i = 0; i < brojTemena; i++)
         stranice[i] = temena.get(i).rastojanje(temena.get(i+1));
      stranice[stranice.length-1] =
           temena.get(brojTemena-1).rastojanje(temena.get(0));
      return stranice;
   }

   // Odradjuje broj dijagonala mnogougla
   public int brojDijagonala()
   {
      return temena.size() * (temena.size() - 3) / 2;
   }

   // Obim mnogougla
   public double obim()
   {
      double obim = 0;
      double[] stranice = stranice();
      for(int i = 0; i < stranice.length; i++)
         obim += stranice[i];
      return obim;
```

```
}

// Povrsina mnogougla
public double povrsina()
{
   double povrsina = 0;
   double[] stranice = stranice();
   if(Mnogougao.kvadrat(stranice))
      return stranice[0] * stranice[0];
   double[] strIdijag = strIdijagTemeA1();
   for(int i = 0 ; i < stranice.length-2; i++)
   {
      double a = strIdijag[i];
      double b = stranice[i+1];
      double c = strIdijag[i+1];
      double s = (a + b + c) / 2.0;
      povrsina += Math.sqrt(s*(s-a)*(s-b)*(s-c));
   }
   return povrsina;
}

// Duzine stranica i dijagonala iz prvog temena
public double[] strIdijagTemeA1()
{
   double[] strIdijag = new double[temena.size()-1];
   for(int i = 0; i < temena.size()-1; i++)
      strIdijag[i] = temena.get(0).rastojanje(temena.get(i+1));
   return strIdijag;
}

// Staticki metod za ispitivanje da li je mnogougao kvadrat
public static boolean kvadrat(double[] stranice)
{
   if(stranice.length != 4)
      return false;
   for(int i = 0; i < stranice.length-1; i++)
      if(stranice[i] != stranice[i+1])
         return false;
   return true;
}

// Provera konveksnosti
public boolean konveksan()
{
   int velicina = temena.size();
   boolean ind = true;
   for(int i = 0; i < velicina-1; i++)
   {
      // Koeficijenti A, B i C jednacine prave koju odredjuju
      // po dva uzastopna temena mnogogugla
      double A = temena.get(i+1).getY() - temena.get(i).getY();
      double B = temena.get(i).getX() - temena.get(i+1).getX();
      double C = temena.get(i+1).getX()*temena.get(i).getY() -
           temena.get(i).getX()*temena.get(i+1).getY();

      // Odradjuje se polozaj sledeceg temena u odnosu na pravu
      int indeks = (i+2) % velicina;
      ind = A*temena.get(indeks).getX() + B*temena.get(indeks).getY() + C > 0;
      // Mnogougao je konveksan ukoliko i preostala temena imaju isti polozaj u
      // odnosu na pravu
      for(int j = 0; j < velicina; j++)
         if(j != i && j != i+1 && j!= i+2 &&
              (A*temena.get(j).getX() + B*temena.get(j).getY() + C > 0) != ind)
            return false;
   }
   return true;
}
```

```
   // String-reprezentacija mnogougla
   public String toString()
   {
      StringBuffer str = new StringBuffer("Temena mnogougla:\n");
      for(int i = 0; i < temena.size(); i++)
        str.append(temena.get(i) + " ");
      str.append("\nStranice mnogougla:\n");
      double[] stranice = stranice();
      for(int i = 0; i < stranice.length; i++)
        str.append(stranice[i] + " ");
      return str.toString();
   }

   // Staticki metod koji ucitava mnogougao
   public static Mnogougao ucitajMnogougao(Scanner skener)
   {
      int brojTemena = skener.nextInt();
      Tacka[] temena = new Tacka[brojTemena];
      for(int i = 0; i < brojTemena; i++)
        temena[i] = new Tacka(skener.nextDouble(), skener.nextDouble());
      return new Mnogougao(temena);
   }

}
```

18. *Lista Osoba - ArrayList*. Написати апликацију којом се описује поступак пријављивања особа на конкурс за улогу у филму. Подаци о особама су име и презиме особе и чувају се у листи. За имплементацију листе користити класу *java.util.ArrayList*. Потребно је реализовати следеће операције: додавање нове особе у листу, провера да ли се дата особа већ налази у листи, листање свих особа у абецедном поретку и смештање листе особа у датотеку *"C:\temp\Osobe.bin"* приликом завршетка рада. Уколико се програм покреће по први пут, кренути од празне листе, у супротном, особе ишчитати из датотеке *"C:\temp\Osobe.bin"*.

Објашњење:

Класа *java.util.ArrayList* представља имплементацију *java.util.List* интерфејса. Имплементиране су све операције са листом и дозвољени су сви типови елемената, укључујући и *null*. Класа обезбеђује све операције као и класа *java.util.Vector*, једина разлика је што је класа *java.util.Vector* синхронизована, а *ArrayList* није. То значи да се не може користити у раду са нитима.

Објекат класе *ArrayList* ради као и низ осим што додатно расте аутоматски, када је потребан већи капацитет. Као и низови, и објекат *ArrayList* садржи референце на објекте, не саме објекте. То је правило за све колекције.

За разлику од низа, објекат *ArrayList* карактерише се *величином (size)* и *капацитетом (capacity).*

Капацитет је максималан број објеката које објекат *ArrayList* може da sadrži. Капацитет је променљив, јер се аутоматски повећава када се дода нови објекат у већ пуну колекцију.

Класа *Osoba* имплементира интерфејсе *Comparable* и *Serializable*.
Имплементација интерфејса *Comparable* је неопходна због метода *compareTo()* којим се пореде две особе лексикографски. Поређење се врши тако што се прво пореде презимена особа, а ако су једнака, пореде се и њихова имена.
Статички метод *ucitajOsobu()* служи за учитавање особе са стандардног улаза.
Класа имплементира *Serializable* интерфејс да би се омогућило читање из датотеке и упис у датотеку објекта који представља листу особа.

Класа *ListaOsoba* садржи колекцију објеката класе *Osoba*. Колекција је типа *ArrayList*. Подаци о особама се учитавају из датотеке *"C:\temp\\ListaOsoba.bin"*, уколико она постоји. Датотека је представљена чланицом *datoteka*, типа *File*.
У конструктору се учитава објекат класе *ArrayList* из датотеке, ако она постоји. У супротном, објекат остаје празан. Као улазни ток користи се објекат класе *ObjectInputStream*, који као аргумент очекује објекат класе *FileInputStream*. Читање објекта се врши позивом метода *readObject()*, а повратна вредност је референца на учитани објекат. Тип повратне вредности је *Object*, тако да је неопходно извршити кастовање у тип објекта који се учитава, у овом случају *ArrayList<Osoba>*. Кастовање доводи до упозорења од стране компајлера, јер се сматра непровереним. Разлог је што компајлер не може да зна ког типа ће бити објекти који се учитавају.

Методом *sacuvaj()* се листа особа чува у датотеци и треба га позвати пре завршетка програма.

Методом *izlistajOsobe()* исписују се особе из колекције у лексикографском поретку. Дати поредак добија се сортирањем објеката у колекцији позивом статичког метода *Collections.sort()* над колекцијом.

Кориснику је при покретању програма понуђен мени са опцијама:
- опција 1 - унос нове особе и њено смештање у колекцију
- опција 2 - провера да ли се особа која се учитава са улаза пријавила на конкурс, тј. да ли се налази у колекцији
- опција 3 - приказ пријављених особа, тј. листање садржаја колекције у лексикографском поретку
- опција 9 - крај програма, када се аутоматски врши испис особа из колекције у датотеку.

Датотека *"C:\temp\\ListaOsoba.bin"* ће бити креирана при првом покретању програма, али је њен садржај иницијално празан. Ако се изабере опција 3, подаци о пријављеним особама се исписују у датотеку, чиме она добија свој садржај.
При сваком следећем покретању програма, почетни садржај датотеке *"C:\temp\\ListaOsoba.bin"* је резултат претходног покретања програма и може се проширити додавањем нових особа, избором опције 1 из менија и опције 3, како би се нови садржај уписао у датотеку.

Опис поступка серијализације:
Серијализација је процес који обухвата упис објеката у датотеку и читање објеката из датотеке.
Класа тих објеката мора да имплементира интерфејс *Serializable*.
У већини случајева довољно је само декларисати да класа имплементира овај интерфејс и није неопходан никакав додатни код. Ако постоји директна или индиректна наткласа која не имплементира *Serializable* интерфејс, та класа мора имати подразумевани конструктор, а изведена класа се мора побринути о прослеђивању чланова базне класе улазном току.
Уколико објекат садржи инстанцне чланице класног типа, те класе такође морају имплементирати *Serializable* интерфејс. Серијализација тих чланова вршиће се аутоматски.

Упис објеката у датотеку:
Упис објеката у датотеку врши се позивом метода *writeObject()* над објектом који представља излазни ток. У питању је објекат типа *ObjectOutputStream*. Потребно је обрадити изузетке који могу бити избачени приликом креирања излазног тока.
Фрагмент кода који креира објекат типа *ObjectOutputStream* и исписује објекат *osobe* типа *ArrayList* у датотеку *"C:\temp\\ListaOsoba.bin":*

```
try
{
   System.out.println("Cuvanje liste osoba");
   ObjectOutputStream out = new ObjectOutputStream(
                              new FileOutputStream("C\\temp\\Imenik.bin"));
   out.writeObject(osobe);
   System.out.println("Kraj");
   out.close();
}
```

```
     catch (IOException e)
     {
        e.printStackTrace();
        System.exit(1);
```

Читање објеката из датотеке:
Као улазни ток користи се објекат класе *ObjectInputStream.* Читање објекта се врши позивом метода *readObject()*, а повратна вредност је референца на учитани објекат. Тип повратне вредности је *Object*, тако да је неопходно извршити кастовање у тип објекта који се учитава, у овом случају *ArrayList<Osoba>*.
Фрагмент кода који креира објекат типа *ObjectInputStream* и чита објекат типа *ArrayList* из датотеке *"C:\temp\\ListaOsoba.bin":*

```
     try
     {
        ObjectInputStream ulazniTok =
             new ObjectInputStream(new FileInputStream(datoteka));
        osobe = (ArrayList<Osoba>) ulazniTok.readObject();
        ulazniTok.close();
     }
     catch (ClassNotFoundException e)
     {
        e.printStackTrace();
        System.exit(1);
     }
     catch (IOException e)
     {
        e.printStackTrace();
        System.exit(1);
     }
```

Решење:

Osoba.java

```
package arrayList;

import java.io.Serializable;
import java.util.Scanner;

public class Osoba implements Comparable<Osoba>, Serializable
{
   private String ime;
   private String prezime;
   private static Scanner skener = new Scanner(System.in);

   public static final long serialVersionUID = 1;

   public Osoba(String ime, String prezime)
   {
      this.ime=ime;
      this.prezime=prezime;
   }

   public int compareTo(Osoba osoba)
   {
      int rezultat = prezime.compareTo(osoba.prezime);
      if(rezultat == 0)
         return ime.compareTo(osoba.ime);
      return rezultat;
   }

   public boolean equals(Object obj)
   {
```

```
      return compareTo((Osoba)obj) == 0;
   }

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

   public static Osoba ucitajOsobu()
   {
      System.out.println("Unesite ime osobe");
      String ime = skener.next();
      System.out.println("Unesite prezime osobe");
      return new Osoba(ime, skener.next());
   }
}
```

ListaOsoba.java

```
package arrayList;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;

public class ListaOsoba implements Serializable
{
   public static final long serialVersionUID = 1;

   private ArrayList<Osoba> osobe = new ArrayList<Osoba>();
   private File datoteka = new File("C:\\temp\\ListaOsoba.bin");

   public ListaOsoba()
   {
      if (datoteka.exists())
         try
         {
            ObjectInputStream ulazniTok =
               new ObjectInputStream(new FileInputStream(datoteka));
            osobe = (ArrayList<Osoba>) ulazniTok.readObject();
            ulazniTok.close();
         }
         catch (ClassNotFoundException e)
         {
            e.printStackTrace();
            System.exit(1);
         }
         catch (IOException e)
         {
            e.printStackTrace();
            System.exit(1);
         }
   }

   public void sacuvaj()
   {
      try
      {
         System.out.println("Cuvanje liste osoba");
         ObjectOutputStream out = new ObjectOutputStream(
```

```
                                    new FileOutputStream(datoteka));
         out.writeObject(osobe);
         System.out.println("Kraj");
         out.close();
      }
      catch (IOException e)
      {
         e.printStackTrace();
         System.exit(1);
      }
   }

   public void dodajOsobu(Osoba osoba)
   {
        osobe.add(osoba);
   }

   public void izlistajOsobe()
   {
        Collections.sort(osobe);
        for(Osoba osoba : osobe)
           System.out.println(osoba);
   }

   public Osoba vratiOsobu(int i)
   {
        return osobe.get(i);
   }

   public boolean sadrzi(Osoba osoba)
   {
      return osobe.contains(osoba);
   }
}
```

TestOsobe.java

```
package arrayList;

import java.util.Scanner;

public class TestOsobe
{
   public static void main(String[] args)
   {
      ListaOsoba osobe = new ListaOsoba();
      Scanner skener = new Scanner(System.in);

      Osoba osoba;

      while(true)
      {
         System.out.println("Unesite:\n" +
                            "1 za unos nove osobe");
         System.out.println("2 za proveru da li se osoba prijavila");
         System.out.println("3 za prikaz osoba u abecednom poretku ");
         System.out.println("9 za kraj programa " +
                             "(automatski se vrsi ispis u datoteku)");

         int opcija = skener.nextInt();

         switch(opcija)
         {
            case 1:
               osobe.dodajOsobu(Osoba.ucitajOsobu());
               break;
```

```
            case 2:
               osoba = Osoba.ucitajOsobu();
               if(osobe.sadrzi(osoba) == false)
                  System.out.println("Osoba se nije prijavila na konkurs");
               else
                  System.out.println("Osoba se prijavila na konkrurs");
               break;

            case 3:
               osobe.izlistajOsobe();
               break;

            case 9:
               osobe.sacuvaj();
               System.out.println("Kraj programa");
               return;

            default:
               System.out.println("Neispravan unos, pokusajte ponovo");
               break;
         }
      }
   }
}
```

19. *Велики бројеви, ArrayList.* Написати класу за рад са великим ненегативним целим бројевима. Бројеви могу бити произвољне дужине.
Од операција имплементирати: унос "великог" броја са тастатуре, креирање "великог" броја од цифара задатог целог броја, сабирање "великих" бројева, множење "великих" бројева, рачунање факторијела ненегативног целог броја и рачунање задатог елемента Фибоначијевог низа.
Предефинисати методе *compareTo()* и *equals()* наслеђене од класе *Object* тако да раде на одговарајући начин са "великим" бројевима.
Написати потом и тест-класу.

Објашњење:
За разлику од задатка ??? овде не постоји ограничење броја цифара, па је за њихово смештање уместо низа коришћен објекат типа *ArrayList<>*. Измене у коду су минималне и углавном се своде на позив метода *get()*, за дохватање вредности цифре из *ArrayList<>* објекта, и *add()*, за додавање цифре у *ArrayList<>* објекат, на местима где је у решењу задатка ??? коришћено индексирање низа. У методу *pomnoziSa10naK(int k)* нема потребе за експлицитним преписивањем постојећих цифара броја, већ је додавање *k* нула на крај броја реализовано у петљи која *k* пута додаје 0 на почетак *ArrayList<>* објекта.

Решење:

Broj.java

```
package velikiBrojeviArrayList;

import java.util.ArrayList;
import java.util.Scanner;

public class Broj implements Comparable<Broj>
{
   private ArrayList<Integer> cifre = new ArrayList<Integer>();

   private static Scanner sc = new Scanner(System.in);

   public Broj(int n)
   {
      // 0 je specijalan slucaj
      if (n == 0)
```

```
   {
      cifre.add(0);
      return;
   }

   while (n != 0)
   {
      cifre.add(n % 10);
      n /= 10;
   }

}

public Broj(final ArrayList<Integer> broj)
{
   cifre = new ArrayList<Integer>(broj);
}

private void obrisiVodeceNule()
{
   /*
    * brisu se sve vodece nule, osim u slucaju da je broj sastavljen od svih
    * nula, tada se ostavlja samo jedna
    */
   int duzina = cifre.size();
   while (duzina > 1 && cifre.get(duzina - 1) == 0)
      cifre.remove(duzina - 1);
}

private void obrniCifre()
{
   for (int i = 0, j = cifre.size() - 1; i < j; i++, j--)
   {
      int pom = cifre.get(i);
      cifre.set(i, cifre.get(j));
      cifre.set(j, pom);
   }
}

public static Broj ucitajBroj()
{
   ArrayList<Integer> cifre = new ArrayList<Integer>();
   String unos = sc.next();

   for (int duzina = 0; duzina < unos.length()
       && Character.isDigit(unos.charAt(duzina)); duzina++)
      cifre.add(unos.charAt(duzina) - '0');

   Broj broj = new Broj(cifre);
   broj.obrniCifre();
   broj.obrisiVodeceNule();

   return broj;
}

public String toString()
{
   StringBuffer rez = new StringBuffer();
   for (int i = cifre.size() - 1; i >= 0; i--)
      rez.append(cifre.get(i));
   return rez.toString();
}

public int compareTo(Broj b)
{

   // Uporedjujemo duzine brojeva
```

```
   int duzina = cifre.size();
   int duzinaB = b.cifre.size();
   if (duzina > duzinaB)
      return 1;
   if (duzina < duzinaB)
      return -1;

   /*
    * U ovom trenutku znamo da su brojevi iste duzine, tako da prelazimo na
    * poredjenje cifra po cifra, pocevsi od cifre najvece tezine
    */
   for (int i = duzina - 1; i >= 0; i--)
   {
      if (cifre.get(i) > b.cifre.get(i))
         return 1;
      if (cifre.get(i) < b.cifre.get(i))
         return -1;
   }
   return 0;
}

public boolean equals(Object b)
{
   return compareTo((Broj) b) == 0;
}

public Broj saberi(final Broj b)
{
   int prenos = 0; // prenos sa prethodne pozicije
   ArrayList<Integer> cifreRezultata = new ArrayList<Integer>();
   int i;

   int duzina = cifre.size();
   int duzinaB = b.cifre.size();

   for (i = 0; i < duzina || i < duzinaB; i++)
   {
      int cifraRezultata = ((i < duzina) ? cifre.get(i) : 0)
          + ((i < duzinaB) ? b.cifre.get(i) : 0) + prenos;

      cifreRezultata.add(cifraRezultata % 10);
      prenos = cifraRezultata / 10;
   }

   if (prenos != 0)
      cifreRezultata.add(prenos);

   return new Broj(cifreRezultata);
}

private Broj pomnoziCifrom(int cifra)
{
   int prenos = 0;

   // Ako je cifra 0, rezultat je 0
   if (cifra == 0)
      return new Broj(0);

   ArrayList<Integer> cifreRezultata = new ArrayList<Integer>();
   for (int i = 0; i < cifre.size(); i++)
   {
      int pom = cifre.get(i) * cifra + prenos;

      cifreRezultata.add(pom % 10);
      prenos = pom / 10;
   }
```

```
   if (prenos != 0)
      cifreRezultata.add(prenos);

   return new Broj(cifreRezultata);
}

/*
 * metod mnozi broj sa 10^k, menja tekuci broj rezultatom.
 * Koristi se za pojednostavljenje mnozenja
 */
private void pomnoziSa10naK(int k)
{
   /*
    * nema svrhe nulu mnoziti sa 10^k jer bi to dalo kao rezultat niz od k+1
    * nule
    */
   if (cifre.size() == 1 && cifre.get(0) == 0)
      return;

   // dodajemo k nula na kraj broja, tj. na pocetak ArrayList
   for (int i = 0; i < k; i++)
      cifre.add(0, 0);

}

public Broj pomnozi(final Broj b)
{

   /*
    * Broj pom ce sadrzati rezultat mnozenja tekuceg broja jednom po jednom
    * cifrom broja b, dok ce se na broj rezultat dodavati svaki put pom*(10^i)
    */

   Broj rezultat = new Broj(0);

   for (int i = 0; i < b.cifre.size(); i++)
   {
      Broj pom = pomnoziCifrom(b.cifre.get(i));

      pom.pomnoziSa10naK(i);
      rezultat = rezultat.saberi(pom);
   }

   return rezultat;

}

// metod izracunava faktorijel nenegativnog broja n kao veliki broj
public static Broj faktorijel(int n)
{
   Broj rezultat = new Broj(1);

   for (int i = 2; i <= n; i++)
   {
      Broj brojI = new Broj(i);
      rezultat = rezultat.pomnozi(brojI);
   }

   return rezultat;
}

private Broj dekrementiraj()
{
   int pozajmica = 0;
   ArrayList<Integer> cifreRezultata = new ArrayList<Integer>();

   if (cifre.get(0) == 0)
```

```
    {
      cifreRezultata.add(9);
      pozajmica = 1;

      for (int i = 1; i < cifre.size(); i++)
      {
        if (cifre.get(i) - pozajmica < 0)
        {
          pozajmica = 1;
          cifreRezultata.add(9);
        } else
        {
          cifreRezultata.add(cifre.get(i) - pozajmica);
          pozajmica = 0;
        }
      }

      if (cifreRezultata.get(cifre.size() - 1) == 0)
        cifreRezultata.remove(cifre.size() - 1);

      return new Broj(cifreRezultata);
    } else
    {
      cifreRezultata.add(cifre.get(0) - 1);
      for (int i = 1; i < cifre.size(); i++)
        cifreRezultata.add(cifre.get(i));
      return new Broj(cifreRezultata);
    }
  }

  // metod racuna faktorijel tekuceg velikog broja
  public Broj faktorijel()
  {
    if (equals(new Broj(0)))
      return new Broj(1);

    Broj manje1 = dekrementiraj();

    return pomnozi(manje1.faktorijel());
  }

  // metod racuna n-ti clan Fibonacijevog niza, rekurzivna varijanta
  public static Broj fibonaciR(int n)
  {
    if (n == 1 || n == 2)
      return new Broj(1);

    return fibonaciR(n - 1).saberi(fibonaciR(n - 2));
  }

  /*
* metod racuna n-ti clan Fibonacijevog niza, iterativna varijanta,
* dinamicko programiranje
*/

  public static Broj fibonaci(int n)
  {
    Broj[] rez = new Broj[n];

    rez[0] = new Broj(1);

    if (n >= 2)
      rez[1] = new Broj(1);

    for (int i = 3; i <= n; i++)
      rez[i - 1] = rez[i - 2].saberi(rez[i - 3]);
```

```
      return rez[n - 1];

   }

}
```

TestVelikiBrojevi.java

```
package velikiBrojeviArrayList;

import java.util.Scanner;

public class TestVelikiBrojevi
{

   public static void main(String[] args)
   {

      Scanner sc = new Scanner(System.in);

      // Unos velikih brojeva
      System.out.print("Unesite prvi broj: ");
      Broj a = Broj.ucitajBroj();

      System.out.print("Unesite drugi broj: ");
      Broj b = Broj.ucitajBroj();

      // Prikaz velikih brojeva
      System.out.println("a je: " + a);
      System.out.println("b je: " + b);

      // Sabiraju se i ispisuje se zbir
      System.out.println("Zbir je: " + a.saberi(b));

      // 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"
                      );


   }
```

```
}
```

У Јавиној библиотеци, у пакету *java.math*, постоји класа *BigInteger* за рад са великим целим бројевима. Ова класа подржава целе бројеве произвољне прецизности и поседује аналогоне за све операције које су у Јави подржане за примитивни тип *int*, као и све релевантне методе класе *java.lang.Math* (испитивање да ли је број прост, генерисање простих бројева, манипулација битовима, највећи заједнички делилац итд.). Дељење нулом, као и код типа *int*, доводи до избацивања изузетка типа *ArithmeticException*. Такође, дељење негативног броја позитивним даје негативан (или 0) остатак. Једино се све везано за прекорачење игнорише, јер *BigInteger* објекти су у стању да се својом величином прилагоде резултату операције произвољне величине. Подржане су и операције шифтовања, битовске и логичке операције, као и операције поређења.

У наставку текста су побројани неки најзначајнији методи класе *BigInteger*.

```
   public BigInteger(String val);
\\ konstruktor koji prevodi String-reprezentaciju u BigInteger

   public static BigInteger valueOf(long val);
      \\ prevodi long u BigInteger

   public BigInteger add(BigInteger val);         \\ vraca: this + val
   public BigInteger substract(BigInteger val);   \\ vraca: this - val
   public BigInteger multiply(BigInteger val);    \\ vraca: this * val
   public BigInteger divide(BigInteger val);      \\ vraca: this / val
public BigInteger[] divideAndRemainder(BigInteger val);
                \\ vraca niz koji sadrzi, redom, this/val i this % val

public BigInteger remainder(BigInteger val);   \\ vraca this % val
public BigInteger pow(int exponent);     \\ vraca this ^ exponent
public BigInteger gcd(BigInteger val);
\\ vraca NZD(abs(this), abs(val))
public BigInteger abs();                     \\ vraca abs(this)
   public BigInteger negate();               \\ vraca -this
   public int signum();    \\ vraca: -1 ako je broj negativan,
                              \\         0 ako je u pitanju 0
                              \\         1 ako je broj pozitivan

   public BigInteger shiftLeft(int n);          \\ vraca this << n
   public BigInteger shiftRight(int n);       \\ vraca this >> n

   public BigInteger and(BigInteger val);       \\ vraca this & val
   public BigInteger or(BigInteger val);      \\ vraca this | val
   public BigInteger xor(BigInteger val);       \\ vraca this ^ val
   public BigInteger not();                     \\ vraca ~this
   public BigInteger andNot(BigInteger val); \\ vraca this & ~val

   public boolean testBit(int n);
   \\ ispitivanje da li je n-ti bit jedinica
   public BigInteger setBit(int n);
   \\ postavljanje n-tog bita na jedinicu
   public BigInteger clearBit(int n);
   \\ postavljanje n-tog bita na nulu
   public BigInteger flipBit(int n);
   \\ invertovanje n-tog bita
   public getLowestSetBit();
   \\ indeks krajnjeg desnog (najmanje tezine) bita jedinice

   public int compareTo(BigInteger val);
   public boolean equals(Object x);

   public BigInteger min(BigInteger val);     \\ min(this, val)
   public BigInteger max(BigInteger val);     \\ max(this, val)
```

```
   public int hashcode();

   public String toString();

   public int intValue();        \\ prevodjenje BigInteger-a u int
   public long longValue();      \\ prevodjenje BigInteger-a u long
   public float floatValue();    \\ prevodjenje BigInteger-a u float
   public double doubleValue();  \\ prevodjenje BigInteger-a u double
```

Следи решење задатка у коме је коришћена библиотечка класа *BigInteger*.
Уместо *ArrayList<>* објекта, за чување великог броја користи се *BigInteger* објекат. Тиме је знатно упрошћена имплементација метода за учитавање великог броја са тастатуре, креирање великог броја од цифара задатог целог броја, сабирање, множење и декрементирање, јер су ове операције, или њихови најбитнији делови, имплементирани у класи *BigInteger*. Тако, за креирање великог броја од цифара датог целог броја користи се статички метод *valueOf()* класе *BigInteger*. Како у класи *BigInteger* не постоји копи-конструктор, прављење копије *BigInteger* објекта у копи-конструктору класе *Broj* врши се позивом конструктора класе *BigInteger* који као аргумент прима *String*-репрезентацију тог објекта. Уведен је и конструктор који као аргумент прима *BigInteger* објекат. Методи *toString()* и *equals()* директно се ослањају на истоимене методе класе *BigInteger*, док методи *saberi()* и *pomnozi()* такође користе одговарајуће методе *add()* и *multiply()* класе *BigInteger*. Декрементирање је реализовано одузимањем јединице (константа *BigInteger.ONE*) помоћу метода *subtract()* класе *BigInteger*. Методи за рачунање факторијела и Фибоначијевих бројева, као и тест-класа, задржали су потпуно исти облик.

Broj.java

```
package velikiBrojeviBigInteger;

import java.math.BigInteger;
import java.util.Scanner;

public class Broj implements Comparable<Broj>
{
   private BigInteger broj;

   private static Scanner sc = new Scanner(System.in);

   public Broj(int n)
   {
      broj = BigInteger.valueOf(n);
   }

   public Broj(final Broj b)
   {
      broj = new BigInteger(b.broj.toString());
   }

   public Broj(final BigInteger bi)
   {
      broj = bi;
   }

   public static Broj ucitajBroj()
   {
      String unos = sc.next();
      return new Broj(new BigInteger(unos));
   }

   public String toString()
   {
      return broj.toString();
   }
```

```
public int compareTo(Broj b)
{
   return broj.compareTo(b.broj);
}

public boolean equals(Object b)
{
   return compareTo((Broj) b) == 0;
}

public Broj saberi(final Broj b)
{
   return new Broj(broj.add(b.broj));
}

public Broj pomnozi(final Broj b)
{
   return new Broj(broj.multiply(b.broj));
}

// metod izracunava faktorijel nenegativnog broja n kao veliki broj
public static Broj faktorijel(int n)
{
   Broj rezultat = new Broj(1);

   for (int i = 2; i <= n; i++)
   {
      Broj brojI = new Broj(i);
      rezultat = rezultat.pomnozi(brojI);
   }

   return rezultat;
}

private Broj dekrementiraj()
{
   return new Broj(broj.subtract(BigInteger.ONE));
}

// metod racuna faktorijel tekuceg velikog broja
public Broj faktorijel()
{
   if (equals(new Broj(0)))
      return new Broj(1);

   Broj manje1 = dekrementiraj();
   return pomnozi(manje1.faktorijel());
}

// metod racuna n-ti clan Fibonacijevog niza, rekurzivna varijanta
public static Broj fibonaciR(int n)
{
   if (n == 1 || n == 2)
      return new Broj(1);

   return fibonaciR(n - 1).saberi(fibonaciR(n - 2));
}

/*
 * metod racuna n-ti clan Fibonacijevog niza, iterativna varijanta,
 * dinamicko programiranje
 */
public static Broj fibonaci(int n)
{
   Broj[] rez = new Broj[n];
```

```
      rez[0] = new Broj(1);

      if (n >= 2)
        rez[1] = new Broj(1);

      for (int i = 3; i <= n; i++)
        rez[i - 1] = rez[i - 2].saberi(rez[i - 3]);

      return rez[n - 1];

   }

}
```

TestVelikiBrojevi.java

```
package velikiBrojeviBigInteger;

import java.util.Scanner;

public class TestVelikiBrojevi
{

   public static void main(String[] args)
   {

      Scanner sc = new Scanner(System.in);

      // Unos velikih brojeva
      System.out.print("Unesite prvi broj: ");
      Broj a = Broj.ucitajBroj();

      System.out.print("Unesite drugi broj: ");
      Broj b = Broj.ucitajBroj();

      // Prikaz velikih brojeva
      System.out.println("a je: " + a);
      System.out.println("b je: " + b);

      // Sabiraju se i ispisuje se zbir
      System.out.println("Zbir je: " + a.saberi(b));

      // 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"
                        );


    }

}
```

У следећој табели дат је упоредни приказ времена извршавања рекурзивног и итеративног(динамичко програмирање) метода за рачунање задатог елемента Фибоначијевог низа. Приказани су резултати за све три варијанте имплементације "великих" бројева: помоћу низа (димензије 2000), *ArrayList* и *BigInteger*. Времена извршавања дата су у наносекундама (*ns*).

Сви програми су извршавани на *Intel* 1.6GHz PC-ју са 1GB RAM-а под *Windows* оперативним системом.

n	***Низ (рек.)***	***Низ (Д.П.)***	***ArrayList (рек.)***	***ArrayList (Д.П.)***	***BigInteger (рек.)***	***BigInteger (Д.П.)***
10	234387	74591	235505	71797	599517	81575
20	18175215	109232	18258745	93029	13284929	58946
30	1212170262	107555	1200488051	102248	241484602	94705
35	13267978192	108673	12835275154	114819	2570869482	115937
40	143979474766	131581	142040744284	110908	23104238108	69841
41	230863297989	139683	224275997089	138565	37505605321	84927
42	371790006551	106439	367851318812	185498	63004935544	96660
44	--	--	--	--	162422594771	87441
50	--	--	--	--	3162917299266	533308
8269 8270		810659455 Java heap space				
12538 12539				8534875242 Java heap space		
38273 38274						1109730148 Java heap space

Из приказаних резултата може се видети да рекурзивна варијанта има време извршавања које је дуже од 1*s* већ за *n*=30 за имплементације које користе низ и *ArrayList*, односно *n*=35 за имплементацију која користи *BigInteger*. За *n*=42, извршавање верзија које користе низ и *ArrayList* траје преко 6 минута, док извршавање верзије која користи *BigInteger* траје нешто преко 1 минута. Највећа вредност индекса за коју је извршаван рекурзивни метод имплементације која користи *BigInteger* је *n*=50, а извршавање метода трајало је 52.7 минута. Време извршавања рекурзивне варијанте метода расте експоненцијално са повећањем вредности индекса.

За сва три начина имплементације, у табели су назначени гранични случајеви које је могуће решити итеративном варијантом метода која користи динамичко програмирање. То су: *n*=8269 за имплементацију која користи низ, *n*=12538 за имплементацију која користи *ArrayList*, и *n*=38273 за имплементацију која користи *BigInteger*. За вредности индекса веће од граничних, програми имају проблема са недостатком меморије. Време извршавања итеративне варијанте метода која користи динамичко програмирање расте линеарно и чак и у граничним случајевима је краће од 10*s*. Посебно, у граничном случају *n*=38273 имплементације која користи *BigInteger* време извршавања је нешто преко 1*s*.

Теоретски, времена извршавања у свакој појединачној колони табеле требало би да чине монотоно растући низ јер је у рачунање произвољног елемента низа укључено и рачунање претходних елемената. Неправилности које су добијене у табели имају објашњење у чињеници да метод *System.nanoTime()* даје *ns* прецизност, али не и *ns* тачност.

20. *Именик.* Написати Јава-програм за рад са једноставним именником. Уноси у именику састоје се од имена и презимена особе и њеног телефонског броја. Телефонски број састоји се из два дела: позивног и броја у локалу. Није потребно вршити провере валидности ових бројева. Од операција над именником реализовати: додавање новог уноса у именик, тражење телефонског броја задате особе, листање свих уноса именика у абецедном поретку по презимену и имену особе и смештање именика у фајл "`C:\temp\Imenik.bin`" приликом завршетка рада. Уколико се програм покреће по први пут, кренути од празног именика, а иначе уносе ишчитати из фајла "`C:\temp\Imenik.bin`".

Објашњење:

Мапа је начин смештања података који минимизује потребу за претраживањем када желимо да приступимо објекту.
Сваком објекту придружује се јединствени кључ који се користи за одређивање где треба сместити референцу на објекат. Објекат се смешта у мапу заједно са својим кључем. Са датом вредношћу за кључ, увек се мање-више директно долази до објекта који одговара том кључу.
Имплементација мапе у *Java collections framework*, обезбеђена класом *HashMap<>*, одвојена је од низа у који се физички смештају парови кључ/објекат. Индекс овог низа, који одговара пару кључ/објекат, добија се од кључа коришћењем метода *hashCode()*. Овај метод је наслеђен у свим класама од класе *Object*, али, као што ћемо ускоро видети, обично га је потребно предефинисати (енг. override) у класи кључа.
Иако сваки кључ мора бити јединствен, не мора се од сваког кључа добити јединствени хешкод. Ситуација када два или више различитих кључева дају исту хеш-вредност назива се колизијом. *HashMap<>* објекат ради са колизијама тако што смешта све кључ/објекат парове са истом хеш-вредношћу у повезану листу. Ако се колизије дешавају пречесто, очигледно долази до успоравања процеса смештања података и приступања подацима. Зато постоји јак подстрек да се број појава колизија минимизује, а цена смањења могућности колизија је много празног простора у хеш-табели.
Класа *Object* дефинише *public* метод *hashCode()*, па се објекти произвољне класе могу користити као кључеви. Метод није баш универзално применљив. Пошто за добијање хеш-вредности објекта обично користи меморијску адресу на којој је објекат смештен, различити објекти увек производе различите хеш-вредности. То је повољно, јер ће операције над хеш-мапом бити ефикасније, међутим оно што није добро је то да различити објекти који имају идентичне вредности одговарајућих инстанцних променљивих производе различите хеш-вредности, па их не можемо поредити. То постаје сметња ако користимо подразумевани *hashCode()* метод у класи објеката које користимо као кључеве. У том случају, објекту смештеном у хеш-мапи никада се не може приступити коришћењем неке

друге инстанце кључа до оне помоћу које је објекат смештен у мапу, чак и ако је тај други кључ идентичан у свим осталим аспектима кључу који је искоришћен за смештање објекта у мапу. А управо је то случај у већини ситуација. Решење овог проблема било би да се некако направи хеширање инстанцних променљивих кључа.

Да би објекти наших класа могли да се користе као кључеви у хеш-табели, мора се предефинисати метод *equals()* класе *Object*. Овај метод се користи у методима класе *HashMap<>* за утврђивање да ли су 2 кључа једнака, па наша верзија овог метода треба да врати *true* када два различита објекта садрже идентичне вредности одговарајућих инстанцних променљивих, а *false* иначе.

Такође, потребно је предефинисати метод *hashCode()*, који враћа хеш-вредност за објекат као вредност типа *int*. Први корак је добити цео број за сваку инстанцну променљиву. А затим треба укомбиновати те бројеве за добијање вредности која ће бити хешкод за објекат.

Један начин да се ово уради јесте да се сваки цео број који одговара некој инстанцној променљивој помножи различитим простим бројем и саберу тако добијени резултати. Није битно које просте бројеве ћемо користити све док:

- нису тако велики да резултат изађе из опсега типа *int*
- користимо различит прост број за сваку инстанцну променљиву

Како од инстанцне променљиве добити цео број?

Генерисање целог броја за инстанцну променљиву типа *String* је једноставно: само позовемо *hashCode()* метод за инстанцну променљиву, који је у класи *String* имплементиран тако да произведе добре хешкод вредности које ће бити једнаке за идентичне стрингове. *int* инстанцне променљиве можемо користити такве какве су, док инстанцне променљиве реалних типова захтевају нешто обраде.

Нпр. желимо да користимо објекте класе *Osoba* као кључеве у хеш-табели, а инстанцне променљиве су *ime* и *prezime*, типа *String*, и *godine*, типа *int*.

Могли бисмо *hashCode()* метод те класе да имплементирамо са:

```
public int hashCode()
{
       return 13 * ime.hashCode() + 17 * prezime.hashCode()
 + 19 * godine;
}
```

Ако је инстанцна променљива референца на објекат неке класе, а не променљива примитивног типа, треба имплементирати *hashCode()* метод за ту класу и користити га у рачунању хешкода објекта класе кључа.

У класи *Osoba*, чије објекте ћемо користити као кључеве у хеш-табели, предефинисали смо методе *equals()* и *hashCode()* наслеђене од класе *Object*. У имплементацији метода *equals()* само позивамо метод *compareTo()* који смо имплементирали због *Comparable<>* интерфејса. Такође, класа имплементира *Serializable* интерфејс како би објекат који представља именик могао да се пише у фајл и чита из њега. Постоји и статички метод *ucitajOsobu()* који служи да се подаци о особи унесу са стандардног улаза.

Унос у именику састоји се од података о особи и њеном телефонском броју. Класа *Unos* имплементира интерфејс *Comparable<>* како би било могуће сортирати листу уноса коришћењем статичког метода *sort()* класе *Collections* из пакета *java.util*. Метод *compareTo()* пореди два уноса поредећи њихове *Osoba*-чланове јер ће сортирање бити извршено абецедно по презимену и имену особе. Класа *Osoba* такође имплементира *Comparable<>* интерфејс, па два *Osoba* објекта поредимо *compareTo()* методом класе *Osoba*.

У класи *Imenik*, за чување *Unos* објеката користи се члан *imenik* типа *HashMap<Osoba, Unos>*. Као кључ сваког уноса користи се објекат типа *Osoba* који представља део тог уноса јер се претрага врши по имену и презимену особе.
Чланом *fajl* одређен је фајл у коме се чува мапа са уносима именика. Ако тај фајл постоји, конструктор чита *HashMap<>* објекат из фајла. Ако фајл не постоји, конструктор не ради ништа, па ће новокреирани *Imenik* објекат користити подразумевани празан *HashMap<>* објекат. Кастовање референце коју је вратио метод *readObject()* у тип *HashMap<Osoba, Unos>* довешће до упозорења од стране компајлера да имамо непроверено кастовање. Не постоји начин да се то избегне јер компајлер не може у фази компајлирања програма знати који ће бити тип објекта који се у току његовог извршавања чита из фајла. Све ће бити у реду док ми сами знамо шта радимо.

Метод *sacuvaj()* обезбеђује чување мапе у фајлу, па га је потребно позвати пре краја програма.

Метод *izlistajUnose()* исписује на стандардни излаз уносе из именика сортиране у абецедном поретку по презимену и имену особе. Позив метода *values()* за објекат *imenik* враћа вредност типа *Collection<>*, која представља колекцију свих објеката из мапе. Међутим, метод *sort()* очекује *List<>* као тип свог аргумента, па повратну вредност метода *values()* прослеђујемо конструктору класе *LinkedList<>* како бисмо добили објекат *unosi* одговарајућег типа. Класа *LinkedList<>* имплементира *List<>* интерфејс, па можемо проследити објекат *unosi* методу *sort()* како би сортирао уносе на жељени начин. Сортирани уноси се затим исписују на стандардни излаз.

Приликом првог извршавања програма, биће креиран нови фајл и читав именик биће смештен у њега. У извршавањима која буду следила, конструктор класе *Imenik* читаће из фајла, па ће претходни уноси бити доступни.

Детаљније о серијализацији:
Серијализација је процес писања објеката у фајл и читања објеката из фајла.
Поступак је зачуђујуће једноставан.
Писање објеката у фајл
Неопходно је да класа објеката које желимо писати у фајл или читати из фајла имплементира *Serializable* интерфејс. У већини случајева довољно је само декларисати да класа имплементира овај интерфејс и није неопходан никакав додатни код. Ситуација када то није довољно је када постоји директна или индиректна суперкласа која не имплементира *Serializable* интерфејс. Тада та суперкласа мора имати подразумевани конструктор, а изведена класа се мора побринути о прослеђивању чланова базне класе у излазни ток.
Уколико постоје чланови објекта који су референце на објекте неких класа, те класе такође морају имплементирати *Serializable* интерфејс и онда ће се њихова серијализација вршити аутоматски.
Писање објеката у фајл обавља се позивом метода *writeObject()* за објекат типа *ObjectOutputStream*. При том треба предузети мере да се ухвате изузеци који могу бити избачени.
Следи фрагмент кода који креира *ObjectOutputStream* објекат и пише објекат *imenik* типа *HashMap<Osoba,Unos>* у фајл "C:\temp\Imenik.bin".

```
try
{
     ObjectOutputStream out = new ObjectOutputStream(
        new FileOutputStream(
new File("C:/temp/Imenik.bin")));
     out.writeObject(imenik);
     out.close();
} catch (IOException e)
{
     e.printStackTrace();
     System.exit(1);
```

```
}
```

Читање објеката из фајла

Подједнако је једноставно као и писање. Креира се *ObjectInputStream* објекат за читање из жељеног фајла, а потом се објекти из тог фајла читају позивима метода *readObject()*. Такође, потребно је побринути се да се обраде изузеци који могу бити избачени. Метод *readObject()* враћа референцу на прочитани објекат као вредност типа *Object*, па је неопходно извршити кастовање у одговарајући тип објекта. Низови такође представљају објекте, па се ово правило о кастовању односи и на њих.

Следи фрагмент кода који креира *ObjectInputStream* objekat и чита објекат типа *HashMap<Osoba,Unos>* из фајла "`C:\temp\Imenik.bin`".

```
try
{
     ObjectInputStream in = new ObjectInputStream(
new FileInputStream(
new File("C:/temp/Imenik.bin")));
     imenik = (HashMap<Osoba, Unos>) in.readObject();
     in.close();
} catch (ClassNotFoundException e)
{
     e.printStackTrace();
     System.exit(1);
} catch (IOException e)
{
     e.printStackTrace();
     System.exit(1);
}
```

Решење:

Osoba.java

```
package imenik;

import java.io.Serializable;
import java.util.Scanner;

public class Osoba implements Comparable<Osoba>, Serializable
{
   private String ime;
   private String prezime;
   private static Scanner sc = new Scanner(System.in);

   public Osoba(String ime, String prezime)
   {
      this.ime = ime;
      this.prezime = prezime;
   }

   public int compareTo(Osoba osoba)
   {
      int rezultat = prezime.compareTo(osoba.prezime);
      if (rezultat == 0)
         return ime.compareTo(osoba.ime);
      return rezultat;
   }

   public boolean equals(Object obj)
   {
      return compareTo((Osoba) obj) == 0;
```

```
   }

   public int hashCode()
   {
      return 7 * ime.hashCode() + 13 * prezime.hashCode();
   }

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

   public static Osoba ucitajOsobu()
   {
      System.out.println("Unesite ime osobe");
      String ime = sc.next();
      System.out.println("Unesite prezime osobe");
      return new Osoba(ime, sc.next());
   }

}
```

BrojTelefona.java

```
package imenik;

import java.io.Serializable;
import java.util.Scanner;

public class BrojTelefona implements Serializable
{
   private String pozivni;
   private String broj;
   private static Scanner sc = new Scanner(System.in);

   public BrojTelefona(String pozivni, String broj)
   {
      this.pozivni = pozivni;
      this.broj = broj;
   }

   public String toString()
   {
      return "(" + pozivni + ")" + broj;
   }

   public static BrojTelefona ucitajBrojTelefona()
   {
      System.out.println("Unesite pozivni broj");
      String poz = sc.next();
      System.out.println("Unesite broj u lokalu");
      return new BrojTelefona(poz, sc.next());
   }

}
```

Unos.java

```
package imenik;

import java.io.Serializable;

public class Unos implements Comparable<Unos>, Serializable
{
   private Osoba osoba;
   private BrojTelefona broj;
```

```
   public Unos(Osoba osoba, BrojTelefona broj)
   {
      this.osoba = osoba;
      this.broj = broj;
   }

   public int compareTo(Unos unos)
   {
      return osoba.compareTo(unos.getOsoba());
   }

   public String toString()
   {
      return osoba + " " + broj;
   }

   public Osoba getOsoba()
   {
      return osoba;
   }

   public BrojTelefona getBroj()
   {
      return broj;
   }

   public static Unos ucitajUnos()
   {
      return new Unos(Osoba.ucitajOsobu(), BrojTelefona.ucitajBrojTelefona());
   }

}
```

Imenik.java

```
package imenik;

import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.io.Serializable;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;

public class Imenik implements Serializable
{
   private HashMap<Osoba, Unos> imenik = new HashMap<Osoba, Unos>();
   private File fajl = new File("C:/temp/Imenik.bin");

   public Imenik()
   {
      if (fajl.exists())
         try
         {
            ObjectInputStream in = new ObjectInputStream(
new FileInputStream(fajl));
            imenik = (HashMap<Osoba, Unos>) in.readObject();
            in.close();
         } catch (ClassNotFoundException e)
         {
            e.printStackTrace();
            System.exit(1);
         } catch (IOException e)
```

```
         {
            e.printStackTrace();
            System.exit(1);
         }
   }

   public void sacuvaj()
   {
      try
      {
         System.out.println("Cuvanje imenika");
         ObjectOutputStream out = new ObjectOutputStream(
             new FileOutputStream(fajl));
         out.writeObject(imenik);
         System.out.println("Zavrseno");
         out.close();
      } catch (IOException e)
      {
         e.printStackTrace();
         System.exit(1);
      }
   }

   public void dodajUnos(Unos unos)
   {
      imenik.put(unos.getOsoba(), unos);
   }

   public void izlistajUnose()
   {
      // smestamo unose u povezanu listu
      LinkedList<Unos> unosi = new LinkedList<Unos>(imenik.values());
      Collections.sort(unosi);
      for (Unos unos : unosi)
         System.out.println(unos);
   }

   public Unos getUnos(Osoba osoba)
   {
      return imenik.get(osoba);
   }

   public BrojTelefona getBroj(Osoba osoba)
   {
      return getUnos(osoba).getBroj();
   }

}
```

TestImenik.java

```
package imenik;

import java.util.Scanner;

public class TestImenik
{

   public static void main(String[] args)
   {
      Imenik imenik = new Imenik();
      Scanner sc = new Scanner(System.in);

      Osoba neko;

      for (;;)
      {
```

```
        System.out.println("Unesite:\n" + "1 za novi unos u imenik");
        System.out.println("2 da nadjete broj za ime");
        System.out.println("3 da izlistate sve unose imenika "
            + "u abecednom poretku");
        System.out.println("9 za kraj programa");

        int sta = sc.nextInt();

        switch (sta)
        {
           case 1:
              imenik.dodajUnos(Unos.ucitajUnos());
              break;

           case 2:
              neko = Osoba.ucitajOsobu();
              Unos unos = imenik.getUnos(neko);
              if (unos == null)
                 System.out.println("Te osobe nema u imeniku");
              else
                 System.out.println("Broj telefona osobe " + neko + " je "
                     + unos.getBroj());
              break;
           case 3:
              imenik.izlistajUnose();
              break;

           case 9:
              imenik.sacuvaj();
              System.out.println("Kraj programa");
              return;
           default:
              System.out.println("Neispravan unos, pokusajte ponovo");
              break;
        }

     }

  }

}
```

21. *Шах*. Написати конзолну апликацију за играње шаха.

Објашњење:

Боје фигура представљене су константама енумерације *Boja*.
Класа *Polje* описује једно поље шаховске табле. Поље је комбинација слова (које може бити: A, B, C, D, E, F, G или H) и цифре (која може бити: 1, 2, 3, 4, 5, 6, 7 или 8). Пошто ћемо имати колекцију фигура које се налазе на табли, при чему ће најчешћа операција бити провера да ли се нека фигура налази на задатом пољу, у овој имплементацији одлучено је да та колекција буде мапа у коју се смештају парови поље/фигура, где поље представља кључ који се користи за смештање одговарајуће фигуре у колекцију. Из тог разлога, у класи *Polje* предефинисани су методи *equals()* и *hashCode()* на уобичајени начин. Претпоставка је да је поље A1 црно, што нигде није експлицитно написано у коду (јер на екрану не представљамо поља у боји), али је тиме одређен иницијални положај дама, а њиме и краљева, на табли.
Класа *Figura* представља базну класу за описивање шаховских фигура. Инстанцне променљиве класе су боја фигуре и референца на поље шаховске табле на коме се фигура налази. Апстрактни метод *odrediDopustivaPolja()* за дату фигуру враћа колекцију оних поља на која је, у складу са правилима игре, допуштено померити фигуру. Свако допустиво поље је или празно. тј. на њему се не налази фигура, или се на њему налази противничка фигура коју можемо да "поједемо". Када није познато о којој фигури је реч, није могуће одредити ни одговарајућа допустива поља, те је метод зато

декларисан као апстрактан. Метод *pomeriFiguru()*, за померање фигуре на задато поље, није апстрактан, јер је принцип за померање фигуре исти независно од типа коме она припада. Фигура се може померити само на одговарајуће допустиво поље, о чему се води рачуна у тест-класи, те метод *pomeriFiguru()* ради под претпоставком да је тај услов испуњен, односно да је задато поље допустиво за текућу фигуру. Даље, ако је задато (допустиво) поље празно, фигура се уклања из колекције (јер се у колекцију смешта на основу вредности поља, која се управо мења), а затим се поставља нова вредност поља и фигура помоћу ње враћа натраг у колекцију. Ако се на (допустивом) пољу налази противничка фигура, она се трајно уклања из колекције, а текућа фигура се, истим поступком као и малочас, помера на њено место, осим у ситуацији када је противничка фигура краљ и када метод враћа вредност *false* као индикатор краја игре, а померање фигура се не врши. У свим осталим случајевима повратна вредност метода је *true*. Метод *toString()* враћа само информацију о боји фигуре, а информацију о њеном типу дописаће истоимени метод изведене класе и то испред информације о боји.

Из класе *Figura* изведене су класе: *Pesak, Skakac, Kralj, Top, Lovac* и *Dama*.

Шаховска табла представља се на екрану на следећи начин: словима се, слева надесно, обележавају колоне шаховске табле, а цифрама, одоздо нагоре, њене врсте. Беле фигуре се иницијално постављају у врсте означене цифрама 1 и 2, а црне фигуре у врсте означене цифрама 7 и 8. У меморији, шаховска табла представљена је једном 8 × 8 матрицом поља. То што се колоне обележавају цифрама одоздо нагоре има за последицу да је одговарајући индекс поља на екрану обележеног цифром *cifra* у матрици поља једнак (8-*cifra*).

Пешак може да се креће само у једном смеру, ка другом крају табле. Разликује се начин на који се креће када "једе" и када "не једе". Иницијално може да се помери напред за једно или два поља, а иначе једно поље напред када не једе, односно једно поље дијагонално напред када једе противничку фигуру.
У класи *Pesak* дефинисане су две константе, *RASTUCI* и *OPADAJUCI*, које одређују смер кретања пешака у односу на цифре којима су обележене врсте шаховске табле. Вредности ових константи биће ускоро објашњене. "Растући" смер кретања имају пешаци који се иницијално налазе у врсти означеној цифром 2 (бели пешаци), док они из врсте 7 (црни пешаци) имају "опадајући" смер кретања. Метод *odrediDopustivaPolja()* креће од празне колекције поља. Ако је неки пешак дошао до врсте 8 или до врсте 1, тај више нема куд (јер једни се могу кретати од врсте 2 до врсте 8, а други од врсте 7 до врсте 1), па у том случају метод враћа празну колекцију поља. Променљива *kandidat* служи за чување референце на поље које нам је кандидат за допустиво поље. Испитујемо да ли се можемо померити једно поље напред (потребно је да то поље буде празно): дакле, потребно је проверити да ли је, користећи поље *kandidat* као кључ, у мапу фигура које се тренутно налазе на табли смештена нека фигура, па ако то није случај, додати поље *kandidat* у колекцију допустивих поља. Најпре се дохвата референца на мапу фигура, која је статички члан класе *SahovskaTabla* управо из разлога да би јој се могло приступати из класа које представљају фигуре. Затим се одређују индекси врсте и колоне које поље *kandidat* има у матрици поља шаховске табле (која је такође статички члан класе *SahovskaTabla*). Што се тиче индекса колоне, ствар је јасна: индекс колоне текућег поља је *slovo*-'A', па ће то бити и индекс поља кандидата, јер пешак остаје у истој колони. Индекс врсте текућег поља је 8-*cifra*. Даље, ако се пешак креће у смеру "растући", прећи ће на поље које се налази у врсти *cifra*+1, односно поље које у матрици поља има индекс врсте 8-*cifra*-1, док, ако се пешак креће у супротном смеру, прећи ће у врсту *cifra*-1, па ће одговарајући индекс поља бити 8-*cifra*+1. Ако то уопштено запишемо као 8-*cifra*+*smer*, јасно је да за "растући" смер променљива *smer* има вредност -1, док за "опадајући" смер има вредност 1, те су отуда управо тим вредностима иницијализоване одговарајуће константе класе *Pesak*. Провера да ли је у мапу смештен неки објекат коришћењем задате вредности кључа врши се позивом метода *get()* класе *HashMap*<>, који враћа вредност *null* када тај кључ није коришћен за смештање објеката у мапу (или је помоћу њега смештен *null* објекат, што ми нећемо никада радити, тако да нема опасности да дође до забуне око тога). Даље, ако се пешак налази на иницијалној позицији (смер је "растући", а цифра 2 или је смер "опадајући", а цифра 7), а претходном провером је утврђено да је и поље непосредно испред њега празно (па смо га додали у колекцију те

она није празна), испитујемо да ли је празно следеће поље (тј. поље чији је индекс врсте у матрици поља једнак 8-*cifra*+2**smer*), и ако јесте, додајемо и њега у колекцију допустивих поља. Затим проверавамо да ли је пешак у могућности да "једе" (на дијагоналном пољу треба да се налази противничка фигура, тј. фигура друге боје): разликују се случајеви када се пешак налази у крајњој колони (A, односно H), када постоји једно поље кандидат ("десни кандидат" – индекс колоне за један већи, односно "леви кандидат" – индекс колоне за један мањи, респективно) и када се налази у некој од средишњих колона (B-G), када постоје 2 поља кандидата (и "десни" и "леви").

Скакач се креће "у Г", односно 2 поља у једном смеру (било ком: горе, доле, лево, десно), а онда 1 поље у смеру ортогоналном на претходни смер кретања. Математички речено, апсолутна вредност разлике одговарајућих координата новог и полазног поља је: 2 за слова и 1 за цифре или 2 за цифре и 1 за слова.
Краљ може да пређе на произвољно њему суседно поље.

Метод *odrediDopustivaPolja()* је у класама *Skakac* и *Kralj* имплементиран на исти начин: дефинисан је по један низ свих допустивих помака по обе координате (и по слову и по цифри), а затим се у петљи пролази кроз тај низ, рачунају координате поља кандидата додавањем помака на координате текућег поља и, уколико се тиме не излази ван граница шаховске табле, проверава да ли је поље кандидат празно или се на њему налази противничка фигура, па ако то јесте случај, поље се додаје у колекцију допустивих поља.

Топ може да се креће по хоризонтали и по вертикали, ловац само дијагонално, док дама може да се креће и као топ и као ловац.

Метод *odrediDopustivaPolja()* је у класама *Top* и *Lovac* имплементиран коришћењем истог принципа: обрађују се сва 4 могућа смера кретања, почев од поља које је суседно текућем пољу, ка одговарајућем крају табле. Ако је поље празно, додаје се у колекцију допустивих поља и прелази на испитивање следећег поља у смеру који се обрађује. Ако се на пољу налази фигура исте боје, стаје се и прелази на испитивање следећег неиспитаног смера, ако такав постоји. Ако се, пак, на пољу налази противничка фигура, поље се дода у колекцију, а онда се прелази на испитивање следећег неиспитаног смера, ако такав постоји.
У класи *Dama*, у телу метода *odrediDopustivaPolja()* налази се садржај истоимених метода класа *Top* и *Lovac*.

Класа *SahovskaTabla* има две статичке променљиве: мапу фигура које се тренутно налазе на табли и матрицу поља шаховске табле. У конструктору се креирају поља матрице, при чему се, као што је већ речено, због обележавања врста цифрама одоздо нагоре приликом приказа шаховске табле на екрану, у *i*-ту врсту матрице смештају поља која су на екрану означена цифром 8-*i*. Затим се креирају фигуре на одговарајућим пољима и смештају у мапу фигура. Дефинисани су и статички методи за дохватање мапе фигура и дохватање поља када су задати индекси врсте и колоне које оно има у матрици поља. Метод *toString()* враћа прихватљиву *String*-репрезентацију шаховске табле и фигура које се тренутно налазе на њој.

У класи *SahApp*, најпре се креира и приказује шаховска табла са иницијално распоређеним фигурама. Први је на потезу играч који игра белим фигурама. Када променљива *dalje* добије вредност *false*, игра је завршена. Све док игра траје, играч који је на потезу уноси поље са ког жели да помери своју фигуру (при том се проверава валидност уноса), те ако на том пољу нема фигуре, исписује се одговарајућа порука и играч поново уноси поље, као и ако се на унетом пољу налази противничка фигура. Ако је унето поље на коме се налази фигура одговарајуће боје, за ту фигуру се позивом метода *odrediDopustivaPolja()* одређује колекција поља на која ју је могуће померити. Ако је колекција празна, исписује се порука да изабрану фигуру није могуће померити и прелази на поновни унос полазног поља, а иначе се кориснику исписују допустива поља и од њега тражи информација о пољу на које жели да премести изабрану фигуру. Уколико се унето одредишно поље налази у колекцији допустивих поља изабране фигуре, позива се метод *pomeriFiguru()* и долази до смене

играча на потезу, а иначе се исписује порука о недопустивости потеза. Метод враћа вредност која одређује да ли се игра наставља или је крај (крај је ако је тим потезом нападнут противнички краљ). На крају игре, поражени краљ остаје на шаховској табли.

Решење:

Boja.java

```
package sah;

public enum Boja
{
   BELA, CRNA
}
```

Polje.java

```
package sah;

public class Polje
{

   private char slovo;
   private int cifra;

   public Polje(int cifra, char slovo)
   {
      this.slovo = slovo;
      this.cifra = cifra;
   }

   public boolean equals(Object o)
   {
      if (!(o instanceof Polje))
         return false;
      Polje p = (Polje) o;
      return p.cifra == cifra && p.slovo == slovo;
   }

   public int hashCode()
   {
      return 11 * cifra + 13 * slovo;
   }

   public int getCifra()
   {
      return cifra;
   }

   public char getSlovo()
   {
      return slovo;
   }

   public String toString()
   {
      return "" + slovo + cifra;
   }

}
```

Figura.java

```
package sah;
```

```
import java.util.ArrayList;

public abstract class Figura
{
   private Boja boja;
   private Polje polje;

   public Figura(Boja boja, Polje polje)
   {
      this.boja = boja;
      this.polje = polje;
   }

   public abstract ArrayList<Polje> odrediDopustivaPolja();

   /*
    * vraca true ako igra nije zavrsena a false inace, tj. kada se
    * ovim potezom napadne protivnicki kralj
    */
   public boolean pomeriFiguru(Polje polje)
   {
      /*
       * ako tekuca figura moze da se pomeri na dato polje,
       * moze jer je to polje ili prazno ili se na njemu nalazi
       * protivnicka figura koju cemo upravo da "pojedemo"
       */
      Figura f = SahovskaTabla.getFigureNaTabli().get(polje);
      if (f == null)
      {
         SahovskaTabla.getFigureNaTabli().remove(this.polje);
         this.polje = polje;
         SahovskaTabla.getFigureNaTabli().put(polje, this);
      } else
      {
         /*
          * "jedemo" figuru sa tog polja pa se mi tu premestamo
          * ako je u pitanju kralj, igra se završava bez pomeranja figura
          */
         if (f instanceof Kralj)
            return false;
         f.polje = null;
         SahovskaTabla.getFigureNaTabli().remove(polje);

         SahovskaTabla.getFigureNaTabli().remove(this.polje);
         this.polje = polje;
         SahovskaTabla.getFigureNaTabli().put(polje, this);
      }

      return true;
   }

   public Polje getPolje()
   {
      return polje;
   }

   public Boja getBoja()
   {
      return boja;
   }

   public String toString()
   {
      return (boja == Boja.BELA) ? "b" : "c";
   }
```

```
}
```

Pesak.java

```
package sah;

import java.util.ArrayList;
import java.util.HashMap;

public class Pesak extends Figura
{
   // smer kretanja u odnosu na cifre polja sahovske table
   final static int RASTUCI = -1;
   final static int OPADAJUCI = +1;

   private int smer;

   public Pesak(Boja boja, Polje polje)
   {
      super(boja, polje);
      smer = (polje.getCifra() == 2) ? RASTUCI : OPADAJUCI;
   }

   public ArrayList<Polje> odrediDopustivaPolja()
   {
      ArrayList<Polje> dopustivaPolja = new ArrayList<Polje>();
      Polje p = getPolje();
      char slovo = p.getSlovo();
      int cifra = p.getCifra();

      if (cifra == 8 || cifra == 1)
         return dopustivaPolja;

      // figura se nalazi na polju SahovskaTabla.tabla[8-cifra][slovo-'A']
      HashMap<Polje, Figura> figureNaTabli = SahovskaTabla.getFigureNaTabli();
      // slucaj pomeranja za jedno polje, bez "jela"
      Polje kandidat = SahovskaTabla.getPolje(8 - cifra + smer, slovo - 'A');
      if (figureNaTabli.get(kandidat) == null)
         dopustivaPolja.add(kandidat);

      // slucaj pomeranja sa inicijalne pozicije za 2 polja
      if (smer == RASTUCI && cifra == 2 && !dopustivaPolja.isEmpty())
      {
         kandidat = SahovskaTabla.getPolje(8 - cifra - 2, slovo - 'A');
         if (figureNaTabli.get(kandidat) == null)
            dopustivaPolja.add(kandidat);
      }

      if (smer == OPADAJUCI && cifra == 7 && !dopustivaPolja.isEmpty())
      {
         kandidat = SahovskaTabla.getPolje(8 - cifra + 2, slovo - 'A');
         if (figureNaTabli.get(kandidat) == null)
            dopustivaPolja.add(kandidat);
      }

      Figura f;
      /*
       * slucaj kada moze da "jede"
       * kandidati: ako je slovo 'A' ima samo kandidata 'B'
       * ako je slovo 'H' ima samo kandidata 'G'
       * a inace ima 2 kandidata: slovo ispred i slovo iza
       */
      if (slovo == 'A')
      {
         // nalazimo se na SahovskaTabla.tabla[8-cifra]['A'-'A']
         kandidat = SahovskaTabla.getPolje(8 - cifra + smer, 'B' - 'A');
         if ((f = figureNaTabli.get(kandidat)) != null &&
```

```
f.getBoja() != getBoja())
          dopustivaPolja.add(kandidat);
     } else if (slovo == 'H')
     {
        // nalazimo se na SahovskaTabla.tabla[8-cifra]['H'-'A']
        kandidat = SahovskaTabla.getPolje(8 - cifra + smer, 'G' - 'A');
        if ((f = figureNaTabli.get(kandidat)) != null &&
f.getBoja() != getBoja())
          dopustivaPolja.add(kandidat);
     } else
     {
        // "levi" kandidat
        kandidat = SahovskaTabla.getPolje(8 - cifra + smer, slovo - 'A' - 1);
        if ((f = figureNaTabli.get(kandidat)) != null &&
f.getBoja() != getBoja())
          dopustivaPolja.add(kandidat);

        // "desni" kandidat
        kandidat = SahovskaTabla.getPolje(8 - cifra + smer, slovo - 'A' + 1);
        if ((f = figureNaTabli.get(kandidat)) != null &&
f.getBoja() != getBoja())
          dopustivaPolja.add(kandidat);
     }

     return dopustivaPolja;

   }

   public String toString()
   {
     return "P" + super.toString();
   }

}
```

Skakac.java

```
package sah;

import java.util.ArrayList;
import java.util.HashMap;

public class Skakac extends Figura
{
   public Skakac(Boja boja, Polje polje)
   {
     super(boja, polje);
   }

   public ArrayList<Polje> odrediDopustivaPolja()
   {
     ArrayList<Polje> dopustivaPolja = new ArrayList<Polje>();
     Polje p = getPolje();
     char slovo = p.getSlovo();
     int cifra = p.getCifra();

     int pomeraji[][] = { { -2, 1 }, { -2, -1 }, { -1, 2 }, { 1, 2 },
         { -1, -2 }, { 1, -2 }, { 2, 1 }, { 2, -1 } };

     int slovoK;
     int cifraK;

     for (int[] pom : pomeraji)
     {
        slovoK = slovo + pom[0];
        cifraK = cifra + pom[1];
```

```
        // ako nismo izasli iz granica table
        if (slovoK >= 'A' && slovoK <= 'H' && cifraK >= 1 && cifraK <= 8)
        {
           /*
            * mozemo da se pomerimo na polje na kome nema figure ili je tamo
            * protivnicka figura
            */
           HashMap<Polje, Figura> figureNaTabli =
             SahovskaTabla.getFigureNaTabli();
           Polje kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
           Figura f;
           if ((f = figureNaTabli.get(kandidat)) == null
               || f.getBoja() != getBoja())
             dopustivaPolja.add(kandidat);
        }
      }

      return dopustivaPolja;
   }

   public String toString()
   {
      return "S" + super.toString();
   }

}
```

Kralj.java

```
package sah;

import java.util.ArrayList;
import java.util.HashMap;

public class Kralj extends Figura
{
   public Kralj(Boja boja, Polje polje)
   {
      super(boja, polje);
   }

   public ArrayList<Polje> odrediDopustivaPolja()
   {
      ArrayList<Polje> dopustivaPolja = new ArrayList<Polje>();
      Polje p = getPolje();
      char slovo = p.getSlovo();
      int cifra = p.getCifra();

      int pomeraji[][] = { { -1, -1 }, { -1, 0 }, { -1, 1 }, { 0, -1 }, { 0, 1 },
          { 1, -1 }, { 1, 0 }, { 1, 1 } };

      int slovoK;
      int cifraK;

      for (int[] pom : pomeraji)
      {
         slovoK = slovo + pom[0];
         cifraK = cifra + pom[1];

         // ako nismo izasli iz granica table
         if (slovoK >= 'A' && slovoK <= 'H' && cifraK >= 1 && cifraK <= 8)
         {
            /*
             * mozemo da se pomerimo na polje na kome nema figure ili je tamo
             * protivnicka figura
             */
            HashMap<Polje, Figura> figureNaTabli =
```

```
            SahovskaTabla.getFigureNaTabli();
          Polje kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
          Figura f;
          if ((f = figureNaTabli.get(kandidat)) == null
              || f.getBoja() != getBoja())
            dopustivaPolja.add(kandidat);
        }
      }

      return dopustivaPolja;

   }

   public String toString()
   {
      return "K" + super.toString();
   }

}
```

Top.java

```
package sah;

import java.util.ArrayList;
import java.util.HashMap;

public class Top extends Figura
{
   public Top(Boja boja, Polje polje)
   {
      super(boja, polje);
   }

   public ArrayList<Polje> odrediDopustivaPolja()
   {
      ArrayList<Polje> dopustivaPolja = new ArrayList<Polje>();
      Polje p = getPolje();
      char slovo = p.getSlovo();
      int cifra = p.getCifra();

      HashMap<Polje, Figura> figureNaTabli = SahovskaTabla.getFigureNaTabli();
      Polje kandidat;
      Figura f;

      // top moze da se krece u 4 smera
      // prvi smer: po vertikali, rastuce
      for (int cifraK = cifra + 1; cifraK <= 8; cifraK++)
      {
         kandidat = SahovskaTabla.getPolje(8-cifraK, slovo - 'A');
         if ((f = figureNaTabli.get(kandidat)) == null)
            dopustivaPolja.add(kandidat);
         else if (f.getBoja() != getBoja())
         {
            dopustivaPolja.add(kandidat);
            break;
         } else
            break;
      }

      // drugi smer: po vertikali, opadajuce

      for (int cifraK = cifra - 1; cifraK >= 1; cifraK--)
      {
         kandidat = SahovskaTabla.getPolje(8 - cifraK, slovo - 'A');
         if ((f = figureNaTabli.get(kandidat)) == null)
            dopustivaPolja.add(kandidat);
```

```
        else if (f.getBoja() != getBoja())
        {
           dopustivaPolja.add(kandidat);
           break;
        } else
           break;
     }

     // treci smer: po horizontali, rastuce
     for (int slovoK = slovo + 1; slovoK <= 'H'; slovoK++)
     {
        kandidat = SahovskaTabla.getPolje(8 - cifra, slovoK - 'A');
        if ((f = figureNaTabli.get(kandidat)) == null)
           dopustivaPolja.add(kandidat);
        else if (f.getBoja() != getBoja())
        {
           dopustivaPolja.add(kandidat);
           break;
        } else
           break;
     }

     // cetvrti smer: po horizontali, opadajuce
     for (int slovoK = slovo - 1; slovoK >= 'A'; slovoK--)
     {
        kandidat = SahovskaTabla.getPolje(8 - cifra, slovoK - 'A');
        if ((f = figureNaTabli.get(kandidat)) == null)
           dopustivaPolja.add(kandidat);
        else if (f.getBoja() != getBoja())
        {
           dopustivaPolja.add(kandidat);
           break;
        } else
           break;
     }

     return dopustivaPolja;

   }

   public String toString()
   {
     return "T" + super.toString();
   }

}
```

Lovac.java

```
package sah;

import java.util.ArrayList;
import java.util.HashMap;

public class Lovac extends Figura
{
   public Lovac(Boja boja, Polje polje)
   {
     super(boja, polje);
   }

   public ArrayList<Polje> odrediDopustivaPolja()
   {
     ArrayList<Polje> dopustivaPolja = new ArrayList<Polje>();
     Polje p = getPolje();
     char slovo = p.getSlovo();
     int cifra = p.getCifra();
```

```
HashMap<Polje, Figura> figureNaTabli = SahovskaTabla.getFigureNaTabli();
Polje kandidat;
Figura f;

// lovac moze da se krece u 4 dijagonalna smera

// prvi smer: rastuce i po horizontali i po vertikali
for (int cifraK = cifra + 1, slovoK = slovo + 1; cifraK <= 8
    && slovoK <= 'H'; cifraK++, slovoK++)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// drugi smer: rastuce po vertikali, opadajuce po horizontali
for (int cifraK = cifra + 1, slovoK = slovo - 1; cifraK <= 8
    && slovoK >= 'A'; cifraK++, slovoK--)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// treci smer: opadajuce po vertikali, rastuce po horizontali
for (int cifraK = cifra - 1, slovoK = slovo + 1; cifraK >= 1
    && slovoK <= 'H'; cifraK--, slovoK++)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// cetvrti smer: opadajuce i po horizontali i po vertikali
for (int cifraK = cifra - 1, slovoK = slovo - 1; cifraK >= 1
    && slovoK >= 'A'; cifraK--, slovoK--)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}
```

```
      return dopustivaPolja;

   }

   public String toString()
   {
      return "L" + super.toString();
   }

}
```

Dama.java

```
package sah;

import java.util.ArrayList;
import java.util.HashMap;

public class Dama extends Figura
{
   public Dama(Boja boja, Polje polje)
   {
      super(boja, polje);
   }

   public ArrayList<Polje> odrediDopustivaPolja()
   {
      // kombinacija topa i lovca
      ArrayList<Polje> dopustivaPolja = new ArrayList<Polje>();
      Polje p = getPolje();
      char slovo = p.getSlovo();
      int cifra = p.getCifra();

      HashMap<Polje, Figura> figureNaTabli = SahovskaTabla.getFigureNaTabli();
      Polje kandidat;
      Figura f;

      // top
      // top, prvi smer: po vertikali, rastuce
      for (int cifraK = cifra + 1; cifraK <= 8; cifraK++)
      {
         kandidat = SahovskaTabla.getPolje(8 - cifraK, slovo - 'A');
         if ((f = figureNaTabli.get(kandidat)) == null)
            dopustivaPolja.add(kandidat);
         else if (f.getBoja() != getBoja())
         {
            dopustivaPolja.add(kandidat);
            break;
         } else
            break;
      }

      // top, drugi smer: po vertikali, opadajuce
      for (int cifraK = cifra - 1; cifraK >= 1; cifraK--)
      {
         kandidat = SahovskaTabla.getPolje(8 - cifraK, slovo - 'A');
         if ((f = figureNaTabli.get(kandidat)) == null)
            dopustivaPolja.add(kandidat);
         else if (f.getBoja() != getBoja())
         {
            dopustivaPolja.add(kandidat);
            break;
         } else
            break;
      }

      // top, treci smer: po horizontali, rastuce
```

```
for (int slovoK = slovo + 1; slovoK <= 'H'; slovoK++)
{
   kandidat = SahovskaTabla.getPolje(8 - cifra, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// top, cetvrti smer: po horizontali, opadajuce
for (int slovoK = slovo - 1; slovoK >= 'A'; slovoK--)
{
   kandidat = SahovskaTabla.getPolje(8 - cifra, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// lovac
// lovac, prvi smer: rastuce i po horizontali i po vertikali
for (int cifraK = cifra + 1, slovoK = slovo + 1; cifraK <= 8
    && slovoK <= 'H'; cifraK++, slovoK++)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// lovac, drugi smer: rastuce po vertikali, opadajuce po horizontali
for (int cifraK = cifra + 1, slovoK = slovo - 1; cifraK <= 8
    && slovoK >= 'A'; cifraK++, slovoK--)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
      dopustivaPolja.add(kandidat);
      break;
   } else
      break;
}

// lovac, treci smer: opadajuce po vertikali, rastuce po horizontali
for (int cifraK = cifra - 1, slovoK = slovo + 1; cifraK >= 1
    && slovoK <= 'H'; cifraK--, slovoK++)
{
   kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
   if ((f = figureNaTabli.get(kandidat)) == null)
      dopustivaPolja.add(kandidat);
   else if (f.getBoja() != getBoja())
   {
```

```
            dopustivaPolja.add(kandidat);
            break;
         } else
            break;
      }

      // lovac, cetvrti smer: opadajuce i po horizontali i po vertikali
      for (int cifraK = cifra - 1, slovoK = slovo - 1; cifraK >= 1
          && slovoK >= 'A'; cifraK--, slovoK--)
      {
         kandidat = SahovskaTabla.getPolje(8 - cifraK, slovoK - 'A');
         if ((f = figureNaTabli.get(kandidat)) == null)
            dopustivaPolja.add(kandidat);
         else if (f.getBoja() != getBoja())
         {
            dopustivaPolja.add(kandidat);
            break;
         } else
            break;
      }

      return dopustivaPolja;

   }

   public String toString()
   {
      return "D" + super.toString();
   }

}
```

SahovskaTabla.java

```
package sah;

import java.util.HashMap;

public class SahovskaTabla
{
   private static HashMap<Polje, Figura> figureNaTabli =
      new HashMap<Polje, Figura>();
   private static Polje[][] tabla = new Polje[8][8];


   public SahovskaTabla()
   {
      // vrste su oznacene ciframa, a kolone slovima
      for (int i = 0; i < 8; i++)
         for (int j = 0; j < 8; j++)
            tabla[i][j] = new Polje(8-i, (char) ('A' + j));

      /*
       * Kreiraju se figure i postavljaju na tablu;
       * u vrstama 1 i 2 bele, a u vrstama 7 i 8 crne
       */
      for (int i = 0; i < 8; i++)
      {
         // vrsta 1: 8 belih pesaka
         figureNaTabli.put(tabla[6][i], new Pesak(Boja.BELA, tabla[6][i]));
         // vrsta 7: 8 crnih pesaka
         figureNaTabli.put(tabla[1][i], new Pesak(Boja.CRNA, tabla[1][i]));
      }

      // Beli topovi: polja A1 i H1
      figureNaTabli.put(tabla[7][0], new Top(Boja.BELA, tabla[7][0]));
      figureNaTabli.put(tabla[7][7], new Top(Boja.BELA, tabla[7][7]));
```

```
   // Crni topovi: polja A8 i H8
   figureNaTabli.put(tabla[0][0], new Top(Boja.CRNA, tabla[0][0]));
   figureNaTabli.put(tabla[0][7], new Top(Boja.CRNA, tabla[0][7]));

   // Beli skakaci: polja B1 i G1
   figureNaTabli.put(tabla[7][1], new Skakac(Boja.BELA, tabla[7][1]));
   figureNaTabli.put(tabla[7][6], new Skakac(Boja.BELA, tabla[7][6]));

   // Crni skakaci: polja B8 i G8
   figureNaTabli.put(tabla[0][1], new Skakac(Boja.CRNA, tabla[0][1]));
   figureNaTabli.put(tabla[0][6], new Skakac(Boja.CRNA, tabla[0][6]));

   // Beli lovci: polja C1 i F1
   figureNaTabli.put(tabla[7][2], new Lovac(Boja.BELA, tabla[7][2]));
   figureNaTabli.put(tabla[7][5], new Lovac(Boja.BELA, tabla[7][5]));

   // Crni lovci: polja C8 i F8
   figureNaTabli.put(tabla[0][2], new Lovac(Boja.CRNA, tabla[0][2]));
   figureNaTabli.put(tabla[0][5], new Lovac(Boja.CRNA, tabla[0][5]));

   // Bela kraljica: belo polje D1
   figureNaTabli.put(tabla[7][3], new Dama(Boja.BELA, tabla[7][3]));

   // Crna kraljica: crno polje D8
   figureNaTabli.put(tabla[0][3], new Dama(Boja.CRNA, tabla[0][3]));

   // Beli kralj: polje E1
   figureNaTabli.put(tabla[7][4], new Kralj(Boja.BELA, tabla[7][4]));

   // Crni kralj: polje E8
   figureNaTabli.put(tabla[0][4], new Kralj(Boja.CRNA, tabla[0][4]));

}

public static HashMap<Polje, Figura> getFigureNaTabli()
{
   return figureNaTabli;
}

public static Polje getPolje(int i, int j)
{
   return tabla[i][j];
}

public String toString()
{
   StringBuffer str = new StringBuffer();
   Figura fig = null;
   str.append("    ");
   for (int i = 0; i < 8; i++)
      str.append((char) ('A' + i) + "  ");
   str.append("\n  -");
   for (int i = 0; i < 8; i++)
      str.append("---");
   str.append("\n");
   for (int i = 0; i < 8; i++)
   {
      str.append((8 - i) + "| ");
      for (int j = 0; j < 8; j++)
         /*
          * Prolazi se kroz kolekciju figura na tabli i proverava da li je neka
          * figura na tekucem polju i koja
          */
         if ((fig = figureNaTabli.get(tabla[i][j])) != null)
            str.append(fig + " ");
         else
```

```
            str.append("   ");
        str.append("|" + (8 - i) + "\n");
      }
      str.append("  -");
      for (int i = 0; i < 8; i++)
        str.append("---");
      str.append("\n    ");
      for (int i = 0; i < 8; i++)
        str.append((char) ('A' + i) + "  ");

      return str.toString();
    }

}
```

SahApp.java

```
package sah;

import java.util.ArrayList;
import java.util.Scanner;

public class SahApp
{
   public static void main(String[] args)
   {
      SahovskaTabla sahovskaTabla = new SahovskaTabla();
      System.out.println(sahovskaTabla);

      Scanner sc = new Scanner(System.in);
      boolean dalje = true;
      Boja igracNaPotezu = Boja.BELA;

      while (true)
      {
         System.out.print("Sa kog polja zelite da pomerite figuru:");
         String poljeS = sc.next();
         char slovo = Character.toUpperCase(poljeS.charAt(0));
         int cifra = poljeS.charAt(1) - '0';
         if (slovo < 'A' || slovo > 'H' || cifra < 1 || cifra > 8)
         {
            System.out.println("Neispravno polje");
            continue;
         }

         Figura f = SahovskaTabla.getFigureNaTabli().get(
             SahovskaTabla.getPolje(8 - cifra, slovo - 'A'));

         if (f == null)
         {
            System.out.println("Na polju nema figure");
            continue;
         }

         if (f.getBoja() != igracNaPotezu)
         {
            System.out.println("Na potezu je " + igracNaPotezu + " figura!");
            continue;
         }

         ArrayList<Polje> dopustivaPolja = f.odrediDopustivaPolja();
         if (dopustivaPolja.isEmpty())
         {
            System.out.println("Tu figuru nije moguce pomeriti");
            continue;
         }
         for (Polje p : dopustivaPolja)
```

```
            System.out.println(p);

         System.out.print("Na koje polje:");
         poljeS = sc.next();
         cifra = poljeS.charAt(1) - '0';
         slovo = Character.toUpperCase(poljeS.charAt(0));
         if (slovo < 'A' || slovo > 'H' || cifra < 1 || cifra > 8)
         {
            System.out.println("Neispravno polje");
            continue;
         }

         Polje p=SahovskaTabla.getPolje(8 - cifra, slovo - 'A');
         if (dopustivaPolja.contains(p))
         {
            dalje = f.pomeriFiguru(p);
            if (dalje == false)
               break;
            igracNaPotezu = (igracNaPotezu == Boja.BELA) ? Boja.CRNA : Boja.BELA;
         } else
            System.out.println("Nedopustiv potez");

         System.out.println(sahovskaTabla);
      }

      System.out.println("Kraj! Pobedio je igrac cija je boja " + igracNaPotezu);
   }
}
```

22. Шах, GUI. Написати аплет за играње шаха.

Објашњење:

Дефиниције класа *Pesak*, *Skakac*, *Kralj*, *Top*, *Lovac* и *Dama* су исте као у задатку ???, те су овде изостављене.

Дефиниција класе *Figura* је такође непромењена. У методу *pomeriFiguru()* не врши се провера да ли се одредишно поље налази у колекцији допустивих поља текуће фигуре, јер се метод позива само ако је тај услов испуњен, што се проверава у позивајућем методу *potez()* класе *SahovskaTabla*.

Класе *Polje* и *SahovskaTabla* допуњене су детаљима који омогућују њихов графички приказ на површи аплета.

Класа *Polje* изведена је из класе *JPanel*, а додате су и инстанцне промењљиве *labela*, која служи за приказ сличице фигуре која се налази на пољу када оно није празно, и *mouseIn*, индикаторска променљива која има вредност *true* када се курсор миша налази изнад површи поља. У конструктору је додат позив конструктора суперкласе. Боја позадине поља поставља се тако да поље A1 буде сиво (не црно јер се онда не би виделе црне фигуре), а онда се наизменично смењују бела и сива поља. Величина поља је нешто већа од величине лабеле, како сличица фигуре не би у потпуности прекрила поље. Пољу се придружује и ослушкивач догађаја мишем, који је објекат угњеждене класе *PoljeMouseListener*. Методом *setIcon()* на лабелу се поставља сличица фигуре. Метод *paint()* позива истоимени метод суперкласе, који исцртава поље на уобичајени начин, а ако се изнад поља налази курсор миша, додатно се исцртава и плави правоугаоник по његовој ивици. Угњеждена класа *PoljeMouseListener* изведена је из класе *MouseAdapter* и у њој су предефинисани методи *mouseEntered(), mouseExited()* и *mouseClicked()*. Сва 3 метода најпре проверавају да ли је игра у току, па ако јесте, *mouseEntered()* поставља вредност промењљиве *mouseIn* на *true* и поново исцртава поље, *mouseExited()* поставља вредност промењљиве на *false* и поново исцртава поље, док *mouseClicked()* позива статички метод *potez()* класе *SahovskaTabla* који обрађује потезе играча. Када је игра завршена, поље не реагује на догађаје мишем, те ова 3 метода у том случају не раде ништа.

Класа *SahovskaTabla* такође је изведена из класе *JPanel* и додате су инстанцне промењиве:

- *prvo* – да ли се ради о првом (изворишном) или другом (одредишном) пољу потеза. Иницијално, ради се о првом пољу потеза.
- *saSlovo* и *saCifra* – слово и цифра изворишног поља потеза
- *igracNaPotezu* – боја фигура играча који је тренутно на потезу
- *nijeKraj* – индикаторска променљива која добија вредност *false* када се игра заврши
- *app* – објекат који представља наш аплет. Приступ овом објекту из класе *SahovskaTabla* неопходан је због исписивања одговарајућих порука кориснику у статусној линији аплета.
- лабеле *bnp, cnp* (бели на потезу, црни на потезу) и панел *igracPanel* – служе за приказ сличице са десне стране шаховске табле која указује на то који играч је тренутно на потезу

Све променљиве којима се приступа из статичког метода *potez()* ове класе, као и из метода класе *Figura* или из ње изведених класа, декларисане су као статичке. Конструктор као аргумент прима објекат аплет којим иницијализује одговарајућу инстанцну променљиву. Панел *tablaPanel* има *GridLayout manager* са 10 врста и 10 колона (8 за поља шаховске табле и по 1 са сваке стране за одговарајућу ознаку, тј. ознаку врсте или колоне). Изнад и испод матрице поља поставља се панел са ознакама колона – панел са 8 лабела на којима су исписана одговарајућа слова. Лабеле су типа угњеждене класе *BLabel*, изведене из класе *JLabel*, са центрираним хоризонталним и вертикалним поравнањем. Креирање панела са ознакама колона издвојено је у посебан метод *nizSlova()*. Испред и иза сваке врсте шаховских поља додаје се лабела са ознаком броја врсте. Фигуре се креирају и додају у мапу фигура које се налазе на шаховској табли на исти начин као у задатку ???. За сваку фигуру додата је по једна наредба којом се сличица фигуре поставља на одговарајуће поље на површи аплета. Креирају се лабеле *bnp* и *cnp* које служе за приказ играча који је тренутно на потезу. Свака од ове две лабеле се, одговарајућим вертикалним поравнањем, поставља уз страну играча на ког се односи. У сваком тренутку биће видљива само она која се односи на играча који је тренутно на потезу. Обе лабеле смештају се у панел *igracPanel*. Коначно, панели *tablaPanel* и *igracPanel* се, редом, коришћењем *FlowLayout manager*-а додају на површ аплета. Метод *promenaIgracaNaPotezu(),* у зависности од вредности променљиве *igracNaPotezu* чини видљивом одговарајућу лабелу из панела *igracPanel*. Овај метод позива се из метода *potez()* када дође до смене играча на потезу. Метод *potez()* састоји се из две целине: прва се односи на реаговање када је корисник кликнуо на изворишно поље потеза (променљива *prvo* има вредност *true*), а друга на реаговање када је корисник кликнуо на одредишно поље потеза (променљива *prvo* има вредност *false*). Када је корисник кликнуо на изворишно поље потеза, проверава се да ли се на том пољу налази фигура. Ако нема фигуре на пољу, у статусној линији аплета корисник се обавештава о томе и, без промене вредности променљиве *prvo,* завршава извршавање метода, тако да ће, када корисник наредни пут буде кликнуо на неко поље, то поље бити потенцијално изворишно поље потеза. Уколико се на пољу налази фигура противничког играча, исписује се порука о томе у статусној линији аплета и завршава извршавање метода, као и у малопређашњем случају. Уколико је на пољу фигура играча који је на потезу, то поље се прихвата за изворишно поље потеза (у променљивама *saSlovo* и *saCifra* памте се координате поља и променљива *prvo* добија вредност *false*) након чега се завршава извршавање метода. Ако је корисник кликнуо на неко поље у тренутку када је вредност променљиве *prvo* једнака *false*, проверава се да ли то поље може да буде одредишно поље потеза чије је изворишно поље одређено променљивама *saSlovo* и *saCifra*. Најпре се одређује која фигура се налази на изворишном пољу (нека фигура играча који је на потезу се свакако налази на том пољу, јер метод *potez()* иначе не би поставио вредност променљиве *prvo* на *false*, а тај метод је једини који то може учинити), а затим и колекција допустивих одредишних поља за ту фигуру. Ако је колекција празна, исписује се порука о немогућности померања изабране фигуре, вредност променљиве *prvo* се поставља на *true* (очекује се избор новог изворишног поља) и завршава извршавање метода. Ако се изабрано одредишно поље (поље на које је корисник управо кликнуо) налази у колекцији допустивих поља, врши се померање фигуре са изворишног на одредишно поље, при чему се на изворишно поље не поставља никаква сличица (*setIcon(null)*), оно постаје празно, док се на одредишно поље поставља сличица фигуре. За ову операцију неопходно је имати име те сличице. Имена сличица су таква да се састоје из два дела: први део је слово 'б' или 'ц' које одређује боју фигуре ('б' – бела, 'ц' – црна) док је други део име класе којој фигура припада. Име класе којој фигура припада добијено је на следећи начин: позивом метода

getClass() добијемо одговарајући *Class* објекат, а затим позивом метода *getSimpleName()* за овај објекат долазимо до жељеног имена класе. По успешном окончању потеза, променљива *prvo* добија вредност *true*. Када се одредишно поље не налази у колекцији допустивих поља за фигуру, исписује се порука о недопустивости потеза и вредност променљиве *prvo* поставља на *true*. Ако је вредност коју је вратио метод за померање фигуре *false*, то значи да је у том потезу нападнут противнички краљ, те се исписује порука о томе да је игра завршена и који играч је победио (након тога, нова вредност променљиве *nijeKraj* узрокује нереаговање поља на догађаје мишем). Поражени краљ остаје на табли.

Дефиниција класе *SahGUI* је прилично једноставна. Креира се шаховска табла и у методу *createGUI()* поставља на површ аплета. У методу *init()*, врши се иницијализација аплета, на уобичајени начин.

Решење:

Boja.java

Polje.java

```
package sah;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.Icon;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Polje extends JPanel
{
   private char slovo;
   private int cifra;

   private JLabel labela;
   private boolean mouseIn = false;

   public Polje(int cifra, char slovo)
   {
      super();
      this.slovo = slovo;
      this.cifra = cifra;

      if ((cifra + slovo - 'A') % 2 == 1)
         setBackground(Color.gray);
      else
         setBackground(Color.white);
      setPreferredSize(new Dimension(42, 42));
      labela = new JLabel();
      labela.setPreferredSize(new Dimension(32, 32));
      add(labela);
      addMouseListener(new PoljeMouseListener());
   }

   void setIcon(Icon i)
   {
      labela.setIcon(i);
   }

   public void paint(Graphics g)
```

```
   {
      super.paint(g);
      if (mouseIn)
      {
         g.setColor(Color.blue);
         g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);
      }
   }

   public boolean equals(Object o)
   {
      if (!(o instanceof Polje))
         return false;
      Polje p = (Polje) o;
      return p.cifra == cifra && p.slovo == slovo;
   }

   public int hashCode()
   {
      return 11 * cifra + 13 * slovo;
   }

   public int getCifra()
   {
      return cifra;
   }

   public char getSlovo()
   {
      return slovo;
   }

   public String toString()
   {
      return "" + slovo + cifra;
   }

   class PoljeMouseListener extends MouseAdapter
   {
      public void mouseEntered(MouseEvent e)
      {
         if (SahovskaTabla.getNijeKraj())
         {
            mouseIn = true;
            repaint();
         }
      }

      public void mouseExited(MouseEvent e)
      {
         if (SahovskaTabla.getNijeKraj())
         {
            mouseIn = false;
            repaint();
         }
      }

      public void mouseClicked(MouseEvent e)
      {
         if (SahovskaTabla.getNijeKraj())
            SahovskaTabla.potez(cifra, slovo);
      }

   }

}
```

Figura.java

Pesak.java

Skakac.java

Kralj.java

Top.java

Lovac.java

Dama.java

SahovskaTabla.java

```
package sah;

import javax.swing.*;

import java.awt.*;
import java.util.*;

public class SahovskaTabla extends JPanel
{
   private static HashMap<Polje, Figura> figureNaTabli =
new HashMap<Polje, Figura>();
private static Polje[][] tabla = new Polje[8][8];

   private static boolean prvo = true; // da li je u pitanju prvo (odredisno)
                                       // polje poteza
   private static char saSlovo;
   private static int saCifra;

   private static Boja igracNaPotezu = Boja.BELA;

   private static boolean nijeKraj = true;

   private static SahGUI app = null;
   private static JLabel bnp, cnp; // beli na potezu, crni na potezu
   private JPanel igracPanel;

   public SahovskaTabla(SahGUI applet)
   {
      super();
      app = applet;

      JPanel tablaPanel = new JPanel();
      tablaPanel.setLayout(new GridLayout(10, 10));
      nizSlova(tablaPanel);
      for (int i = 0; i < 8; i++)
      {
         tablaPanel.add(new BLabel((8 - i) + ""));
         for (int j = 0; j < 8; j++)
         {
            tabla[i][j] = new Polje(8 - i, (char) ('A' + j));
            tablaPanel.add(tabla[i][j]);
         }
         tablaPanel.add(new BLabel((8 - i) + ""));
      }
      nizSlova(tablaPanel);

      /*
       * Kreiraju se figure i postavljaju na tablu;
       * u vrstama 1 i 2 bele, a u vrstama 7 i 8 crne
       */
      for (int i = 0; i < 8; i++)
```

```
{
   // vrsta 2: 8 belih pesaka
   figureNaTabli.put(tabla[6][i], new Pesak(Boja.BELA, tabla[6][i]));
   tabla[6][i].setIcon(new ImageIcon("images/bPesak.gif"));

   // vrsta 7: 8 crnih pesaka
   figureNaTabli.put(tabla[1][i], new Pesak(Boja.CRNA, tabla[1][i]));
   tabla[1][i].setIcon(new ImageIcon("images/cPesak.gif"));
}

// Beli topovi: polja A1 i H1
figureNaTabli.put(tabla[7][0], new Top(Boja.BELA, tabla[7][0]));
tabla[7][0].setIcon(new ImageIcon("images/bTop.gif"));
figureNaTabli.put(tabla[7][7], new Top(Boja.BELA, tabla[7][7]));
tabla[7][7].setIcon(new ImageIcon("images/bTop.gif"));

// Crni topovi: polja A8 i H8
figureNaTabli.put(tabla[0][0], new Top(Boja.CRNA, tabla[0][0]));
tabla[0][0].setIcon(new ImageIcon("images/cTop.gif"));
figureNaTabli.put(tabla[0][7], new Top(Boja.CRNA, tabla[0][7]));
tabla[0][7].setIcon(new ImageIcon("images/cTop.gif"));

// Beli skakaci: polja B1 i G1
figureNaTabli.put(tabla[7][1], new Skakac(Boja.BELA, tabla[7][1]));
tabla[7][1].setIcon(new ImageIcon("images/bSkakac.gif"));
figureNaTabli.put(tabla[7][6], new Skakac(Boja.BELA, tabla[7][6]));
tabla[7][6].setIcon(new ImageIcon("images/bSkakac.gif"));

// Crni skakaci: polja B8 i G8
figureNaTabli.put(tabla[0][1], new Skakac(Boja.CRNA, tabla[0][1]));
tabla[0][1].setIcon(new ImageIcon("images/cSkakac.gif"));
figureNaTabli.put(tabla[0][6], new Skakac(Boja.CRNA, tabla[0][6]));
tabla[0][6].setIcon(new ImageIcon("images/cSkakac.gif"));

// Beli lovci: polja C1 i F1
figureNaTabli.put(tabla[7][2], new Lovac(Boja.BELA, tabla[7][2]));
tabla[7][2].setIcon(new ImageIcon("images/bLovac.gif"));
figureNaTabli.put(tabla[7][5], new Lovac(Boja.BELA, tabla[7][5]));
tabla[7][5].setIcon(new ImageIcon("images/bLovac.gif"));

// Crni lovci: polja C8 i F8
figureNaTabli.put(tabla[0][2], new Lovac(Boja.CRNA, tabla[0][2]));
tabla[0][2].setIcon(new ImageIcon("images/cLovac.gif"));
figureNaTabli.put(tabla[0][5], new Lovac(Boja.CRNA, tabla[0][5]));
tabla[0][5].setIcon(new ImageIcon("images/cLovac.gif"));

// Bela kraljica: belo polje D1
figureNaTabli.put(tabla[7][3], new Dama(Boja.BELA, tabla[7][3]));
tabla[7][3].setIcon(new ImageIcon("images/bDama.gif"));

// Crna kraljica: crno polje D8
figureNaTabli.put(tabla[0][3], new Dama(Boja.CRNA, tabla[0][3]));
tabla[0][3].setIcon(new ImageIcon("images/cDama.gif"));

// Beli kralj: polje E1
figureNaTabli.put(tabla[7][4], new Kralj(Boja.BELA, tabla[7][4]));
tabla[7][4].setIcon(new ImageIcon("images/bKralj.gif"));

// Crni kralj: polje E8
figureNaTabli.put(tabla[0][4], new Kralj(Boja.CRNA, tabla[0][4]));
tabla[0][4].setIcon(new ImageIcon("images/cKralj.gif"));

igracPanel = new JPanel();
igracPanel.setLayout(new GridLayout(2, 1));

Icon beliIcon = new ImageIcon("images/wMove.gif");
bnp = new JLabel(beliIcon);
```

```
   Icon crniIcon = new ImageIcon("images/bMove.gif");
   cnp = new JLabel(crniIcon);

   bnp.setVerticalAlignment(JLabel.BOTTOM);
   bnp.setHorizontalAlignment(JLabel.CENTER);
   bnp.setPreferredSize(new Dimension(104, 168));

   cnp.setVerticalAlignment(JLabel.TOP);
   cnp.setHorizontalAlignment(JLabel.CENTER);
   cnp.setPreferredSize(new Dimension(104, 168));
   cnp.setVisible(false);

   igracPanel.add(cnp);
   igracPanel.add(bnp);

   setLayout(new FlowLayout());
   add(tablaPanel);
   add(igracPanel);
}

private void nizSlova(JPanel p)
{
   p.add(new JPanel());
   for (int i = 0; i < 8; i++)
     p.add(new BLabel((char) ('A' + i) + ""));
   p.add(new JPanel());
}

static void promenaIgracaNaPotezu()
{
   if (igracNaPotezu == Boja.BELA)
   {
     bnp.setVisible(true);
     cnp.setVisible(false);
   } else
   {
     cnp.setVisible(true);
     bnp.setVisible(false);
   }
}

class BLabel extends JLabel
{
   BLabel(String s)
   {
     super(s);
     setHorizontalAlignment(CENTER);
     setVerticalAlignment(CENTER);
   }
}

public static void potez(int cifra, char slovo)
{
   if (prvo)
   {
     Figura f = SahovskaTabla.getFigureNaTabli().get(
         SahovskaTabla.getPolje(8 - cifra, slovo - 'A'));
     if (f == null)
     {
       app.showStatus("Na polju nema figure");
       return;
     }

     if (f.getBoja() != igracNaPotezu)
     {
       app.showStatus("Na potezu je " + igracNaPotezu + " figura!");
```

```
            return;
          }

          app.showStatus("Pomeranje figure sa polja " + slovo + cifra
              + ". Izaberite odredisno polje");
          saSlovo = slovo;
          saCifra = cifra;
          prvo = false;
          return;
        }

        Figura f = figureNaTabli.get(tabla[8 - saCifra][saSlovo - 'A']);
        ArrayList<Polje> dopustivaPolja = f.odrediDopustivaPolja();
        if (dopustivaPolja.isEmpty())
        {
          app.showStatus("Nekorektan potez (polazna figura sa polja "
              + f.getPolje() + " ne moze se pomeriti)");
          prvo = true;
          return;
        }

        if (dopustivaPolja.contains(tabla[8 - cifra][slovo - 'A']))
        {
          nijeKraj = f.pomeriFiguru(tabla[8 - cifra][slovo - 'A']);
if (nijeKraj == false)
          {
            app.showStatus("Kraj! Pobedio je igrac cija je boja "
+ igracNaPotezu);
            return;
          }
          // premestamo slicicu figure na odgovarajuce polje
          tabla[8 - saCifra][saSlovo - 'A'].setIcon(null);
          String imeKlase = f.getClass().getSimpleName();
          String slika = "images/" + ((f.getBoja() == Boja.BELA) ? "b" : "c")
              + imeKlase + ".gif";
          tabla[8 - cifra][slovo - 'A'].setIcon(new ImageIcon(slika));
          prvo = true;
        } else
        {
          app.showStatus("Nedopustiv potez");
          prvo = true;
          return;
        }

        igracNaPotezu = (igracNaPotezu == Boja.BELA) ? Boja.CRNA : Boja.BELA;
        app.showStatus("Izaberite figuru");
        promenaIgracaNaPotezu();

    }

    public static HashMap<Polje, Figura> getFigureNaTabli()
    {
      return figureNaTabli;
    }

    public static Polje getPolje(int i, int j)
    {
      return tabla[i][j];
    }

    public static boolean getNijeKraj()
    {
      return nijeKraj;
    }

}
```

SahGUI.java

```
package sah;

import javax.swing.*;

public class SahGUI extends JApplet
{

   private SahovskaTabla sahovskaTabla = new SahovskaTabla(this);

   public void init()
   {
      SwingUtilities.invokeLater(new Runnable()
      {
         public void run()
         {
            createGUI();
         }
      });
   }

   public void createGUI()
   {
      setSize(550, 500);
      add(sahovskaTabla);
   }

}
```

Задаци са колоквијума и испита:

1. *Први колоквијум 2008*, *Геометријски објекти*.

Написати:

1. класу *Tacka* којом се представљају тачке у дводимензионој равни. Класа садржи следеће методе:
(а) подразумевани конструктор (тачка се поставља у координатни почетак)
(б) конструктор са датим координатама
(в) приступне методе
(г) метод за одређивање растојања између две тачке
(д) метод за померање тачке дуж *x* и *y* осе
(ђ) метод за испис тачке
(е) омогућити креирање новог објекта класе на основу већ постојећег, али независног од њега

2. апстрактну класу *GeometrijskiObjekat* за рад са геометријским објектима у 2D који се карактеришу својом централном тачком. Сваки геометријски објекат има и свој тип. Све чланице класе декларисати
као private. Обезбедити следеће методе:
(а) конструктор са датом вредношћу за тип (центар је у координатном почетку)
(б) конструктор на основу задате вредности за тип и задате централне тачке
(в) конструктор на основу задате вредности за тип и задатих координата централне тачке
(г) приступни и мутатор метод за центар
(д) метод за рачунање површине
(ђ) метод за испис геометријског објекта

3. класу *Kvadrat* која наслеђује класу *GeometrijskiObjekat* којом се описује квадрат са произвољним положајем у равни. Квадрат се још карактерише и једним (било којим) теменом. Обезбедити још и следеће методе:

(а) конструктор са датим теменом
(б) конструктор са датим теменом и центром
(в) метод за одређивање странице квадрата
(г) метод за испис квадрата у формату: Kvadrat: (x1,y1), (x2,y2)
(д) омогућити креирање новог објекта класе на основу већ постојећег, али независног од њега

4. класу *Krug* која наслеђује класу *GeometrijskiObjekat* којом се описује круг у равни. Круг се још карактерише и својим полупречником. Обезбедити још и следеће методе:
(а) конструктор са датим полупречником
(б) конструктор са датим полупречником и центром
(в) метод за испис круга у формату: Krug: (x1,x2), r
(г) омогућити креирање новог објекта класе на основу већ постојећег, али независног од њега

5. класу *PiramidaKupa* која симулира геометријско тело које се може понашати као пирамида или купа, у зависности од основе која може бити квадрат или круг. Објекти ове класе се карактеришу својом основом и висином. Обезбедити следеће методе:
(а) конструктор на основу дате основе и висине
(б) метод за одређивање запремине оваквог геометријског тела
(в) метод за испис на стандардни излаз у формату: Piramida/Kupa: osnova, visina

6. класу *TestGeometrija* за тестирање рада програма. Омогућити учитавање дводимензионих геометријских објеката (квадрата и кругова) са стандардног улаза.
(а) Квадрати се учитавају тако што се унесе ниска *kvadrat*, а затим се уносе вредности којима се описује конкретан објекат овог типа.
(б) Кругови се учитавају тако што се унесе ниска *krug*, а затим се уносе вредности којима се описује конкретан објекат овог типа.
Најпре се учитава број који представља колико ће дводимензионих објеката бити унешено са улаза. Објекти се учитавају у горе описаном формату. Учитане објекте смештати у одговарајући низ. Након тога за сваки учитани објекат креирати пирамиду/купу чија је основа тај објекат, при чему се висина пирамиде/купе уноси са стандардног улаза. Након тога исписати прво податке за креирану пирамиду/купу, а потом и одговарајућу запремину.

Објашњење:

Класа *GeometrijskiObjekat* садржи две инстанцне променљиве, тип геометријског објекта и централну тачку.
Методи:

- Конструктор за задатим типом. Централна тачка се поставља у координатни почетак наредбом
  `centar = new Tacka();`
- Конструктор на основу задатог типа и задате централне тачке. Наредбом
  `centar = new Tacka(c);` креира се централна тачка као идентична копија дате тачке *c*.
- Конструктор на основу задатог типа и задатих координата централне тачке. Наредбом
  `centar = new Tacka(x, y);` креира се централна тачка на основу датих координата.
- *get()* и *set()* методи за централну тачку
- метод *toString()* за генерисање *String*-репрезентације геометријског објекта. Формат се одређује на основу траженог формата исписа објеката класа *Kvadrat* и *Krug* у поставци задатка. Пошто и обе класе имају чланице *tip* и *centar*, *String*-репрезентација је облика *tip + ": " + centar*.
- Апстрактан метод *povrsina()* за рачунање површине геометријског објекта. Метод је апстрактан, јер дефиниција метода зависи од типа геометријског објекта.

Класа *Kvadrat* наслеђује класу *GeometrijskiObjekat*. Квадрат додатно има још инстанцну променљиву *teme*, типа *Tacka*, која представља једно од темена квадрата.
Методи:

- Конструктор са задатим теменом. Наслеђене чланице *tip* и *centar* иницијализују се позивом конструктора натклase наредбом `super("Kvadrat");` где се централна тачка поставља у координатни почетак. Позив конструктора натклase мора бити прва наредба у телу конструктора потклase.
- Конструктор са датом централном тачком и датим теменом. Наредбом `super("Kvadrat", centar);` иницијализују се наслеђене чланице *tip* и *centar*, док се наредбом `this.teme = new Tacka(teme);` теме квадрата иницијализује копијом датог темена.
- Конструктор копије који креира нови објекат на основу постојећег објекта *k*. Позивом конструктора натклase наредбом `super("Kvadrat", k.vratiCentar());` централна тачка се иницијализује копијом централне тачке квадрата *k*. Наредбом `this.teme = new Tacka(k.teme);` иницијализује се теме квадрата копијом темена квадрата *k*. Алтернатива су следеће три наредбе:
```
super("Kvadrat");
postaviCentar(new Tacka(k.vratiCentar()));
this.teme = new Tacka(k.teme);
```
- Метод *stranica()* рачуна страницу квадрата по формули $\frac{d}{2}\sqrt{2}$, где је *d* дужина дијагонале квадрата.
- Метод *povrsina()* који дефинише апстрактни метод из класе *GeometrijskiObjekat* по формули за рачунање површине квадрата.
- Метод *vratiTeme()* враћа вредност инстанцне промењиве *teme*
- Метод *toString()* позива метод *toString()* натклase наредбом `super.toString()` како би се на *String*-репрезентацију наслеђених чланица додала репрезентација чланице *teme*.

Класа *Krug* наслеђује класу *GeometrijskiObjekat*. Круг додатно има још инстанцну променљиву *poluprecnik*, типа *double*.
Методи:

- Конструктор са задатим полупречником. Наслеђене чланице *tip* и *centar* иницијализују се позивом конструктора натклase наредбом `super("Krug");` где се централна тачка поставља у координатни почетак.
- Конструктор са датом централном тачком и датим полупречником. Наредбом `super("Krug", centar);` иницијализују се наслеђене чланице *tip* и *centar*, док се наредбом `poluprecnik = r;` полупречник круга иницијализује на дату вредност.
- Конструктор копије који креира нови објекат на основу постојећег објекта *k*. Позивом конструктора натклase наредбом `super("Krug", k.vratiCentar());` централна тачка се иницијализује копијом централне тачке круга *k*. Наредбом `this.poluprecnik = k.poluprecnik;` иницијализује се полупречник круга на вредност полупречника круга *k*. Алтернатива су следеће три наредбе:
```
super("Krug");
postaviCentar(new Tacka(k.vratiCentar()));
this.poluprecnik = k.poluprecnik;
```
- Метод *povrsina()* који дефинише апстрактни метод из класе *GeometrijskiObjekat* по формули за рачунање површине круга.
- Метод *toString()* позива метод *toString()* натклase наредбом `super.toString()` како би се на *String*-репрезентацију наслеђених чланица додала репрезентација чланице *poluprecnik*.

Класа *PiramidaKupa* дефинише геометријско тело које може да се понаша и као пирамида и као купа у зависности од основе. Пошто основа може бити типа *Kvadrat* или *Krug*, треба ставити да је основа типа *GeometrijskiObjekat*, јер је то базна класа и за класу *Kvadrat* и за класу *Krug*.
Методи:

- Конструктор на основу дате основе и висине иницијализује инстанцне промењиве наредбама:
```
this.osnova = osnova;
this.visina = visina;
```
Пошто је класа *GeometrijskiObjekat* апстрактна класа, није дозвољено користити оператор *new* у првој наредби. Алтернатива за прву наредбу је следећи фрагмент:
```
if(this.osnova instanceof Kvadrat)
```

```
        this.osnova = new Kvadrat((Kvadrat)osnova);
      else
        this.osnova = new Krug((Krug)osnova);
```

- Метод *zapremina()* који рачуна запремину по заједничкој формули за рачунање запремине пирамиде и купе.
- метод *toString()* дефинише *String*-репрезентацију према типу основе. Тип се проверава оператором *instanceof*. Ако је основа квадрат, у питању је пирамида, иначе, купа.

Решење:

Tacka.java

```
package treca_grupa;

public class Tacka
{
   private double x;
   private double y;

   public Tacka()
   {
      x = y = 0.0;
   }

   public Tacka(double x, double y)
   {
      this.x = x;
      this.y = y;
   }

   public Tacka(final Tacka t)
   {
      this();
      postaviX(t.vratiX());
      postaviY(t.vratiY());
   }

   public double vratiX()
   {
      return x;
   }

   public double vratiY()
   {
      return y;
   }

   public void postaviX(double x)
   {
      this.x = x;
   }

   public void postaviY(double y)
   {
      this.y = y;
   }

   public double rastojanje(Tacka t)
   {
      return Math.sqrt(Math.pow(t.x - x, 2) + Math.pow(t.y - y, 2));
   }

   public void pomeri(double dx, double dy)
   {
      x += dx;
```

```
      y += dy;
   }

   public String toString()
   {
      return "(" + x + "," + y + ")";
   }

}
```

Kvadrat.java

```
package treca_grupa;

public class Kvadrat extends GeometrijskiObjekat
{
   // Kvadrat se karakterise jos jednim (bilo kojim) temenom
   private Tacka teme;

   // Konstruktor na osnovu zadatog temena
   public Kvadrat(Tacka teme)
   {
      super("Kvadrat");
      this.teme = new Tacka(teme);
   }

   // Konstruktor na osnovu zadatog temena i centralen tacke
   public Kvadrat(Tacka teme, final Tacka centar)
   {
      super("Kvadrat", centar);
      this.teme = teme;
   }

   // Konstruktor kopije
   public Kvadrat(final Kvadrat k)
   {
      super("Kvadrat", k.vratiCentar());
      this.teme = new Tacka(k.teme);
   }

   public Tacka vratiTeme()
   {
      return teme;
   }

   // Racuna stranicu kvadrata
   public double stranica()
   {
      return teme.rastojanje(vratiCentar()) * Math.sqrt(2);
   }

   // Racuna povrsinu kvadrata
   public double povrsina()
   {
      return Math.pow(stranica(), 2);
   }

   // String reprezentacija kvadrata
   public String toString()
   {
      return super.toString() + ", " + teme;
   }
}
```

Krug.java

```
package treca_grupa;

public class Krug extends GeometrijskiObjekat
{
   // Krug se karakterise i poluprecnikom
   private double poluprecnik;

   // Konstruktor na osnovu zadatog poluprecnika
   public Krug(double r)
   {
      super("Krug");
      poluprecnik = r;
   }

   // Konstruktor na osnovu zadatog poluprecnika i centra
   public Krug(double r, final Tacka centar)
   {
      super("Krug", centar);
      poluprecnik = r;
   }

   // Konstruktor kopije
   public Krug(final Krug k)
   {
      super("Krug", k.vratiCentar());
      this.poluprecnik = k.poluprecnik;
   }

   // Povrsina kruga
   public double povrsina()
   {
      return Math.pow(poluprecnik, 2);
   }

   // String reprezentacija kruga
   public String toString()
   {
      return super.toString() + ", " + poluprecnik;
   }
}
```

PiramidaKupa.java

```
package treca_grupa;

public class PiramidaKupa
{
   private GeometrijskiObjekat osnova;
   private double visina;

   // Konstruktor na osnovu date osnove i date visine
   public PiramidaKupa(GeometrijskiObjekat osnova, double visina)
   {
      this.osnova = osnova;
      this.visina = visina;
   }

   // Zapremina piramide/kupa
   public double zapremina()
   {
      return (osnova.povrsina() * visina) / 3;
   }

   // String reprezentacija piramide/kupe
   public String toString()
   {
```

```
      if (osnova instanceof Kvadrat)
         return "Piramida:" + "\nOsnova:    " + osnova +
                  "\nVisina:    " + visina;
      else if (osnova instanceof Krug)
         return "Kupa:" + "\nOsnova:    " + osnova +
                 "\nVisina:   " + visina;
      else
         return "";
   }

}
```

TestGeometrija.java

```
package treca_grupa;

import java.util.Scanner;

public class TestGeometrija
{
   public static void main(String[] args)
   {
      Scanner skener = new Scanner(System.in);

      System.out.println("Unesi broj objekata");
      int n = skener.nextInt();

      GeometrijskiObjekat[] objekti = new GeometrijskiObjekat[n];

      String tip;

      for(int i=0; i<objekti.length; i++)
      {
         System.out.println("Unesi geometrijski objekat (kvadrat/krug):");
         tip = skener.next();
         if(tip.equalsIgnoreCase("kvadrat"))
         {
              System.out.println("Unesi centar kvadrata:");
              System.out.print("x = ");
              double x = skener.nextDouble();
              System.out.print("y = ");
              double y = skener.nextDouble();
              Tacka centar = new Tacka(x, y);

              System.out.println("Unesi teme kvadrata:");
              System.out.print("x = ");
              x = skener.nextDouble();
              System.out.print("y = ");
              y = skener.nextDouble();
              Tacka teme = new Tacka(x, y);

              objekti[i] = new Kvadrat(teme, centar);
         }
         else if(tip.equalsIgnoreCase("krug"))
         {
              System.out.println("Unesi centar kruga:");
              System.out.print("x = ");
              double x = skener.nextDouble();
              System.out.print("y = ");
              double y = skener.nextDouble();
              Tacka centar = new Tacka(x, y);

              System.out.println("Unesi poluprecnik kruga:");
              System.out.print("r = ");
              double r = skener.nextDouble();

              objekti[i] = new Krug(r, centar);
```

```
         }
         else
         {
            System.out.println("Nepoznati objekat");
            i--;
         }
      }

      PiramidaKupa piramidaKupa;

      System.out.println("\nKreiranje piramida i kupa:\n");
      for(int i=0; i<objekti.length; i++)
      {
         System.out.println(objekti[i]);
         System.out.print("Unesi visinu: ");
         double visina = skener.nextDouble();
         piramidaKupa = new PiramidaKupa(objekti[i], visina);

         System.out.println("\n" + piramidaKupa);
         System.out.println("Zapremina: "+ piramidaKupa.zapremina() +"\n");
      }
   }
}
```

2. *Први колоквијум 2008, Функције*

Написати:

1. апстрактну класу *Funkcija* за рад са функцијама једне променљиве. Класа садржи следеће методе:
(а) метод за одређивање вредности функције у датој тачки (резултат је реалан број)
(б) метод за одређивање извода функције (резултат је функција)
(в) метод за одређивање вредности извода у датој тачки (резултат је реалан број)
(г) метод за испис функције

2. класу *Trig* којом се описују тригонометријске функције облика *a * sinx + b * cosx*. Свака функција из ове класе карактерише се конкретним вредностима за коефицијенте *a* и *b*. Обезбедити неопходне констукторе и дефинисати потребне методе. Формат исписа тригонометријских функција је:

Trigonometrijska: a*sinx + b*cosx

3. класу *Exp* којом се описују експоненцијалне функције облика $a * e^{b*x}$. Свака функција из ове класе карактерише се конкретним вредностима за коефицијенте *a* и *b*. Обезбедити неопходне констукторе и дефинисати потребне методе. Формат исписа експоненцијалних функција је:

Eksponencijalna: a*e ^ (b*x)

4. класу *TestFunkcija* за тестирање рада са функцијама. Омогућити учитавање функција са стандардног улаза.

(а) Тригонометријске функције се учитавају тако што се унесе слово *t*, а затим се уносе вредности којима се описује конкретан објекат овог типа.

(б) Експоненцијалне функције се учитавају тако што се унесе слово *e*, а затим се уносе вредности којима се описује конкретан објекат овог типа.

Функције се учитавају у горе описаном формату, све док се не унесе слово *x* као сигнал краја уноса. Сматрати да је могуће унети податке о највише 30 функција. Функције смештати у одговарајући низ. Исписати број креираних функција. Са стандардног улаза учитати број који представља тачку у којој се рачунају вредности функција. Након тога сваку функцију из низа исписати на излаз, затим исписати њену вредност у учитаној тачки, исписати њен извод, као и вредност извода у учитаној тачки.

Објашњење:

Класа *Funkcija* је апстрактна класа која нема инстанцних променљивих. Садржи следеће методе:

a) Апстрактни метод *vrednost()* рачуна вредност функције у датој тачки. Метод је апстрактан, јер његова дефиниција зависи од конкретног типа функције.
b) Апстрактан метод *izvod()* рачуна извод функције који је такође функција.
c) Метод *vrednostIzvoda()* рачуна вредност извода функције у датој тачки. Овај метод је могуће дефинисати на основу претходна два метода: методом *izvod()* одреди се функција која представља извод полазне функције, а затим се над добијеном функцијом примени метод *vrednost()* којим се срачуна вредност у датој тачки.
d) Апстрактан метод *toString()* за *String*-репрезентацију функције

Класа *Trig* наслеђује класу *Funkcija* и има две инстанцне променљиве, коефицијенте *a* и *b* тригонометријске функције, типа *double*.
Садржи следеће методе:

a) Конструктор на основу задатих вредности коефицијената *a* и *b*
b) Метод *izvod()* дефинише истоимени апстрактан метод из базне класе *Funkcija* и рачуна извод као нову тригонометријску функцију са коефицијентима *–b, a*.
c) Метод *vrednost()* дефинише истоимени апстрактан метод из базне класе и рачуна вредност тригонометријске функције у датој тачки.
d) Метод *toString()* дефинише истоимени апстрактан метод из базне класе и генерише *String*-репрезентацију тригонометријске функције у траженом облику.

Класа *Exp* наслеђује класу *Funkcija* и има врло сличне особине као и класа *Trig*, тако да неће бити посебно навођене.

У класи *TestFunkcija* прво се дефинише низ где се смештају учитане функције. Низ има највише 30 елемената. Потом се у *while* петљи учитава тип функције: слово *'t'* за тригонометријске, односно, слово *'e'* за експоненцијалне функције, док је слово *'x'* сигнал краја.
Тип се учитава као *String*, при чему се методом *charAt(0)* издваја карактер којим је тип представљен, како би се извршила провера употребом *switch*-а. У зависности од типа, учитавају се вредности коефицијената одговарајуће функције, креира се објекат тог типа и смешта у низ. Функције се учитавају док се не унесе слово које је сигнал краја или се не прекорачи димензија низа.
Након тога, исписује се број учитаних функција. Учитава се реални број и у петљи се за сваку функцију најпре испише њена *String*-репрезентација, а потом рачуна и исписује њена вредност у учитаној тачки, извод, као и вредност извода у учитаној тачки.

Решење:

Funkcija.java

```
package cetvra_grupa;

public abstract class Funkcija
{
   // Vrednost funkcije u datoj tacki.
   public abstract double vrednost(double x);

   // Izvod funkcije.
   public abstract Funkcija izvod();

   // Vrednost izvoda funkcije u datoj tacki.
   public double vrednostIzvoda(double x)
   {
      return izvod().vrednost(x);
   }
```

```
   // String reprezentacija funkcije.
   public abstract String toString();
}
```

Trig.java

```
package cetvra_grupa;

public class Trig extends Funkcija
{
   // Svaka funkcija ovog tipa se karakterise koeficijentima a i b
   private double a;
   private double b;

   // Konstruktor na osnovu datih koeficijenata
   public Trig(double a, double b)
   {
      this.a = a;
      this.b = b;
   }

   // Izvod funkcije a*sin(x)+b*cos(x) je (-b)*sin(x)+a*cos(x)
   public Funkcija izvod()
   {
      return new Trig(-b, a);
   }

   // Vrednost f-je u datoj tacki
   public double vrednost(double x)
   {
      return a*Math.sin(x) + b*Math.cos(x);
   }

   // String reprezentacija trigonometrijske funkcije
   public String toString()
   {
      return "Trigonometrijska: " + a + "*sinx + " + b + "*cosx";
   }
}
```

Exp.java

```
package cetvra_grupa;

public class Exp extends Funkcija
{
   // Svaka funkcija ovog tipa se karakterise koeficijentima a i b
   private double a;
   private double b;

   // Konstruktor na osnovu datih koeficijenata
   public Exp(double a, double b)
   {
      this.a = a;
      this.b = b;
   }

   // Izvod funkcije a*e^b je (a*b)*e^b
   public Funkcija izvod()
   {
      return new Exp(a*b, b);
   }

   // Vrednost funkcije u tacki
   public double vrednost(double x)
   {
```

```
      return a*Math.exp(b*x);
   }

   // String reprezentacija eksponencijalne funkcije
   public String toString()
   {
      return "Eksponencijalna: " + a + "*e^(" + b + "*x)";
   }
}
```

TestFunkcija.java

```
package cetvra_grupa;

import java.util.Scanner;

public class TestFunkcija
{
   public static void main(String[] args)
   {
      Scanner skener = new Scanner(System.in);

      Funkcija[] funkcije = new Funkcija[30];

      String tip;
      System.out.println("Unesite tip funkcije (t/e), " + "odnosno x za kraj: ");
      tip = skener.next();
      int i=0;
      while(tip.compareTo("x") != 0 && i<30)
      {

         switch(tip.charAt(0))
         {
            case 't':
               System.out.println("Unesi koeficijent a:");
               System.out.print("a = ");
               double a = skener.nextDouble();
               System.out.println("Unesi koeficijent b:");
               System.out.print("b = ");
               double b = skener.nextDouble();
               funkcije[i++] = new Trig(a, b);
               break;
            case 'e':
               System.out.println("Unesi koeficijent a:");
               System.out.print("a = ");
               a = skener.nextDouble();
               System.out.println("Unesi koeficijent b:");
               System.out.print("b = ");
               b = skener.nextDouble();
               funkcije[i++] = new Exp(a, b);
               break;
            default:
               System.out.println("Nepoznata funkcija");
               break;
         }
         System.out.println("Unesite tip funkcije (t/e), " +
                            "odnosno x za kraj:");
         tip = skener.next();
      }

      System.out.println("Broj kreiranih funkcija: " + i);

      double x;
      System.out.println("Unesi tacku u kojoj se racunaju " +
                         "vrednosti funkcija: ");
      x = skener.nextDouble();
```

```
      System.out.println("Kreirane funkcije:");
      for(int j=0; j<i; j++)
      {
         System.out.println("\n" + funkcije[j]);
         System.out.print("Vrednost funkcije u datoj tacki: ");
         System.out.println(funkcije[j].vrednost(x));

         Funkcija izvod = funkcije[j].izvod();
         System.out.println("Izvod:\n" + izvod);
         System.out.println("Vrednost izvoda:" + izvod.vrednost(x));
      }
   }
}
```

3. први колоквијум 2008, Особе. Написати класу *Osoba* која садржи инстанцне променљиве: име, датум рођења и адресу становања (све типа *String*).
Djak је особа за коју се додатно знају назив школе (типа *String*), разред који похађа (типа *int*) и просечна оцена (типа *double*).
Student је особа за коју се додатно зна име факултета и име смера (типа *String*), као и година уписа на факултет и година студија на коју је студент тренутно уписан (типа *int*).
Zaposleni је особа за коју се додатно зна име фирме и име одељења у коме ради (типа *String*) и месечна плата (типа *double*).
Обезбедити да се полиморфно извршава метод:

```
void markica();
```

који треба да испише боју маркице која одговара објектима класе у којој је метод имплементиран, и то: "плава маркица" за ђаке, "зелена маркица" за студенте и "црвена маркица" за запослене.
У свакој од класа *Djak*, *Student* і *Zaposleni* имплементирати метод:

```
double prosek();
```

који у класи *Djak* рачуна просечну оцену на основу свих креираних објеката класе *Djak*, у класи *Student* рачуна просечно време студирања на основу свих креираних објеката класе *Student*, а у класи *Zaposleni* рачуна просечну месечну плату на основу свих креираних објеката класе *Zaposleni*.
Поред ових метода, у класама имплементирати и све остале методе неопходне за коректно извршавање главног програма, као и средство за креирање новог објекта класе идентичног датом постојећем објекту, али независног од њега.
Написати тест-класу која ради следеће:

- захтева да корисник са стандардног улаза унесе број објеката, а затим
- у петљи очекује да корисник унесе "ђак", "студент" или "запослени"
- након тога податке помоћу којих се креира објекат изабраног типа
- тако учитане објекте потребно је смештати у одговарајућу структуру података
- главни програм, затим, за сваки креирани објекат исписује његову *String*-репрезентацију, боју маркице карактеристичну за објекте тог типа, као и просечну вредност срачунату помоћу одговарајућег метода *prosek()*.

Објашњење:

У базној класи, *Osoba*, метод *markica()* има празно тело, а биће имплементиран на одговарајући начин у изведеним класама.

Изведена класа *Djak* има две статичке променљиве, *zbirOcena* и *brojac*, које служе за рачунање просечне оцене свих ђака. Обе променљиве иницијализоване су нулама, а приликом креирања сваког новог објекта ове класе, у конструктору, врши се инкрементирање бројача и увећање променљиве *zbirOcena* за вредност просечне оцене ђака који је управо креиран. Просечна оцена свих креираних ђака рачуна се у статичком методу *prosek()* дељењем вредности *zbirOcena* вредношћу *brojac*. Обе променљиве, као и метод *prosek()*, су статички чланови класе јер се односе на класу као целину, а не на њене појединачне објекте. Метод *markica()* само исписује боју маркице карактеристичну за објекте ове класе.

Класе *Student* и *Zaposleni* не разликују се пуно од класе *Djak*. У класи *Student* дефинисана је константа која представља текућу годину, а године студирања једног студента рачунају се као разлика текуће и године уписа на студије.

У тест-класи најпре се уноси жељени број објеката и дефинише низ чији су објекати типа базне класе, јер је у њега могуће смештати референце на објекте произвољних изведених класа. Затим се, у *for*-петљи, изабира тип објекта и уносе подаци неопходни за његово креирање. Креирани објекат се смешта у низ. По завршетку петље прелази се на исписивање тражених података. Полиморфни позив метода *markica()* врши се помоћу промењиве која представља одговарајући елемент низа. За позив одговарајућег метода *prosek()* неопходно је утврдити стварни тип објекта што се чини оператором *instanceof*. Метод *prosek()* је статички, те је приликом позива његово име неопходно квалификовати именом класе.

Решење:

Osoba.java

```
package osobe;

public class Osoba
{
   private String ime;
   private String datum;
   private String adresa;

   public Osoba()
   {
   }

   public Osoba(String ime, String datum, String adresa)
   {
      this.ime = ime;
      this.datum = datum;
      this.adresa = adresa;
   }

   public Osoba(final Osoba o)
   {
      this(o.ime, o.datum, o.adresa);
   }

   public void markica()
   {
   }

   public String getIme()
   {
      return ime;
   }

   public String getAdresa()
   {
      return adresa;
   }

   public String getDatum()
   {
      return datum;
   }

   public void setIme(String ime)
   {
      this.ime = ime;
```

```
   }

   public void setAdresa(String adresa)
   {
      this.adresa = adresa;
   }

   public void setDatum(String datum)
   {
      this.datum = datum;
   }

   public String toString()
   {
      return "Osoba " + ime + ", rodjena je: " + datum
                      + ", stanuje u " + adresa;
   }

}
```

Djak.java

```
package osobe;

public class Djak extends Osoba
{
   private static double zbirOcena = 0.0;
   private static int brojac = 0;

   private String skola;
   private int razred;
   private double prosecnaOcena;

   public Djak(String ime, String datum, String adresa,
               String skola, int razred, double prosecnaOcena)
   {
      super(ime, datum, adresa);
      this.skola = skola;
      this.razred = razred;
      this.prosecnaOcena = prosecnaOcena;
      zbirOcena += prosecnaOcena;
      ++brojac;
   }

   public Djak(final Djak djak)
   {
      this(djak.getIme(), djak.getDatum(), djak.getAdresa(),
           djak.skola, djak.razred, djak.prosecnaOcena);

   }

   public void setSkola(String skola)
   {
      this.skola = skola;
   }

   public void setRazred(int razred)
   {
      this.razred = razred;
   }

   public void setProsecnaOcena(double prosecnaOcena)
   {
      this.prosecnaOcena = prosecnaOcena;
   }

   public String getSkola()
```

```
   {
      return skola;
   }

   public int getRazred()
   {
      return razred;
   }

   public double getProsecnaOcena()
   {
      return prosecnaOcena;
   }

   public void markica()
   {
      System.out.println("plava markica");
   }

   static public double prosek()
   {
      return zbirOcena / brojac;

   }

   public String toString()
   {
      return super.toString() + ", djak je skole " + skola + ", ide u "
          + razred + ". razred i ima prosecnu ocenu " + prosecnaOcena;
   }

}
```

Student.java

```
package osobe;

public class Student extends Osoba
{
   private static final int TEKUCA_GODINA = 2008;

   private static double zbirGodinaStudiranja = 0.0;
   private static int brojac = 0;

   private String fakultet;
   private String smer;
   private int godinaUpisa;
   private int godinaStudija;

   public Student(String ime, String datum, String adresa,
                  String fakultet, String smer,
                  int godinaUpisa, int godinaStudija)
   {
      super(ime, datum, adresa);
      this.fakultet = fakultet;
      this.smer = smer;
      this.godinaUpisa = godinaUpisa;
      this.godinaStudija = godinaStudija;

      zbirGodinaStudiranja += TEKUCA_GODINA - godinaUpisa;
      ++brojac;
   }

   public Student(final Student s)
   {
      this(s.getIme(), s.getDatum(), s.getAdresa(),
           s.fakultet, s.smer, s.godinaUpisa, s.godinaStudija);
```

```
    }

    public void setFakultet(String fakultet)
    {
       this.fakultet = fakultet;
    }

    public void setSmer(String smer)
    {
       this.smer = smer;
    }

    public void setGodinaUpisa(int godinaUpisa)
    {
       this.godinaUpisa = godinaUpisa;
    }

    public void setGodinaStudija(int godinaStudija)
    {
       this.godinaStudija = godinaStudija;
    }

    public String getFakultet()
    {
       return fakultet;
    }

    public String getSmer()
    {
       return smer;
    }

    public int getGodinaUpisa()
    {
       return godinaUpisa;
    }

    public int getGodinaStudija()
    {
       return godinaStudija;
    }

    public void markica()
    {
       System.out.println("zelena markica");
    }

    static public double prosek()
    {
       return zbirGodinaStudiranja / brojac;

    }

    public String toString()
    {
       return super.toString() + ", student je fakulteta " + fakultet
           + " na smeru " + smer + " upisala je fakultet " + godinaUpisa
           + ". godine i trenutno je na " + godinaStudija + ". godini studija";
    }

}
```

Zaposleni.java

```
package osobe;

public class Zaposleni extends Osoba
```

```
{
   private static double zbirPlata = 0.0;
   private static int brojac = 0;

   private String firma;
   private String odeljenje;
   private double plata;

   public Zaposleni(String ime, String datum, String adresa,
               String firma, String odeljenje, double plata)
   {
      super(ime, datum, adresa);
      this.firma = firma;
      this.odeljenje = odeljenje;
      this.plata = plata;
      zbirPlata += plata;
      ++brojac;
   }

   public Zaposleni(final Zaposleni z)
   {
      this(z.getIme(), z.getDatum(), z.getAdresa(),
            z.firma, z.odeljenje, z.plata);
   }

   public void markica()
   {
      System.out.println("crvena markica");
   }

   static public double prosek()
   {
      return zbirPlata / brojac;
   }

   public String toString()
   {
      return super.toString() + ", zaposlena je u firmi " + firma
          + " na odeljenju " + odeljenje + " i ima platu " + plata + " din.";
   }

}
```

Test.java

```
package osobe;

import java.util.Scanner;

public class Test
{
   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);
      System.out.print("Unesite broj objekata: ");
      int n = sc.nextInt();
      if (n <= 0)
         System.exit(0);

      Osoba[] osobe = new Osoba[n];
      for (int i = 0; i < n; i++)
      {
         System.out.println((i + 1)
             + ". Unesite \"djak\" ,\"student\" ili \"zaposleni\": ");
         String izbor = sc.next();
         if (izbor.equalsIgnoreCase("djak"))
         {
```

```
            System.out.print("ime djaka: ");
            String ime = sc.next();
            System.out.print("datum rodjenja: ");
            String datum = sc.next();
            System.out.print("adresa stanovanja: ");
            String adresa = sc.next();
            System.out.print("skola: ");
            String skola = sc.next();
            System.out.print("razred: ");
            int razred = sc.nextInt();
            System.out.print("prosecna ocena: ");
            double prosecnaOcena = sc.nextDouble();

            osobe[i] = new Djak(ime, datum, adresa, skola, razred, prosecnaOcena);

         } else if (izbor.equalsIgnoreCase("student"))
         {
            System.out.print("ime studenta: ");
            String ime = sc.next();
            System.out.print("datum rodjenja: ");
            String datum = sc.next();
            System.out.print("adresa stanovanja: ");
            String adresa = sc.next();
            System.out.print("fakultet: ");
            String fakultet = sc.next();
            System.out.print("smer: ");
            String smer = sc.next();
            System.out.print("godina upisa: ");
            int godinaUpisa = sc.nextInt();
            System.out.print("godina studija: ");
            int godinaStudija = sc.nextInt();

            osobe[i] = new Student(ime, datum, adresa,
fakultet, smer, godinaUpisa, godinaStudija);

         } else if (izbor.equalsIgnoreCase("zaposleni"))
         {
            System.out.print("ime zaposlenog: ");
            String ime = sc.next();
            System.out.print("datum rodjenja: ");
            String datum = sc.next();
            System.out.print("adresa stanovanja: ");
            String adresa = sc.next();
            System.out.print("ime firme: ");
            String firma = sc.next();
            System.out.print("ime odeljenja: ");
            String odeljenje = sc.next();
            System.out.print("plata: ");
            double plata = sc.nextDouble();

            osobe[i] = new Zaposleni(ime, datum, adresa, firma, odeljenje, plata);

         } else
         {
            System.out.println("Greska. Pokusajte ponovo.");
            --i;
         }
      }

      for (Osoba osoba : osobe)
      {
         System.out.print(osoba + " ");
         osoba.markica();
         if (osoba instanceof Djak)
            System.out.println(" prosek svih djaka: " + Djak.prosek());
         else if (osoba instanceof Student)
            System.out.println(" prosek studiranja svih studenata: "
```

```
            + Student.prosek());
        else if (osoba instanceof Zaposleni)
           System.out.println(" prosek plata zaposlenih: " + Zaposleni.prosek());
     }
   }
}
```

4. *Други колоквијум 2008, Домине*.

Написати аплет који садржи:

• Два поља за унос два цела броја са одговарајућим лабелама. Први број представља број тачака на једној половини домине, а други на другој половини домине. Минималан број тачака на једној половини је 0, а максималан 6. Домина може имати исти број тачака на обе половине.

• Лабелу *poruka* за испис потребних порука.

- Креирати све могуће домине које задовољавају дати услов и сместити их у колекцију која се може реализовати вектором. Домине произвољно промешати. Креирање домина и мешање извшити пре креирања компоненти које су део аплета.
- Горња поља служе за уношење података о домини која се бира из колекције. ОБАВЕЗНО проверити да ли је домина коректно унешена и у случају неисправног уноса исписати одговарајућу поруку у лабели *poruka*. Бира се тачно 5 домина. Свака домина која се бира из колекције расположивих домина истовремено се из ње **уклања**. ОБАВЕЗНО проверити да ли је домина која се бира већ раније изабрана и у том случају исписати одговарајућу поруку у лабели *poruka*. Када се изабере свих 5 домина, онемогућити даљи унос у поља, а лабела *poruka* треба да садржи обавештење да је изабран потребан број домина.

• Дугме *Izaberi*. Једним кликом на дугме бира се одговарајућа домина из колекције. Успешно бирање домине за последицу има и истовремено креирање и додавање дугмета аплету које као лабелу има приказ изабране домине. Домине се приказују у облику [broj1 | broj2]. (погледати метод *validate()* у класи *java.awt.Container*)

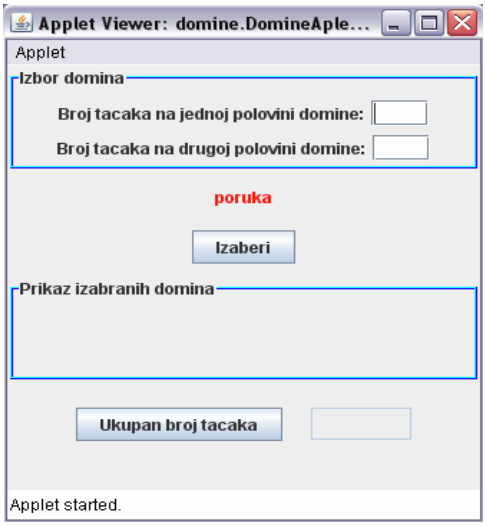
- Омогућити да, кад се миш позиционира на неко од изабраних дугмета, на дугмету се испише укупан број тачака на домини која је на њему приказана и то црвеном бојом. Када се миш помери са дугмета, дугме добија свој првобитан изглед.

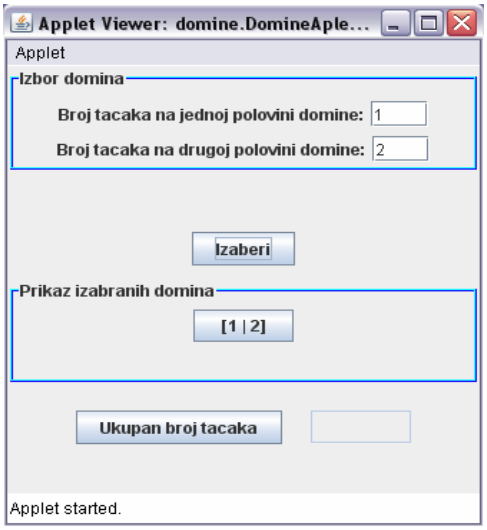
• Дугме *UkupanBrojTacaka* и поље за испис резултата. Кликом на дугме у пољу се аутоматски приказује укупан број тачака на свим доминама које су изабране. Онемогућити било какав унос у дато поље.

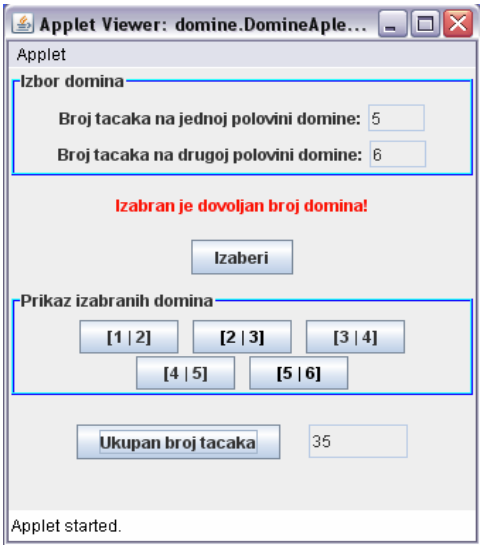
- Омогућити да, када се мишем уђе на површину поља, резултат се прикаже црвеном бојом, при чему је величина фонта увећана за 10 пиксела. Померањем миша са површине поља резултат добија првобитан изглед.

Поставити величину аплета тако да када се аплет покрене све његове компоненте буду видљиве и распоређене као на датим сликама.

Аплет иницијално треба да буде приказан као на првој слици. Друга слика показује избор домине и додавање дугмета са изабраном домином аплету. На последњој слици су приказане све изабране домине и тражени резултат.


Applet Viewer: domine.DomineAple...
Applet
Izbor domina
Broj tacaka na jednoj polovini domine:
Broj tacaka na drugoj polovini domine:
poruka
Izaberi
Prikaz izabranih domina
Ukupan broj tacaka
Applet started.


Applet Viewer: domine.DomineAple...
Applet
Izbor domina
Broj tacaka na jednoj polovini domine: 1
Broj tacaka na drugoj polovini domine: 2
Izaberi
Prikaz izabranih domina
[1 | 2]
Ukupan broj tacaka
Applet started.


Applet Viewer: domine.DomineAple...
Applet
Izbor domina
Broj tacaka na jednoj polovini domine: 5
Broj tacaka na drugoj polovini domine: 6
Izabran je dovoljan broj domina!
Izaberi
Prikaz izabranih domina
[1 | 2]
[2 | 3]
[3 | 4]
[4 | 5]
[5 | 6]
Ukupan broj tacaka
35
Applet started.

Објашњење:

Класа *Domina* има две инстанцне променљиве типа *int*, које представљају број тачака на једној, односно, на другој половини домине.
Конструктор може да избаци изузетак типа *IllegalArgumentException*, уколико задате вредности за инстанцне променљиве не припадају опсегу [0, 6].
Метод *zbir()* рачуна укупан број тачака на домини.
Метод *jednake()* проверава да ли су домина која се дефинише и домина која се прослеђује као аргумент методу једнаке. Домине су једнаке ако имају исти број тачака на обе половине.

Класа *KolekcijaDomina* дефинише колекцију домина при чему је коришћен објекат класе *java.util.Vector* за дефиницију колекције.
У конструктору се креира колекција свих могућих домина које задовољавају задате услове. Треба водити рачуна да је *Domina(i, j)* исто што и *Domina(j, i)*, па зато у другој *for* петљи у телу конструктора бројач *j* има иницијалну вредност *i*, а не *0*.
Метод *promesaj()* позива статички метод *Collections.shuffle()* да на случајан начин промеша објекте у колекцији.
Метод *ukloni()* уклања домину са датом комбинацијом бројева тачака из колекције. За пролаз кроз колекцију користи се итератор. Када се пронађе домина која задовољава услов, методом *remove()* се уклања из колекције и враћа се *true*, као сигнал успеха. Уколико у колекцији нема такве домине, враћа се *false*.

У аплету *DomineAplet*, пре креирања графичких компоненти, креира се колекција расположивих домина и произвољно се промеша. Потом се креирају компоненте аплета као на приказаној слици. Креирање компоненти неће бити посебно објашњавано.
Обрада догађаја обухвата следеће:

- Клик на дугме *izaberi*. На основу вредности у текстуалним пољима *unos_a* и *unos_b*, креира се објекат класе *Domina*, уколико су задате вредности коректне. У супротном се избацује изузетак типа *IllegalArgumentException* или типа *NumberFormatException*. Први, ако вредности нису у интервалу [0, 6], а други, ако вредности нису бројеви. Уколико домина није раније бирана, биће пронађена у колекцији *kolekcija_domina*, уклоњена из ње и додата колекцији изабраних домина, *domine_izbor*. За тако изабрану домину, креира се дугме са лабелом једнаком *String*-репрезентацији домине. Дугме се додаје у колекцију изабраних дугмета *dugmeta_izbor*.
  У оквиру обраде овог догађаја, обрађују се и догађаји везани за свако од изабраних дугмета. Када се мишем уђе на површину изабраног дугмета, лабела дугмета треба да прикаже збир бројева на левој и десној половини домине која је придружена том дугмету. Када се напусти површина дугмета, лабела дугмета добија свој првобитни изглед. Дефинисани су методи *mouseEntered()* и *mouseExited()* за ове обраде.
  У методу *mouseEntered()* идентификује се дугме над којим се догађај дешава позивом метода *e.getComponent()*, где је *e* објекат типа *MouseEvent*. Одређује се индекс дугмета у колекцији *dugmeta_izbor* методом *indexOf()*, како би се на основу тог индекса пронашла домина у колекцији *domine_izbor* која му одговара, тј. која има исти индекс. Домина је потребна како би се одредио збир бројева тачака на њеној левој и десној половини и приказао у лабели дугмета.
  У методу *mouseExited()* се на исти начин за дугме проналази одговарајућа домина и враћа се првобитни изглед лабеле дугмета.
  Након успешног избора домине, бројач се увећава за један и уколико је достигао максималну вредност (5), исписује се одговарајућа порука у лабели *poruka* и онемогућава се даље едитовање садржаја текстуалних поља *unos_a* и *unos_b*.
  Приказ изабраног дугмета у прозору аплета врши се позивом метода *validate()*. Овај позив је неопходан јер се дугмета динамички додају прозору аплета, тј. нису део садржаја аплета при стартовању.

- Клик на дугме *ukupno_tacaka*. Обрада се састоји у томе да се прође кроз све елементе колекције изабраних домина и срачуна се збир бројева тачака на свим доминама. Добијени резултат се приказује у лабели *polje_rezultat*.

- Прелазак мишем преко текстуалног поља *polje_rezultat*. Обрада се састоји у томе да се у методу *mouseEntered()* најпре сачува текући фонт и текућа боја за испис, како би се омогућило тражено увећање величине фонта. Неопходно је сачувати и текућу боју, јер је потребна у методу *mouseExited()*, како би се приказ резултата вратио на првобитни облик.

Решење:

Domina.java

```
package domine;

public class Domina
{
   private int a;
   private int b;

   public Domina(int a, int b) throws IllegalArgumentException
   {
      if(a >= 0 && a <= 6)
         this.a = a;
      else
         throw new IllegalArgumentException("Nekorektna vrednost domine");
      if(b >= 0 && b <= 6)
         this.b = b;
      else
         throw new IllegalArgumentException("Nekorektna vrednost domine");
   }

   public int vratiA()
   {
      return a;
   }

   public int vratiB()
   {
      return b;
   }

   public int zbir()
   {
      return a + b;
   }

   public boolean jednake(Domina domina)
   {
      if((a == domina.a && b == domina.b) || (a == domina.b && b == domina.a))
         return true;
      else
         return false;
   }

   public String toString()
   {
      return "[" + a + " | " + b + "]";
   }
}
```

KolekcijaDomina.java

```
package domine;
```

```
import java.util.*;

public class KolekcijaDomina
{
   private Vector<Domina> domine = new Vector<Domina>();

   public KolekcijaDomina()
   {
      for(int i = 0; i <= 6; i++)
         for(int j = i; j <= 6; j++)
            domine.add(new Domina(i, j));
   }

   public void promesaj()
   {
      Collections.shuffle(domine);
   }

   public int velicina()
   {
      return domine.size();
   }

   public Domina vratiDominu(int i)
   {
      return domine.get(i);
   }

   // Izbaci iz kolekcije dominu sa datom kombinacijom brojeva
   public boolean ukloni(Domina domina)
   {
      Iterator<Domina> iter = domine.iterator();
      while(iter.hasNext())
      {
         if(iter.next().jednake(domina))
         {
            iter.remove();
            return true;
         }
      }
      return false;
   }
}
```

DomineAplet.java

```
package domine;

import javax.swing.BorderFactory;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
```

```
import java.util.Vector;

public class DomineAplet extends JApplet
{
   static final long serialVersionUID = 1;

   final static int za_izbor = 5;
   private int brojac = 0;
   private Vector<JButton> dugmeta_izbor = new Vector<JButton>();
   private Vector<Domina> domine_izbor = new Vector<Domina>();
   private KolekcijaDomina kolekcija_domina = new KolekcijaDomina();
   private JButton izabrano_dugme;
   private Domina izabrana_domina;

   private JPanel panel_unos;
   private JTextField unos_a, unos_b;
   private JLabel labela_poruka;

   private JPanel panel_izbor;
   private JButton izaberi;

   private JPanel panel_ispis;

   private JPanel panel_rezultat;
   private JButton ukupno_tacaka;
   private JTextField polje_rezultat;

   public void init()
   {
      setSize(450, 450);
      setLayout(new GridLayout(0,1));

      kolekcija_domina.promesaj();
      for(int i=0; i<kolekcija_domina.velicina(); i++)
         System.out.println(kolekcija_domina.vratiDominu(i));
      System.out.println();

      panel_unos = new JPanel(new GridLayout(0,1));
      panel_unos.setBorder(BorderFactory.createTitledBorder(
                  BorderFactory.createEtchedBorder(Color.BLUE, Color.CYAN),
                           "Izbor domina"));


      JPanel izbor1 = new JPanel();
      JLabel labela_a = new JLabel("Broj tacaka na jednoj polovini domine:");
      unos_a = new JTextField("");
      unos_a.setPreferredSize(new Dimension(40, 20));
      izbor1.add(labela_a);
      izbor1.add(unos_a);

      JPanel izbor2 = new JPanel();
      JLabel labela_b = new JLabel("Broj tacaka na drugoj polovini domine:");
      unos_b = new JTextField("");
      unos_b.setPreferredSize(new Dimension(40, 20));
      izbor2.add(labela_b);
      izbor2.add(unos_b);

      panel_unos.add(izbor1);
      panel_unos.add(izbor2);

      add(panel_unos);

      panel_izbor = new JPanel(new GridLayout(0, 1));

      labela_poruka = new JLabel("poruka");
      labela_poruka.setHorizontalAlignment(SwingConstants.CENTER);
      labela_poruka.setForeground(Color.RED);
```

```
panel_izbor.add(labela_poruka);

JPanel panel_izaberi = new JPanel();
izaberi = new JButton("Izaberi");
panel_izaberi.add(izaberi);
panel_izbor.add(panel_izaberi);
add(panel_izbor);

panel_ispis = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 20));
panel_ispis.setBorder(BorderFactory.createTitledBorder(
     BorderFactory.createEtchedBorder(Color.BLUE, Color.CYAN),
          "Prikaz izabranih domina"));
add(panel_ispis);

izaberi.addActionListener(
     new ActionListener()
     {
        public void actionPerformed(ActionEvent e)
        {
           try
           {
             String unos = unos_a.getText();
             int a = Integer.parseInt(unos);
             unos = unos_b.getText();
             int b = Integer.parseInt(unos);
             izabrana_domina = new Domina(a, b);
             labela_poruka.setText("");
             if(kolekcija_domina.ukloni(izabrana_domina))
             {
               // Ponovni ispis kolekcije u konzoli kako bi se videlo da
               // je domina zaista uklonjena
               for(int i=0; i<kolekcija_domina.velicina(); i++)
                 System.out.println(kolekcija_domina.vratiDominu(i));
               System.out.println();

               izabrano_dugme = new JButton(izabrana_domina.toString());
               izabrano_dugme.setPreferredSize(new Dimension(70, 25));
               panel_ispis.add(izabrano_dugme);
               dugmeta_izbor.add(izabrano_dugme);
               domine_izbor.add(izabrana_domina);

               izabrano_dugme.addMouseListener(
                    new MouseAdapter()
                    {
                      public void mouseEntered(MouseEvent e)
                      {
                        JButton dugme = (JButton)e.getComponent();
                        int index = dugmeta_izbor.indexOf(dugme);
                        dugme.setText(Integer.toString(
                             domine_izbor.elementAt(index).zbir()));
                        dugme.setForeground(Color.MAGENTA);
                      }
                      public void mouseExited(MouseEvent e)
                      {
                        JButton dugme = (JButton)e.getComponent();
                        int index = dugmeta_izbor.indexOf(dugme);
                        dugme.setText(domine_izbor.
                                      elementAt(index).toString());
                        dugme.setForeground(Color.BLACK);
                      }
                    }
               );

               brojac++;

               if(brojac == za_izbor)
               {
```

```
                labela_poruka.setText("Izabran je dovoljan broj domina!");
                unos_a.setEditable(false);
                unos_b.setEditable(false);
              }
            }
            else
            {
              labela_poruka.setText("Domina je vec izabrana");
            }
            validate();
          }
          catch(NumberFormatException nfe)
          {
            labela_poruka.setText("Domina nije korektno unesena!");
          }
          catch(IllegalArgumentException iae)
          {
            labela_poruka.setText("Domina nije korektno unesena!");
          }

        }
      }
);

panel_rezultat = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 20));
ukupno_tacaka = new JButton("Ukupan broj tacaka");
polje_rezultat = new JTextField("");
polje_rezultat.setPreferredSize(new Dimension(70, 25));
polje_rezultat.setEditable(false);
panel_rezultat.add(ukupno_tacaka);
panel_rezultat.add(polje_rezultat);

ukupno_tacaka.addActionListener(
     new ActionListener()
     {
       public void actionPerformed(ActionEvent e)
       {
            int zbir = 0;
            for(Domina domina : domine_izbor)
              zbir += domina.zbir();
            polje_rezultat.setText(Integer.toString(zbir));

       }
     }
);

polje_rezultat.addMouseListener(
   new MouseAdapter()
   {
     private Font stari_font;
     private Color stara_boja;

     public void mouseEntered(MouseEvent e)
     {
       stari_font = polje_rezultat.getFont();
       stara_boja = polje_rezultat.getForeground();
       polje_rezultat.setFont(new Font(stari_font.getFamily(),
            Font.ITALIC, stari_font.getSize() + 10));
       polje_rezultat.setForeground(Color.RED);
     }

     public void mouseExited(MouseEvent e)
     {
       polje_rezultat.setFont(stari_font);
       polje_rezultat.setForeground(stara_boja);

     }
```

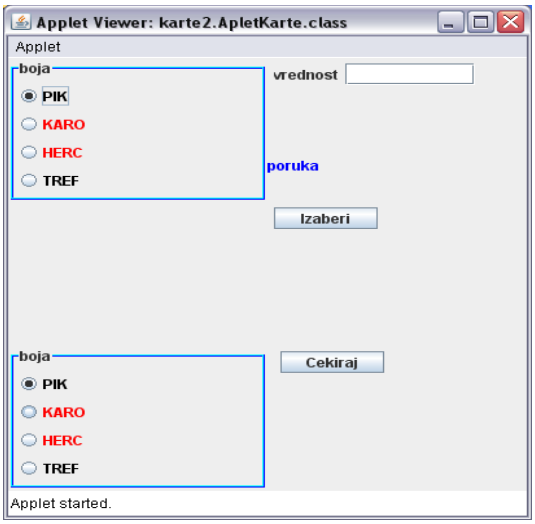
```
        }
      );

      add(panel_rezultat);
    }
}
```

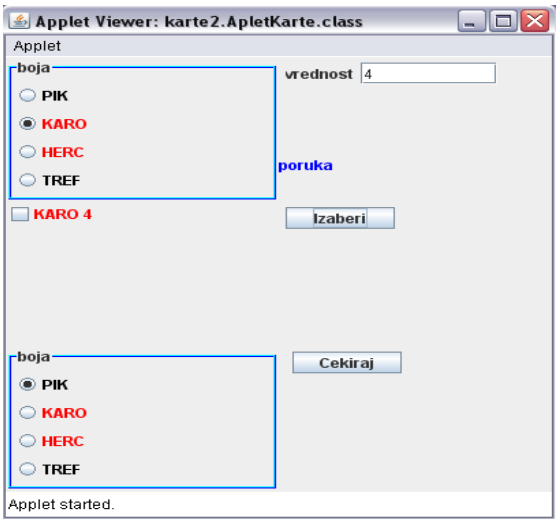
5. *Други колоквијум 2008, Карте*.

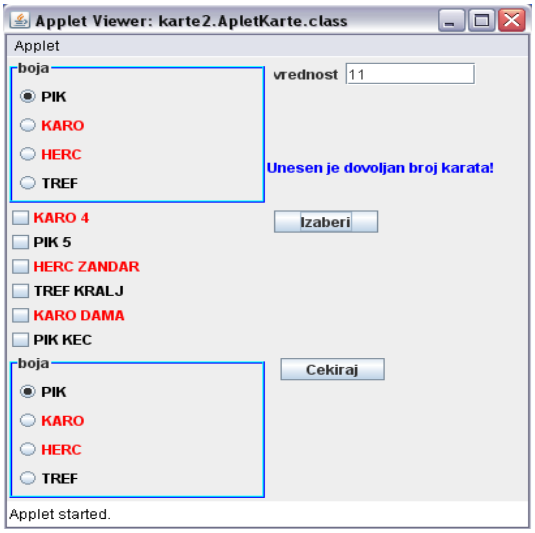
Написати аплет који садржи:

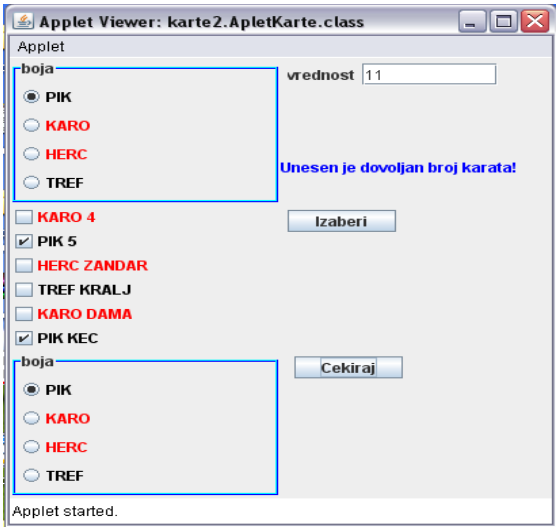
- Четири радио дугмета за избор боје карте са лабелама које одговарају могућим бојама (PIK, HERC, KARO, TREF). Дугмета треба да буду приказана оном бојом коју садрже (црна за TREF и PIK, црвена за KARO и HERC). Тачно једно дугме треба да буде селектовано.
- Поље за унос вредности карте са одговарајућом лабелом.
- Лабелу *poruka* за испис поруке.
- Дугме *Izaberi*.
    - Корисник бира тачно 6 карата, при чему не треба проверавати да ли је нека карта већ изабрана. ОБАВЕЗНО проверити да ли је вредност карте коректно унешена и у случају неисправног уноса у лабели *poruka* исписати одговарајућу поруку. Карта се бира кликом на дугме *Izaberi*. За сваку изабрану карту аутоматски се на површини аплета појављује *check-box* дугме са лабелом облика *boja_karte vrednost_karte* (на пример TREF 2). Лабела треба да се прикаже у боји изабране карте. Иницијално, *check-box* дугме које одговара изабраној карти није селектовано.
    - Након успешног бирања тачно 6 карата, сваки даљи клик на дугме изабери има за последицу испис поруке у лабели порука да је изабран довољан број карата.
- Четири радио дугмета истог облика као и претходна четири. Тачно једно дугме треба да буде селектовано.
- Дугме *Cekiraj*.
    - Избором неке од боја са радио дугмета и кликом на дугме *Cekiraj*, селектују се она *check-box* dugmeta која садрже карту у изабраној боји. Обезбедити да корисник не може кликом на *check-box* дугме да промени његово стање (селектовано/није селектовано).

Аплет иницијално треба да буде приказан као на првој слици. Друга слика показује избор карте и приказ *check-box* дугмета са изабраном картом. На трећој слици су приказане све изабране карте, а на четвртој реакција на клик на дугме *Cekiraj*.









Објашњење:

Класа *Karta* има две инстанцне променљиве, боју и вредност карте, обе типа *int*. Боја карте може бити нека од четири константне вредности задате статичким чланицама PIK, KARO, HERC і TREF.
Вредност карте припада интервалу [2, 14], при чему су за вредности 11, 12, 13, 14 дефинисане статичке константе KEC, ZANDAR, DAMA, KRALJ са тим вредностима, редом.
У конструктору се чланице *boja* и *vrednost* иницијализују на дате вредности, уколико су оне коректно задате. У супротном се избацује изузетак типа *IllegalArgumentException*.
String-репрезентација карте се гради постепено у зависности од боје карте и вредности карте. Чини је вредност боје за којом следи вредност карте.

Садржај аплета је организован у шест панела, при чему се као *LayoutManager* користи *GridLayout(0, 2)*.
Панел *panel1* садржи четири *radio* дугмета за избор боје карте са лабелама које одговарају могућим бојама. *Radio* дугмета се смештају у низ. Боје се такође чувају у низу тако да дугме и боја која му одговара буду на истим позицијама у датим низовима. Иницијално, селектовано је прво *radio* дугме у низу.

Панел *panel2* садржи текстуално поље *vrednost* за унос вредности карте, као и лабелу за испис порука. Као *LayoutManager* користи се *GridLayout(0, 1)*, при чему су текстуално поље и његова лабела организовани у један потпанел, а лабела за испис порука у други потпанел. Оба потпанела су организована у *FlowLayout(FlowLayout.LEFT)*.

Панел *panelRez* садржи шест *check-box* дугмади која иницијално немају никакав садржај у лабели и нису видљива. Дугмад се додају колекцији *checxBoxovi*, при чему је искоришћена класа *Vector*. Овакав начин је једноставнији када се зна колико дугмади треба да буде приказано. У супротном, дугмад морају да се додају једно по једно на панел и тада је неопходна употреба метода *validate()*, како би дугмад била видљива, јер нису део садржаја аплета при стартовању.

Панел *panel4* садржи дугме *izaberi*.
Панел *panel5* има исти облик као и *panel1*.
Панел *panel6* садржи дугме *cekiraj* и организован је на исти начин као и *panel4*.

Обрада догађаја обухвата:

a) Клик на дугме *izaberi*. Из поља *vrednost* учитава се вредност карте и проверава се њена коректност, у смислу да ли се ради о целом броју. У супротном се избацује изузетак типа *NumberFormatException*. Изабрана боја одређује се на основу селектованог *radio* дугмета, тако што се из низа *boje* бира она боја која има исти индекс као и селектовано дугме у низу *dugmeta_boje*. Када су *vrednost* и *boja* коректно изабрани, креира се карта, додаје у колекцију изабраних карата, а вредност променљиве *brojac* се повећава за *1*. Уколико је променљива достигла вредност *zaIzbor* (6), онемогућавају се даље акције над дугметом *izaberi* позивом метода *setEnabled(false)*. Истовремено се у лабели *poruka* исписује одговарајућа порука.
У колекцији *checkBoxovi* проналази се оно дугме које има исти индекс као и изабрана карта у колекцији изабраних карата. *String*-репрезентација те карте постаје садржај лабеле дугмета, док је боја за испис управо боја карте. Дугме постаје видљиво позивом метода *setVisible(true)*.
Истовремено се обрађује догађај везан за *check-box* дугме, који подразумева онемогућавање корисника да промени статус дугмета. Дакле, када се мишем уђе на површину дугмета (метод *mouseEntered()*), дугме постаје неактивно, чиме се корисник спречава да промени статус дугмета (селектовано/деселектовано).

b) Клик на дугме *cekiraj*. Прво се одреди која је тражена боја, на основу тога које је *radio* дугме селектовано. Потом се у колекцији *check-box* дугмади проналази оно чија лабела почиње називом тражене боје

```
checkBox.getText().startsWith(trazenaBoja[i].getText())
```

и то дугме се чекира.

Решење:

Karta.java

```
package karte;

public class Karta
{
   // Vrednosti boja
   public static final int PIK = 0;
   public static final int KARO = 1;
   public static final int HERC = 2;
   public static final int TREF = 3;

   // Vrednosti slika u kartama
   public static final int KEC = 11;
   public static final int ZANDAR = 12;
   public static final int DAMA = 13;
   public static final int KRALJ = 14;

   private int boja;
   private int vrednost;

   // Konstruktor za zadatu boju i vrednost
   public Karta(int boja, int vrednost) throws IllegalArgumentException
   {
      if(vrednost >= 2 && vrednost <= KRALJ)
         this.vrednost = vrednost;
      else
         throw new IllegalArgumentException("Nekorektna vrednost");
      if(boja >= PIK && boja <= TREF)
         this.boja = boja;
      else
         throw new IllegalArgumentException("Nekorektna boja");
   }

   public String toString()
   {
      String karta = "";
```

```
      switch(boja)
      {
      case TREF:
         karta = "TREF ";
         break;
      case KARO:
         karta = "KARO ";
         break;
      case HERC:
         karta = "HERC ";
         break;
      case PIK:
         karta = "PIK ";
         break;
      }

      switch(vrednost)
      {
      case KEC:
         karta += "KEC";
         break;
      case ZANDAR:
         karta += "ZANDAR";
         break;
      case DAMA:
         karta += "DAMA";
         break;
      case KRALJ:
         karta += "KRALJ";
         break;
      default:
         karta += Integer.toString(vrednost);
         break;
      }
      return karta;
   }
}
```

ApletKarte.java

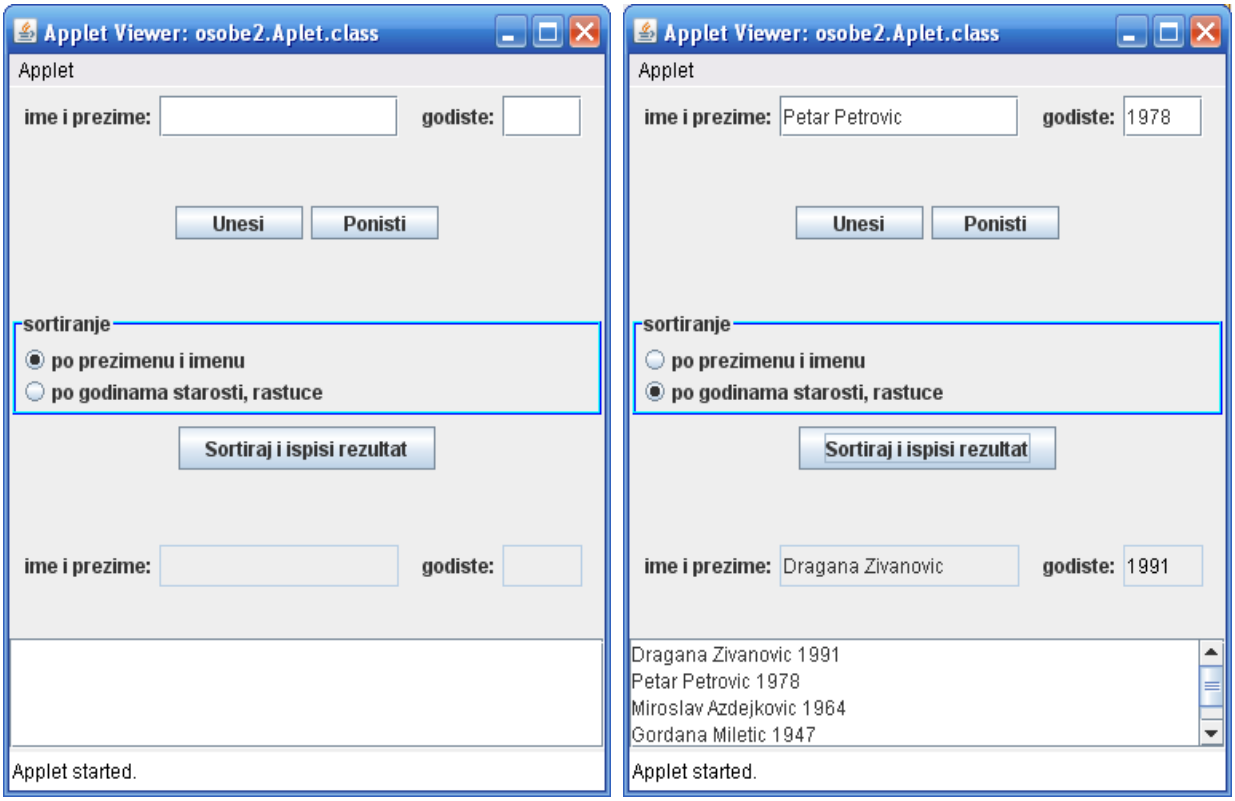
```
package karte;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

public class ApletKarte extends JApplet
{
   public static final long serialVersionUID = 1;
```

```
private JRadioButton[] dugmeta_boje=new JRadioButton[4];
private int boje[] = new int[4];
private JTextField vrednost;
private JButton izaberi;
private JLabel poruka;

final static int zaIzbor = 6;
private Vector<Karta> karte=new Vector<Karta>();
private int brojac = 0;
private Vector<JCheckBox> checkBoxovi=new Vector<JCheckBox>();
private JPanel panelRez=new JPanel(new GridLayout(0,1));

private JRadioButton[] trazenaBoja=new JRadioButton[4];
private JButton cekiraj=new JButton("Cekiraj");


public void init()
{
   setSize(400, 400);
   setLayout(new GridLayout(0,2));
   JPanel panel1=new JPanel(new GridLayout(0,1));
   panel1.setBorder(BorderFactory.createTitledBorder(
        BorderFactory.createEtchedBorder(Color.BLUE,Color.CYAN),"boja"));
   ButtonGroup group=new ButtonGroup();
   for(int i=0; i<4; i++){
      if(i == Karta.TREF)
      {
         dugmeta_boje[i] = new JRadioButton("TREF", false);
         dugmeta_boje[i].setForeground(Color.BLACK);
         boje[i] = Karta.TREF;

      }
      else if(i == Karta.HERC)
      {
         dugmeta_boje[i] = new JRadioButton("HERC", false);
         dugmeta_boje[i].setForeground(Color.RED);
         boje[i] = Karta.HERC;

      }
      else if(i == Karta.KARO)
      {
         dugmeta_boje[i] = new JRadioButton("KARO", false);
         dugmeta_boje[i].setForeground(Color.RED);
         boje[i] = Karta.KARO;
      }
      else
      {
         dugmeta_boje[i] = new JRadioButton("PIK", false);
         dugmeta_boje[i].setForeground(Color.BLACK);
         boje[i] = Karta.PIK;
      }
      group.add(dugmeta_boje[i]);
      panel1.add(dugmeta_boje[i]);
   }
   dugmeta_boje[0].setSelected(true);

   add(panel1);

   JPanel panel2=new JPanel(new GridLayout(0, 1));
   JPanel panel2A = new JPanel(new FlowLayout(FlowLayout.LEFT));
   panel2A.add(new JLabel("vrednost"));
   vrednost=new JTextField("");
   vrednost.setPreferredSize(new Dimension(100, 20));
   panel2A.add(vrednost);
   JPanel panel2B = new JPanel(new FlowLayout(FlowLayout.LEFT));
   poruka = new JLabel("poruka");
```

```
poruka.setForeground(Color.BLUE);
panel2B.add(poruka);
panel2.add(panel2A);
panel2.add(panel2B);

add(panel2);

for(int i = 0; i < zaIzbor; i++)
{
   JCheckBox check = new JCheckBox("");
   check.setVisible(false);
   checkBoxovi.add(check);
   panelRez.add(check);
}
add(panelRez);

JPanel panel4=new JPanel(new FlowLayout(FlowLayout.LEFT,10,5));
izaberi=new JButton("Izaberi");
izaberi.setPreferredSize(new Dimension(80,20));
izaberi.addActionListener(
     new ActionListener()
     {
        public void actionPerformed(ActionEvent e)
        {
           Karta karta;
           int boja = 0;
           for(int i=0; i< dugmeta_boje.length; i++)
             if(dugmeta_boje[i].isSelected())
             {
               boja = boje[i];
               break;
             }

           int vr = 0;
           try
           {
             vr = Integer.parseInt(vrednost.getText());
           }
           catch(NumberFormatException nfe)
           {
             poruka.setText("Greska!");
             return;
           }
           try
           {
              karta = new Karta(boja, vr);
           }
           catch(IllegalArgumentException iae)
           {
             poruka.setText(iae.getMessage());
             return;
           }

           karte.add(karta);
           brojac++;

           if(brojac == zaIzbor)
           {
             izaberi.setEnabled(false);
             poruka.setText("Izabran je dovoljan broj karata!");
           }

           JCheckBox cbox = checkBoxovi.elementAt(karte.indexOf(karta));
           cbox.setForeground(boja == Karta.TREF || boja == Karta.PIK ?
                 Color.BLACK : Color.RED);
           cbox.setText(karta.toString());
           cbox.setVisible(true);
```

```
            cbox.addMouseListener(
                new MouseAdapter()
                {
                  public void mouseEntered(MouseEvent e)
                  {
                    ((JCheckBox)e.getComponent()).setEnabled(false);
                  }

                  public void mouseExited(MouseEvent e)
                  {
                    ((JCheckBox)e.getComponent()).setEnabled(true);
                  }
                }

            );
          }
        }
    );

    add(panel4);
    panel4.add(izaberi);

    JPanel panel5=new JPanel(new GridLayout(0,1));
    panel5.setBorder(BorderFactory.createTitledBorder(
         BorderFactory.createEtchedBorder(Color.BLUE,Color.CYAN),"boja"));
    ButtonGroup group1=new ButtonGroup();
    for(int i=0; i<4; i++)
    {
      trazenaBoja[i] = new JRadioButton(dugmeta_boje[i].getText(), false);
      trazenaBoja[i].setForeground(dugmeta_boje[i].getForeground());
      panel5.add(trazenaBoja[i]);
      group1.add(trazenaBoja[i]);
    }
    trazenaBoja[0].setSelected(true);

    add(panel5);

    JPanel panel6=new JPanel(new FlowLayout(FlowLayout.LEFT,10,5));
    cekiraj.setPreferredSize(new Dimension(80,20));
    add(panel6);
    panel6.add(cekiraj);
    cekiraj.addActionListener(
         new ActionListener(){
           public void actionPerformed(ActionEvent e)
           {
             int i;
             for(i=0; i<trazenaBoja.length; i++)
               if(trazenaBoja[i].isSelected())
                 break;

             for(JCheckBox checkBox:checkBoxovi)
             {
               if(checkBox.getText().startsWith(trazenaBoja[i].getText()))
                 checkBox.setSelected(true);
               else
                 checkBox.setSelected(false);
             }

           }
         }
    );

  }
}
```

6. *други колоквијум 2008, Особе 2.* Написати аплет чији графички кориснички интерфејс би иницијално требало да изгледа као на првој приказаној слици:



а нпр. као на другој приказаној слици након одговарајућих интеракција од стране корисника, тј. аплет треба да има поље за унос имена и презимена особе са одговарајућом лабелом испред, а затим у истом реду и поље за унос годишта те особе и одговарајућу лабелу испред тог поља. У прво од ових поља име и презиме уносе се раздвојени једним или већим бројем размака.

У случају да није исправно унето име и презиме у поље за унос имена и презимена уписати "Неисправан унос!", а уколико је неисправно унето годиште у поље за унос имена и презимена уписати "Неисправно годиште!".

У другом реду треба да буду 2 дугмета: "Унеси" и "Поништи". Притиском на дугме "Унеси", унете податке о особи треба смештати у колекцију особа, а притисак на дугме "Поништи" треба да поништи уносе у горња 2 текстуална поља.

Даље, треба да постоје и 2 радио-дугмета са лабелама: "по презимену и имену" и "по годинама старости, растуће", а испод њих и дугме "Сортирај и испиши резултат". Радио-дугмад сместити у панел са одговарајућом границом (*border*).

Иницијално треба да буде селектовано радио-дугме "по презимену и имену".

Кликом на дугме "Сортирај и испиши резултат" треба сортирати податке о унетим особама по критеријуму изабраном селектованим радио-дугметом и одговарајуће резултате исписати у део графичког корисничког интерфејса који се налази испод.

Део графичког корисничког интерфејса предвиђен за испис резултата садржи у првом реду лабелу "име и презиме:", затим поље за унос текста у коме је онемогућен унос, а онда и лабелу "годиште:" и још једно поље за унос текста у коме је такође онемогућен унос. У наредном реду треба да се налази поље које може да прихвати више редова текста, а које по потреби може да се *scroll*-ује. У ова поља потребно је кликом на дугме "Сортирај и испиши резултат" уписати следеће: у први ред податке о особи која је прва у колекцији сортираној по изабраном критеријуму, а у доње поље читаву колекцију сортирану по изабраном критеријуму, и то податке о свакој особи у посебном реду.

Прелазак курсора миша преко поља на којима је онемогућен унос треба да промени боју исписаног текста у тим пољима на црвену и да повећа фонт за 5 пиксела. Излазак курсора миша из области ових поља треба да врати њихов првобитни изглед.
Поставити величину аплета тако да када се аплет покрене све његове компоненте буду видљиве и распоређене као на датим сликама.

Објашњење:

Дефинисана је помоћна класа *Osoba* са инстанцним променљивама *ime, prezime* и *godiste* и статичком променљивом *izborLeksikografski*. Неопходно је да класа имплементира *Comparable<>* интерфејс, како би било могуће сортирати колекцију објеката класе коришћењем статичког метода *sort()* класе *Collections* из пакета *java.util*. Треба омогућити лексикографско сортирање по презимену и имену особе, као и сортирање по годинама старости, у растућем поретку. Статичком променљивом *izborLeksikografski* одређено је да ли ће се вршити лексикографско сортирање или не. У методу *compareTo()* се, на основу постављене вредности ове променљиве, одређује однос текућег и задатог објекта класе *Osoba*.

Осим компоненти које учествују у обради догађаја, класа *Aplet* има и инстанцну променљиву *osobe* типа *ArrayList<Osoba>*. Овај објекат је иницијално празан, а сваким притиском на дугме "Унеси", уколико су исправно унети сви неопходни подаци, у њега ће бити додаван по један објекат типа *Osoba*.
Layout manager аплета је *GridLayout manager* који има једну колону, а број врста је 0, што значи да ће их бити довољно да се смести сав садржај који буде додат. На површ аплета затим се, редом, додају следећи панели:

- *panel1*, са потпанелима *panel11* (лабела и текстуално поље за унос имена и презимена) и *panel12* (лабела и текстуално поље за унос годишта).
- *panel2* (дугмад "Унеси" и "Поништи")
- *panel3* (*GridLayout manager* и *TitledBorder* граница, радио-дугмад за избор начина сортирања)
- *panel4* (дугме "Сортирај и испиши резултат")
- *panel5*, са потпанелима *panel51* (лабела и текстуално поље за испис имена и презимена) и *panel52* (лабела и текстуално поље за испис годишта). У текстуалним пољима је позивима метода *setEditable()* са аргументом *false* онемогућен унос корисника.
- *panel5* (*JScrollPane* типа, у коме се налази *JTextArea* компонента за приказ сортиране листе особа)

Следе објашњења обрада догађаја (сви ослушкивачи догађаја су објекти одговарајућих анонимних класа):

- дугме *unesi*: очитава се текст који је корисник унео у текстуално поље *ime* за унос имена и презимена. Позивом метода *indexOf()* одређује се позиција прве белине у том тексту. Део од почетка текста до те позиције узима се за име особе, док се од осталог дела методом *trim()* одсецају белине са почетка и краја и резултат се узима за презиме особе. Очитава се и текст који је корисник унео у текстуално поље *godiste* за унос годишта и методом *Integer.parseInt()* врши издвајање унетог целог броја. Од добијених података креира се објекат типа *Osoba* и додаје у колекцију *osobe*. Уколико неки податак није исправно унет, у пољу за унос имена и презимена исписује се одговарајућа порука.
- дугме *ponisti*: постављањем празног стринга у текстуална поља за унос имена и презимена и годишта "бришу" се евентуални уноси корисника из тих поља
- радио-дугмад *sortiranjeLeksikografski* и *sortiranjeNumericki*: постављање вредности статичке променљиве *izborLeksikografski* класе *Osoba* на одговарајућу вредност.
- дугме *sortiraj*: у случају празне колекције *osobe*, у текстуално поље за испис имена и презимена уписује се порука да је колекција празна, а иначе се методом *sort()* врши сортирање колекције у жељеном поретку и у поља за испис имена и презимена и годишта уписују одговарајући подаци прве особе из сортиране колекције. Име и презиме се претходно

раздвајају једном белином, а од броја који представља годиште потребно је добити одговарајући *String*, што се постиже позивом статичког метода *valueOf()* класе *String*. Текст који се исписује у *JTextArea* компоненти добија се тако што се пође од празног *StringBuffer* објекта, а затим се, у петљи, пролази кроз сортирану колекцију особа и надовезују подаци о текућој особи, а затим и знак за нови ред. Методом *toString()* се од резултујућег *StringBuffer* објекта добија одговарајући *String* објекат.

- обрада догађаја ниског нивоа, текстуална поља *imePrezime* и *god*: сваком од ових поља придружује се по један ослушкивач догађаја миша. У класама ослушкивача, које су анонимне класе изведене из класе *MouseAdapter*, декларисана је инстанцна променљива *stari* која чува изглед фонта када курсор миша није изнад површи одговарајуће компоненте. У методу *mouseEntered()* у променљивој *stari* се упамти стари изглед фонта, а затим се за текући фонт постави исти такав, само за 5 пиксела већи. Такође, боја текста се постави на црвену. У методу *mouseExited()* фонт се враћа на *stari*, а боја на црну.

Решење:

Osoba.java

```
package osobe2;

public class Osoba implements Comparable<Osoba>
{
   private static boolean izborLeksikografski = true;

   private String ime;
   private String prezime;
   private int godiste;

   public Osoba(String ime, String prezime, int godiste)
   {
      this.ime = ime;
      this.prezime = prezime;
      this.godiste = godiste;
   }

   public int compareTo(Osoba osoba)
   {
      if (izborLeksikografski)
      {
         int rezultat = prezime.compareTo(osoba.prezime);
         if (rezultat == 0)
            return ime.compareTo(osoba.ime);
         return rezultat;
      } else
         return osoba.godiste - godiste;
   }

   public String getIme()
   {
      return ime;
   }

   public String getPrezime()
   {
      return prezime;
   }

   public int getGodiste()
   {
      return godiste;
   }

   public static void setIzborLeksikografski(boolean izbor)
```

```
   {
      izborLeksikografski = izbor;
   }

}
```

Aplet.java

```
package osobe2;

import javax.swing.JApplet;
import javax.swing.SwingUtilities;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JRadioButton;
import javax.swing.JButton;

import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;

import java.awt.Container;
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.Dimension;
import java.awt.Color;
import java.awt.Font;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.ArrayList;
import java.util.Collections;

import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

public class Aplet extends JApplet
{
   private JTextField ime = new JTextField();
   private JTextField godiste = new JTextField();

   private JButton unesi;
   private JButton ponisti;

   private JRadioButton sortiranjeLeksikografski;
   private JRadioButton sortiranjeNumericki;

   private JTextField imePrezime = new JTextField();
   private JTextField god = new JTextField();

   private JButton sortiraj;

   private ArrayList<Osoba> osobe = new ArrayList<Osoba>();

   private JTextArea izlaz = new JTextArea();

   public void init()
   {
      SwingUtilities.invokeLater(new Runnable()
      {
         public void run()
         {
            createGUI();
         }
```

```
   });
}

private void createGUI()
{
   setSize(400, 400);
   Container content = getContentPane();
   content.setLayout(new GridLayout(0, 1));
   JPanel panel1 = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
   JPanel panel11 = new JPanel();
   JLabel imeLabel = new JLabel("ime i prezime:");
   panel11.add(imeLabel);
   panel11.add(ime);
   ime.setPreferredSize(new Dimension(150, 25));
   panel1.add(panel11);

   JPanel panel12 = new JPanel();
   JLabel godisteLabel = new JLabel("godiste:");
   panel12.add(godisteLabel);
   panel12.add(godiste);
   godiste.setPreferredSize(new Dimension(50, 25));
   panel1.add(panel12);

   content.add(panel1);

   JPanel panel2 = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));

   panel2.add(unesi = new JButton("Unesi"));
   unesi.setPreferredSize(new Dimension(80, 20));
   unesi.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
      {
         String imeprezimeS = ime.getText();
         String imeS = null;
         String prezimeS = null;
         // razdvajamo ime i prezime
         int belina = imeprezimeS.indexOf(" ");
         if (belina == -1)
         {
            ime.setText("Neispravan unos");
            return;
         } else
         {
            imeS = imeprezimeS.substring(0, belina);
            prezimeS = imeprezimeS.substring(belina + 1).trim();
         }
         int god = 0;
         try
         {
            god = Integer.parseInt(godiste.getText());
         } catch (NumberFormatException nfe)
         {
            ime.setText("Neispravno godiste!");
            return;
         }

         osobe.add(new Osoba(imeS, prezimeS, god));

      }
   });

   panel2.add(ponisti = new JButton("Ponisti"));
   ponisti.setPreferredSize(new Dimension(80, 20));
   ponisti.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
```

```
        {
           ime.setText("");
           godiste.setText("");
        }
     });

     content.add(panel2);

     JPanel panel3 = new JPanel(new GridLayout(0, 1));
     panel3.setBorder(
BorderFactory.createTitledBorder(
BorderFactory.createEtchedBorder(Color.blue, Color.cyan),
"sortiranje"));
     ButtonGroup group = new ButtonGroup();
     panel3.add(sortiranjeLeksikografski =
new JRadioButton("po prezimenu i imenu", true));
     panel3.add(sortiranjeNumericki =
new JRadioButton("po godinama starosti, rastuce"));
     group.add(sortiranjeLeksikografski);
     group.add(sortiranjeNumericki);
     sortiranjeLeksikografski.addActionListener(new ActionListener()
     {
        public void actionPerformed(ActionEvent e)
        {
           Osoba.setIzborLeksikografski(true);
        }
     });

     sortiranjeNumericki.addActionListener(new ActionListener()
     {
        public void actionPerformed(ActionEvent e)
        {
           Osoba.setIzborLeksikografski(false);
        }
     });

     content.add(panel3);

     JPanel panel4 = new JPanel();
     content.add(panel4);
     panel4.add(sortiraj = new JButton("Sortiraj i ispisi rezultat"));
     sortiraj.addActionListener(new ActionListener()
     {
        public void actionPerformed(ActionEvent e)
        {
           if (osobe.size() != 0)
           {
              Collections.sort(osobe);
              imePrezime.setText(osobe.get(0).getIme() + " "
                  + osobe.get(0).getPrezime());
              god.setText(String.valueOf(osobe.get(0).getGodiste()));

              StringBuffer str = new StringBuffer();
              for (Osoba osoba : osobe)
              {
                 str.append(osoba.getIme() + " " + osoba.getPrezime() + " "
                           + osoba.getGodiste() + "\n");
              }
              izlaz.setText(str.toString());

           } else
              imePrezime.setText("Prazna kolekcija!");
        }
     });

     JPanel panel5 = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 5));
     JPanel panel51 = new JPanel();
```

```
     panel51.add(new JLabel("ime i prezime:"));
     imePrezime.setSize(new Dimension(200, 20));
     imePrezime.setPreferredSize(new Dimension(150, 25));
     imePrezime.setEditable(false);
     panel51.add(imePrezime);
     imePrezime.addMouseListener(new MouseAdapter()
     {
        private Font stari;

        public void mouseEntered(MouseEvent e)
        {
           stari = imePrezime.getFont();
           imePrezime.setFont(
new Font(stari.getFamily(), stari.getStyle(),
 stari.getSize() + 5));
           imePrezime.setForeground(Color.RED);
        }

        public void mouseExited(MouseEvent e)
        {
           imePrezime.setFont(stari);
           imePrezime.setForeground(Color.BLACK);
        }

     });
     panel5.add(panel51);

     JPanel panel52 = new JPanel();
     panel52.add(new JLabel("godiste:"));
     panel52.add(god);
     god.setPreferredSize(new Dimension(80, 20));
     god.setEditable(false);
     god.setPreferredSize(new Dimension(50, 25));
     god.addMouseListener(new MouseAdapter()
     {
        private Font stari;

        public void mouseEntered(MouseEvent e)
        {
           stari = god.getFont();
           god.setFont(
new Font(stari.getFamily(), stari.getStyle(),
 stari.getSize() + 5));
           god.setForeground(Color.RED);
        }

        public void mouseExited(MouseEvent e)
        {
           god.setFont(stari);
           god.setForeground(Color.BLACK);
        }
     });
     panel5.add(panel52);

     content.add(panel5);

     JScrollPane panel6 = new JScrollPane(izlaz);
     content.add(panel6);

   }

}
```

7. *Колоквијум 2009, Исказне формуле.*

У математичкој логици исказне формуле се дефинишу на следећи начин:

- исказна слова и логичке константе су исказне формуле

- негација, коњункција, дисјункција, импликација и еквиваленција исказних формула су исказне формуле.

1. Апстрактна класа *IskaznaFormula* описује исказне формуле у математичкој логици. За сваку исказну формулу треба омогућити одређивање њене вредности и одређивање њене *String* репрезентације.
2. Класа *IskaznoSlovo* представља исказна слова, при чему се свако исказно слово карактерише својим симболом и својом вредношћу.
3. Класа *Negacija* представља негацију исказне формуле.
4. Класе *Konjunkcija*, *Disjunkcija*, *Implikacija* и *Ekvivalencija* представљају редом конјункцију, дисјункцију, импликацију и еквиваленцију две исказне формуле.

У свакој од класа под 2. 3. и 4. имплементирати:

а) констуктор за креирање објеката на основу свих потребних података
б) копи – конструктор за креирање копије објекта
в) неопходне *get*()* и *set*()* методе
г) метод за одређивање вредности исказне формуле
д) метод који враћа *String* репрезентацију објекта класе

Формати исписа:

Класа *Negacija*	not(formula)
Класа *Konjunkcija*	formula && formula
Класа *Disjunkcija*	formula | formula
Класа *Implikacija*	formula => formula
Класа *Ekvivalencija*	formula <=> formula

Приликом исписа, потформула мора бити ограђена заградама, осим ако је у питању исказно слово или негација исказног слова.

Све инстанцне променљиве класа декларисати као *private*.

Обезбедити да се полиморфно извршава метод:

boolean vrednost();

за рачунање вредности исказне формуле.

5. Написати тест класу у којој за формуле:

(p ⇒ q) ⇔(¬p ∨ q) i
(¬q ⇒ ¬p) ∨ (p ⇒ q)

треба урадити следеће:

а) креирати објекат који представља конкретну формулу
б) исписати формулу
в) тестирати да ли је формула таутологија и исписати одговарајућу таблицу као и одговарајућу поруку

Пример:

(p ⇒ q) ⇔(¬p ∨ q)
p q (p ⇒ q) ⇔(¬p ∨ q)
false false vrednost
false true vrednost
true false vrednost
true true vrednost
Poruka

Објашњење:

Класа *IskaznaFormula* је апстрактна и садржи само два апстрактна метода – *vrednost()* и *toString()*. Вредност исказне формуле може бити *true* или *false*.

Класа *IskaznoSlovo* има две инстанцне променљиве. Свако исказно слово има свој симбол и сваком слову је могуће доделити вредност – *true/false*.

Садржи следеће методе:

- Подразумевани конструктор
- Конструктор са једним аргументом поставља одговарајући симбол за исказно слово, при чему се слову придружује вредност *false*
- Конструктор са два аргумента поставља и симбол и вредност исказног слова на дате вредности
- Конструктор копије
- Метод *postaviNa()* поставља вредност слова на задату вредност
- Метод *vrednost()* дефинише наслеђени метод из базне класе. Враћа се вредност исказног слова.
- Метод *toString()* дефинише наслеђени метод из базне класе. *String*-репрезентацију исказног слова чини његов симбол. Пошто је чланица *slovo* типа *char*, њена конверзија у *String* врши се позивом статичког метода *toString()* класе *Character*: `Character.toString(slovo)`.

Инстанцна чланица класе *Negacija* је исказна формула која се негира. Вредност негације добија се управо негирањем њене вредности.

Класе *Konjunkcija*, *Disjunkcija*, *Implikacija* и *Ekvivalencija* имају исту структуру, пошто дефинишу бинарне логичке операторе. Разликују се дефиниције метода *vrednost()*, док се у методу *toString()* разликује једино симбол којим је оператор представљен. Инстанцне чланице у свакој од ових класа су две исказне формуле, леви и десни операнд.
У телу конструктора сваке од ових класа је битно да се инстанцне чланице иницијализују датим вредностима без коришћења оператора *new*. Дакле, треба креирати конјункцију, дисјункцију, импликацију, односно, еквиваленцију две задате формуле, а не њихових идентичних копија.
Ради се овако, јер су за рачунање вредности формуле потребне вредности њених потформула.

String-репрезентација се гради постепено, па се користи објекат класе *StringBuffer*.
Уколико је леви операнд исказно слово или негација исказне формуле, није потребно оградити га заградама. Провера се врши оператором *instanceof*. На *String*-репрезентацију левог операнда допише се симбол који означава оператор, затим се допише и *String*-репрезентација десног операнда, након што се и за њега уради иста провера. Тако добијени *String* представља тражену репрезентацију.

У класи *Konjunkcija*, метод *vrednost()* враћа вредност која се добија као резултат примене оператора && логичке конјнкције на вредност левог и десног операнда. У класи *Disjunkcija* се само примени оператор логичке дисјункције ||.
У класама *Implikacija* и *Ekvivalencija* метод *vrednost()* се дефинише у складу са таблицама за импликацију и еквиваленцију.

У тест класи се најпре креира прва исказна формула, постепено, почев од исказних слова. Није неопходно чувати референце на потформуле, пошто се не тражи исписивање њихових вредности, већ само вредности исказних слова и целе формуле. Дакле, прва формула се може креирати и на следећи начин:

```
new Ekvivalencija(new Implikacija(p, q),new Disjunkcija(new Negacija(p), q));
```

Да би се исписала таблица вредности формуле и испитало да ли је формула таутологија, за све могуће комбинације вредности исказних слова, тј. за све валуације, одреди се вредност формуле. За два исказна слова могуће комбинације су (*false*, *false*), (*false*, *true*), (*true*, *false*) и (*true*, *true*). Формула је таутологија ако су све добијене вредности *true*, тј. ако је за сваку валуацију формула тачна.
Исти поступак понавља се и за другу исказну формулу.

Решење:

IskaznaFormula.java

```
package iskazneFormule;

// Bazna klasa iskaznih formula.
```

```
public abstract class IskaznaFormula
{
   // Apstraktni metod za racunanje vrednosti iskazne formule
   public abstract boolean vrednost();

   // Apstraktni metod za dobijanje String reprezentacie iskazne formule
   public abstract String toString();
}
```

IskaznoSlovo.java

```
package iskazneFormule;

// Klasa koja predstavlja iskazno slovo
public class IskaznoSlovo extends IskaznaFormula
{
   /* Svako iskazno slovo ima svoj simbol i svakom slovu je
    * moguce dodeliti vrednost - true/false
    */
   private char slovo;
   private boolean vrednost;

   // Podrazumevani konstruktor
   public IskaznoSlovo()
   {}

   /* Konstruktor koji postavlja odgovarajuci simbol,
    * pri cemu se slovu pridruzuje vrednost false.
    */
   public IskaznoSlovo(char slovo)
   {
      this.slovo = slovo;
   }

   // Konstruktor koji postavlja i simbol i vrednost iskaznog slova
   public IskaznoSlovo(char slovo, boolean vrednost)
   {
      this.slovo = slovo;
      this.vrednost = vrednost;
   }

   // Kopi-konstruktor
   public IskaznoSlovo(final IskaznoSlovo s)
   {
      this(s.slovo, s.vrednost());
   }

   // Postavlja vrednost slova na zadatu
   public void postaviNa(boolean vrednost)
   {
      this.vrednost = vrednost;
   }

   // Definicija nasledjenog metoda vrednost() iz bazne klase
   public boolean vrednost()
   {
      return vrednost;
   }

   // Definicija nasledjenog metoda toString()
   public String toString()
   {
      return Character.toString(slovo);
   }
}
```

Negacija.java

```
package iskazneFormule;

package iskazneFormule;

public class Negacija extends IskaznaFormula
{
   private IskaznaFormula formula;

   public Negacija(IskaznaFormula formula)
   {
      this.formula = formula;
   }

   // Konstruktor kopije
   public Negacija(final Negacija neg)
   {
      this(neg.formula);
   }

   // String reprezentacija negacije
   public String toString()
   {
      return "not(" + formula + ")";
   }

   public boolean vrednost()
   {
      return !formula.vrednost();
   }
}
```

Konjunkcija.java

```
package iskazneFormule;

public class Konjunkcija extends IskaznaFormula
{
   private IskaznaFormula leva_strana;
   private IskaznaFormula desna_strana;

   public Konjunkcija()
   {}

   public Konjunkcija(IskaznaFormula leva, IskaznaFormula desna)
   {
      leva_strana = leva;
      desna_strana = desna;

   }

   public Konjunkcija(final Konjunkcija konj)
   {
      this(konj.leva_strana, konj.desna_strana);
   }

   public String toString()
   {
      StringBuffer ispis = new StringBuffer();

      if(leva_strana instanceof IskaznoSlovo
                  || leva_strana instanceof Negacija)
         ispis.append(leva_strana);
      else
      {
```

```
        ispis.append("(");
        ispis.append(leva_strana);
        ispis.append(")");
      }

      ispis.append(" && ");

      if(desna_strana instanceof IskaznoSlovo
                  || desna_strana instanceof Negacija)
        ispis.append(desna_strana);
      else
      {
        ispis.append("(");
        ispis.append(desna_strana);
        ispis.append(")");
      }
      return ispis.toString();
    }

    public boolean vrednost()
    {
      return leva_strana.vrednost() && desna_strana.vrednost();
    }
}
```

Disjunkcija.java

```
package iskazneFormule;

public class Disjunkcija extends IskaznaFormula
{
    private IskaznaFormula leva_strana;
    private IskaznaFormula desna_strana;

    public Disjunkcija()
    {}

    public Disjunkcija(IskaznaFormula leva, IskaznaFormula desna)
    {
      leva_strana = leva;
      desna_strana = desna;
    }

    public Disjunkcija(final Disjunkcija disj)
    {
      this(disj.leva_strana, disj.desna_strana);
    }

    public String toString()
    {
      StringBuffer ispis = new StringBuffer();

      if(leva_strana instanceof IskaznoSlovo
                || leva_strana instanceof Negacija)
        ispis.append(leva_strana);
      else
      {
        ispis.append("(");
        ispis.append(leva_strana);
        ispis.append(")");
      }

      ispis.append(" || ");

      if(desna_strana instanceof IskaznoSlovo
                || desna_strana instanceof Negacija)
```

```
        ispis.append(desna_strana);
      else
      {
        ispis.append("(");
        ispis.append(desna_strana);
        ispis.append(")");
      }
      return ispis.toString();
   }

   public boolean vrednost()
   {
      return leva_strana.vrednost() || desna_strana.vrednost();
   }
}
```

Implikacija.java

```
package iskazneFormule;

public class Implikacija extends IskaznaFormula
{
   private IskaznaFormula leva_strana;
   private IskaznaFormula desna_strana;

   public Implikacija()
   {}

   public Implikacija(IskaznaFormula leva, IskaznaFormula desna)
   {
      leva_strana = leva;
      desna_strana = desna;
   }

   public Implikacija(final Implikacija impl)
   {
      this(impl.leva_strana, impl.desna_strana);
   }

   public String toString()
   {
      StringBuffer ispis = new StringBuffer();

      if(leva_strana instanceof IskaznoSlovo
                || leva_strana instanceof Negacija)
        ispis.append(leva_strana);
      else
      {
        ispis.append("(");
        ispis.append(leva_strana);
        ispis.append(")");
      }

      ispis.append(" => ");

      if(desna_strana instanceof IskaznoSlovo
                || desna_strana instanceof Negacija)
        ispis.append(desna_strana);
      else
      {
        ispis.append("(");
        ispis.append(desna_strana);
        ispis.append(")");
      }
      return ispis.toString();
   }
```

```
   public boolean vrednost()
   {
      if(leva_strana.vrednost() && !desna_strana.vrednost())
         return false;
      return true;
   }
}
```

Ekvivalencija.java

```
package iskazneFormule;

public class Ekvivalencija extends IskaznaFormula
{
   private IskaznaFormula leva_strana;
   private IskaznaFormula desna_strana;

   public Ekvivalencija()
   {}

   public Ekvivalencija(IskaznaFormula leva, IskaznaFormula desna)
   {
      leva_strana = leva;
      desna_strana = desna;
   }

   public Ekvivalencija(final Ekvivalencija ekv)
   {
      this(ekv.leva_strana, ekv.desna_strana);
   }

   public String toString()
   {
      StringBuffer ispis = new StringBuffer();

      if(leva_strana instanceof IskaznoSlovo
                || leva_strana instanceof Negacija)
         ispis.append(leva_strana);
      else
      {
         ispis.append("(");
         ispis.append(leva_strana);
         ispis.append(")");
      }

      ispis.append(" <=> ");

      if(desna_strana instanceof IskaznoSlovo
                || desna_strana instanceof Negacija)
         ispis.append(desna_strana);
      else
      {
         ispis.append("(");
         ispis.append(desna_strana);
         ispis.append(")");
      }
      return ispis.toString();
   }

   public boolean vrednost()
   {
      if(leva_strana.vrednost() == desna_strana.vrednost())
         return true;
      return false;
   }
}
```

TestIskazneFormule.java

```
package iskazneFormule;

public class TestIskazneFormule
{
   public static void main(String[] args)
   {
      IskaznaFormula formula;

      // Prva iskazna formula: (p => q) <=> (not(p) || q)
      System.out.println("Prva formula:");
      IskaznoSlovo q = new IskaznoSlovo('q');
      IskaznoSlovo p = new IskaznoSlovo('p');

      Negacija nep = new Negacija(p);

      Implikacija leva = new Implikacija(p, q);
      Disjunkcija desna = new Disjunkcija(nep, q);

      formula = new Ekvivalencija(leva, desna);
      /* Drugi nacin:
       * formula = new Ekvivalencija(new Implikacija(p, q),
       *                             new Disjunkcija(new Negacija(p), q));
       */

      System.out.println(formula);
      System.out.println(p + "\t" + q + "\t" + formula);

      // Moguce vrednosti iskaznih slova
      boolean[] valuacije = {false, true};

      boolean ind = true;
      for(boolean p1 : valuacije)
         for(boolean q1 : valuacije)
         {
            p.postaviNa(p1);
            q.postaviNa(q1);
            System.out.println(p.vrednost() + "\t" + q.vrednost() + "\t\t" +
                               formula.vrednost());
            if(!formula.vrednost())
            {
               ind = false;
            }
         }

      if(ind)
         System.out.println("Formula je tautologija");
      else
         System.out.println("Formula nije tautologija");


      /* Isti postupak primenjuje se i za drugu formulu:
       * (not(q) => not(p)) || (p => q)
       */
      System.out.println("\nDruga formula:");
      IskaznoSlovo Q = new IskaznoSlovo('q');
      IskaznoSlovo P = new IskaznoSlovo('p');

      Negacija neQ = new Negacija(Q);
      Negacija neP = new Negacija(P);

      Implikacija leva_strana = new Implikacija(neQ, neP);
      Implikacija desna_strana = new Implikacija(P, Q);
```

```
      formula = new Disjunkcija(leva_strana, desna_strana);
      /* Drugi nacin:
       * formula = new Disjunkcija(
       *                 new Implikacija(
       *                         new Negacija(Q),
       *                         new Negacija(P)),
       *                 new Implikacija(P, Q));
       */
      System.out.println(formula);
      System.out.println(P + "\t" + Q + "\t" + formula);

      ind = true;
      for(boolean p1 : valuacije)
        for(boolean q1 : valuacije)
        {
          P.postaviNa(p1);
          Q.postaviNa(q1);
          System.out.println(P.vrednost() + "\t" + Q.vrednost()
                             + "\t\t" + formula.vrednost());
          if(!formula.vrednost())
            ind = false;

        }

      if(ind)
        System.out.println("Formula je tautologija");
      else
        System.out.println("Formula nije tautologija");
    }
}
```

8. *Колоквијум 2009, Шах (краљ, ловац, топ)*.

1. Класа *Polje* представља једно поље на шаховској табли.

Садржи:

- две инстанцне промењљиве које означавају координате позиције поља на табли. Прва координата узима вредности из скупа {’A’, ’B’, ’C’, ’D’, ’E’, ’F’, ’G’, ’H’}, а друга из скупа {’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’}
- трећа инстанцна променљива означава да ли се на пољу налази фигура.

Имплементирати:

- конструктор са вредностима за координате поља на табли
- метод за проверу исправности координата поља
- неопходне *get*()* и *set*()* методе
- метод који враћа *String*- репрезентацију поља (нпр. “A3”, “B6”, “G8”)

2. Апстрактна класа *Figura* представља фигуру на шаховској табли.

Инстанцне промењљиве класе су:

- *boja* (типа String)
- *polje* (типа Polje) – поље на шаховској табли на коме се налази фигура

Класе *Kraljica* і *Skakac* описују врсте фигура које постоје на шаховској табли.

У свакој од наведених класа имплементирати:

a) конструктор за креирање фигуре на основу задате боје и поља, при чему је потребно бројати колико је креирано белих, а колико црних фигура
   На табли може постојати највише једна црно/бела краљица и највише два црна/бела скакача.

b) метод *boolean korektanPotez(Polje p)* којим се проверава да ли је померање фигуре на дато поље *p* коректно.

У класи *Figura* имплементирати метод

boolean pomeriFiguru(Polje p)

којим се врши померање фигуре на дато поље *p* – враћа *true* ако је померање успешно, иначе *false*. Уколико на пољу *p* већ постоји нека фигура само исписати поруку *„Фигуре се туку“*.

Имплементирати и метод који враћа *String*- репрезентацију фигуре.

Примери:
KraljicaC polje
SkakacB polje

3. Написати тест класу *SahovskaTabla* у којој се најпре креира шаховска табла – квадратна матрица димензије 8x8 чији су елементи типа *Polje*.
Потом се са стандардног улаза учитава број фигура које ће бити креиране.
Фигуре се смештају у низ.
Након тога се учитавају фигуре тако што се најпре учита врста фигуре (краљица или скакач).
За сваку врсту фигуре учитати боју и при том водити рачуна да се не прекорачи максимални дозвољени број фигура дате врсте у датој боји. У случају неуспеха, поновити унос фигуре.
Кад је боја коректно задата, унети и координате поља са шаховске табле на које се поставља дата фигура. Неопходно је проверити коректност задатих координата поља, као и да ли се на том пољу већ налази нека фигура. У случају неуспеха, поновити унос фигуре.
Након креирања фигура, 3 пута се случајно бира фигура из датог низа и исписује се њена *String* репрезентација.
Потом се уносе координате поља на које треба померити фигуру. Ако је померање било успешно исписати *String* репрезентацију фигуре, иначе исписати поруку *„Nekorektno pomeranje“*.

Објашњење:

Класа *Polje* дефинише једно поље на шаховској табли. Поље се описује координатама *x* и *y*, у складу са ознакама појединачних поља на шаховској табли, на пример A4, B8, H1. Координате су типа *char*.
Чланица *zauzeto* указује да ли се на пољу налази фигура, или не.

Статички метод *ispravneKoordinate(int, int)* проверава да ли су дате координате исправне координате једног поља. Координате су исправне ако прва припада секвенци [A, H], а друга секвенци [1, 8].
Метод *postaviFiguruNaPolje()* поставља чланицу *zauzeto* na *true*, док је метод *pomeriFiguruSaPolja()* поставља на *false*.
Метод *figuraNaPolju()* враћа *true*, ако се на пољу налази фигура, у супротном *false*.

Класа *Figura* је апстрактна класа којом се описује фигура на шаховској табли. Инстанцне чланице су боја фигуре, типа *String* и поље на коме се налази фигура, типа *Polje*.
У конструктору, приликом иницијализације чланице *polje*, не треба правити копију поља које је задато као аргумент. Фигура треба да се постави на постојеће поље на шаховској табли, тако да чланица *polje* треба да реферише на њега. Поље се означава заузетим позивом метода *postaviFiguruNaPolje()*.

Метод *pomeriFiguru(Polje)* помера фигуру на дато поље и враћа *true* ако је померање успело.
Процедура померања је иста за сваку фигуру и подразумева најпре проверу коректности потеза, тј. проверу да ли је дозвољено померити фигуру на поље *p*. Ако потез није дозвољен, враћа се *false*. У супротном, проверава се да ли на пољу *p* већ постоји нека фигура и, ако постоји, исписује се порука *"Figure se tuku"* и враћа *false*. Дакле, фигура се у овом случају не помера, јер би то захтевало имплементацију операције уклањања фигуре са шаховске табле, што се није тражило у задатку. Ако је поље p слободно, фигура се помера тако што се ослободи поље на коме се тренутно налази, а ново поље на коме је фигура, је управо поље *p;* поље *p* постаје заузето.
Пошто је процедура померања иста за све фигуре, метод је дефинисан у базној класи. Једино што је специфично за сваку фигуру јесте провера коректности потеза, стога је метод *korektanPotez()* апстрактан. Провера да ли су координате поља *p* коректне врши се у тест класи.

Метод *toString()* дефинисан је у базној класи употребом оператора *instanceof* за проверу типа фигуре.

У класама *Kraljica* и *Skakac* дефинисане су статичке константе MAX_CRNIH и MAX_BELIH, које представљају максимални дозвољени број црних, односно, белих фигура тог типа. Такође,

дефинисане су и статичке чланице *broj_belih* и *broj_crnih*, које броје беле, односно, црне фигуре тог типа. Класе немају инстанцне чланице.
У конструктору се позива конструктор наткласе да постави боју фигуре и поље на које се фигура поставља. Проверава се које је боје фигура и повећава одговарајући бројач.
Провера да ли је креиран максималан дозвољени број краљица/скакача одговарајуће боје врши се у тест класи, не у конструктору. Дакле, у тест класи се прво провери колико тренутно има краљица/скакача дате боје, па ако није достигнут максимум, креира се нова фигура.

Краљица може да се помера дијагонално, вертикално и хоризонтално у односу на текући положај. Рачуна се вредност разлике *x* и у координата текућег поља и поља на које треба померити краљицу. Ако померање није дијагонално `(Math.abs(x) != Math.abs(y)`, није хоризонтално и није вертикално, враћа се *false*, иначе, ако бар један од услова није задовољен, враћа се *true*.

Померање скакача је коректно ако су апсолутне вредности разлике *x* и *y* координата текућег поља и поља на које треба померити скакача једнаке 1, односно 2 или обрнуто.

У тест класи се прво креира шаховска табла. Потом се са стандарног улаза задаје број фигура које ће бити распоређене на табли. Фигуре се чувају у низу. Фигура се учитава тако што се прво учита врста фигуре, а потом и боја фигуре и проверава да ли је достигнут максималан дозвољени број фигура дате врсте у датој боји. Ако јесте, учитавање се понавља.
Потом се учитавају координате поља где фигура треба да буде постављена. Пошто су координате типа *char*, учитавање се врши позивом `skener.next().charAt(0);`. Уколико координате поља нису коректне, учитавање се понавља.
Потом се из низа учитаних фигура случајно бирају 3 фигуре и врши се њихово померање на поље чије се координате уносе са улаза. Ако је померање фигуре успешно, испише се нова позиција фигуре, у супротном порука да је померање некоректно.

Решење:

Polje.java

```
package figureSah1;

public class Polje
{
   private char x;
   private char y;

   // Clanica koja oznacava da li se na polju nalazi figura.
   private boolean zauzeto;

   public Polje(char x, char y)
   {
      this.x = x;
      this.y = y;
   }

   public static boolean ispravneKoordinate(int x, int y)
   {
      return x >= 'A' && x <= 'H' && y >= '1' && y <= '8';
   }

   public void postaviX(char x)
   {
      this.x = x;
   }

   public void postaviY(char y)
   {
      this.y = y;
```

```
   }

   public void postaviFiguruNaPolje()
   {
      zauzeto = true;
   }

   public void pomeriFiguruSaPolja()
   {
      zauzeto = false;
   }

   public boolean figuraNaPolju()
   {
      return zauzeto;
   }

   public char vratiX()
   {
      return x;
   }

   public char vratiY()
   {
      return y;
   }

   public String toString()
   {
      return "" + x + y;
   }
}
```

Figura.java

```
package figureSah1;

public abstract class Figura
{
   private String boja;
   private Polje polje;

   public Figura(String boja, Polje p)
   {
      this.boja = boja;
      this.polje = p;
      p.postaviFiguruNaPolje();
   }

   public boolean pomeriFiguru(Polje p)
   {
      if(korektanPotez(p))
      {
         if(p.figuraNaPolju())
         {
            System.out.println("Postoji figura na polju: figure se tuku!");
            return false;
         }
         this.vratiPolje().pomeriFiguruSaPolja();
         this.postaviPolje(p);
         p.postaviFiguruNaPolje();

         return true;
      }
      return false;
   }
```

```
   public abstract boolean korektanPotez(Polje p);

   public String vratiBoju()
   {
      return boja;
   }

   public Polje vratiPolje()
   {
      return polje;
   }

   // Postavlja novo polje na kome se nalazi figura.
   public void postaviPolje(Polje polje)
   {
      this.polje = polje;
   }

   public String toString()
   {
      StringBuffer ispis = new StringBuffer();
      if(this instanceof Kraljica)
         ispis.append("Kraljica");
      else if(this instanceof Skakac)
         ispis.append("Skakac");

      if(boja.equalsIgnoreCase("bela"))
         ispis.append("B");
      else if(boja.equalsIgnoreCase("crna"))
         ispis.append("C");

      ispis.append(" ");
      ispis.append(polje);
      return ispis.toString();
   }
}
```

Kraljica.java

```
package figureSah1;

public class Kraljica extends Figura
{
   // Maksimalan dozvoljeni broj belih i crnih kraljica
   public static final int MAX_CRNIH = 1;
   public static final int MAX_BELIH = 1;

   /* Staticke clanice koje broje koliko je kreirano belih,
    * a koliko crnih kraljica
    */
   public static int broj_belih;
   public static int broj_crnih;

   public Kraljica(String boja, Polje p)
   {
      super(boja, p);
      if(boja.equalsIgnoreCase("bela"))
         broj_belih++;
      else
         broj_crnih++;
   }

   // Provera korektnosti poteza
   public boolean korektanPotez(Polje p)
   {
```

```
      int x = p.vratiX() - vratiPolje().vratiX();
      int y = p.vratiY() - vratiPolje().vratiY();
      if(Math.abs(x) != Math.abs(y) && x != 0 && y != 0)
         return false;
      else
         return true;
   }
}
```

Skakac.java

```
package figureSah1;

public class Skakac extends Figura
{
   // Maksimalan dozvoljeni broj belih i crnih skakaca
   public static final int MAX_CRNIH = 2;
   public static final int MAX_BELIH = 2;

   /* Staticke clanice koje broje koliko je kreirano belih,
    * a koliko crnih skakaca
    */
   public  static int broj_belih;
   public static int broj_crnih;

   public Skakac(String boja, Polje p)
   {
      super(boja, p);
      if(boja.equalsIgnoreCase("bela"))
         broj_belih++;
      else
         broj_crnih++;
   }

   public boolean korektanPotez(Polje p)
   {
      int x = Math.abs(p.vratiX() - vratiPolje().vratiX());
      int y = Math.abs(p.vratiY() - vratiPolje().vratiY());

      if((x == 1 && y == 2) || (x == 2 && y == 1))
         return true;
      return false;
   }
}
```

SahovskaTabla.java

```
package figureSah1;

import java.util.Random;
import java.util.Scanner;

public class SahovskaTabla
{
   public static void main(String[] args)
   {
      Scanner skener = new Scanner(System.in);
      Random izbor = new Random();

      Polje[][] tabla = new Polje[8][8];
      // Kreira se sahovska tabla
      for(int i = 0; i < 8; i++)
         for(int j = 0; j < 8; j++)
         {
            char x = 'A';
            x += i;
```

```
      char y = '1';
      y += j;
      tabla[i][j]= new Polje(x, y);
   }

System.out.println("Unesi broj figura na sahovskoj tabli:");
int n = skener.nextInt();

Figura[] figure = new Figura[n];

for(int i = 0; i < n; i++)
{
   System.out.print("Vrsta figure: ");
   String vrsta = skener.next();
   if(vrsta.equalsIgnoreCase("kraljica"))
   {
      System.out.print("Boja figure: ");
      String boja = skener.next();
      if(boja.equalsIgnoreCase("bela"))
      {
         if(Kraljica.broj_belih == Kraljica.MAX_BELIH)
         {
            System.out.println("Bela kraljica je rasporedjena!");
            i--;
            continue;
         }
      }
      else if(boja.equalsIgnoreCase("crna"))
      {
         if(Kraljica.broj_crnih == Kraljica.MAX_CRNIH)
         {
            System.out.println("Crna kraljica je rasporedjena!");
            i--;
            continue;
         }
      }

      System.out.println("Pozicija figure: ");
      char x = skener.next().charAt(0);
      char y = skener.next().charAt(0);
      if(!Polje.ispravneKoordinate(x, y))
      {
         System.out.println("Neispravne koordinate polja - pokusajte ponovo");
         i--;
         continue;
      }

      Polje p = tabla[x-'A'][y-'1'];
      if(p.figuraNaPolju())
      {
         System.out.println("Polje je zauzeto.");
         i--;
         continue;
      }
      figure[i] = new Kraljica(boja, p);
   }
   else if(vrsta.equalsIgnoreCase("skakac"))
   {
      System.out.print("Boja figure: ");
      String boja = skener.next();
      if(boja.equalsIgnoreCase("bela"))
      {
         if(Skakac.broj_belih == Skakac.MAX_BELIH)
         {
            System.out.println("Dva bela skakaca rasporedjena!");
            i--;
            continue;
```

```
            }
         }
         else if(boja.equalsIgnoreCase("crna"))
         {
            if(Skakac.broj_crnih == Skakac.MAX_CRNIH)
            {
               System.out.println("Dva crna skakaca rasporedjena!");
               i--;
               continue;
            }
         }

         System.out.println("Pozicija figure: ");
         char x = skener.next().charAt(0);
         char y = skener.next().charAt(0);
         if(!Polje.ispravneKoordinate(x, y))
         {
            System.out.println("Neispravne koordinate polja - pokusajte ponovo");
            i--;
            continue;
         }

         Polje p = tabla[x-'A'][y-'1'];
         if(p.figuraNaPolju())
         {
            System.out.println("Polje je zauzeto.");
            i--;
            continue;
         }
         figure[i] = new Skakac(boja, p);
      }
   }

   // Iz niza se slucajno biraju tri figure koje se pomeraju na zadato polje
   Figura izabrana;
   for(int i = 0; i < 3; i++)
   {
      izabrana = figure[izbor.nextInt(figure.length)];
      System.out.println("Izabrana figura:");
      System.out.println(izabrana);
      System.out.println("Polje na koje se pomera:");
      char x = skener.next().charAt(0);
      char y = skener.next().charAt(0);
      if(Polje.ispravneKoordinate(x, y))
         if(izabrana.pomeriFiguru(tabla[x - 'A'][y - '1']))
            System.out.println("Nakon pomeranja: " + izabrana);
         else
            System.out.println("Nekorektno pomeranje");

   }
  }
}
```

9. *Колоквијум 2009, Шах (краљ, ловац и топ).*

1. Класа *Polje* представља једно поље на шаховској табли.

Садржи:

- две инстанцне промењљиве које означавају координате позиције поља на шаховској табли. Обе координате узимају вредности из скупа {0, 1, 2, 3, 4, 5, 6, 7}.
- трећа инстанцна промењљива означава да ли се на пољу налази фигура.

Имплементирати:

- конструктор са вредностима за координате поља на табли
- метод за проверу исправности координата поља
- неопходне *get*()* и *set*()* методе
- метод који враћа *String* репрезентацију поља (нпр. “A3”, “B6”, “G8”)

2. Апстрактна класа *Figura* представља фигуру на шаховској табли.
Инстанцне променљиве класе су:
- *boja* (типа int)
- *polje* (типа Polje) – поље на шаховској табли на коме се налази фигура

Класе *Kralj, Lovac и Top* описују врсте фигура које постоје на шаховској табли.
У свакој од наведених класа имплементирати:
а) конструктор за креирање фигуре на основу задате боје и поља, при чему је потребно бројати колико је креирано белих, а колико црних фигура
На табли може постојати највише један црн/бели краљ, највише два црна/бела ловца и највише два црна/бела топа.
б) метод *boolean korektanPotez(Polje p)* којим се проверава да ли је померање фигуре на дато поље *p* коректно.

У класи *Figura* имплементирати метод
boolean pomeriFiguru(Polje p) којим се врши померање фигуре на дато поље *p* – враћа *true* ако је померање успешно, иначе *false*. Уколико на пољу *p* већ постоји нека фигура само исписати поруку *„Figure se tuku“* и вратити *true*.
Имплементирати и метод који враћа *String* репрезентацију фигуре.
Пример *String* репрезентације:

KraljC polje
KraljB polje

3. Написати тест класу *SahovskaTabla* у којој се најпре креира шаховска табла – квадратна матрица димензије 8x8 чији су елементи типа *Polje*.
Потом се са стандардног улаза учитава број фигура које ће бити креиране случајним избором.
Фигуре се смештају у низ.
Након тога се псеудослучајно бирају поље са табле (водити рачуна да мора бити слободно), затим врста и боја и креирају фигуре на основу тако изабраних података. Приликом избора боје водити рачуна да се не прекорачи максимални дозвољени број фигура дате врсте у датој боји. У случају неуспеха, поновити унос фигуре.
Након креирања фигура, 3 пута се случајно бира фигура из датог низа и исписује се њена *String* репрезентација.
Потом се уносе координате поља на које треба померити фигуру. Ако је померање било успешно исписати нову *String* репрезентацију фигуре, иначе исписати поруку *„Nekorektno pomeranje“*.

Објашњење:

Класа *Polje* дефинише једно поље на шаховској табли. Поље се описује координатама *x* и *y*, које одговарају индексима у квадратној матрици димензије 8. Координате су типа *int*.
Чланица *zauzeto* указује да ли се на пољу налази фигура, или не.

Статички метод *ispravneKoordinate(int, int)* проверава да ли су дате координате исправне координате једног поља. Координате су исправне ако припадају секвенци [0, 7].
Метод *postaviFiguruNaPolje()* поставља чланицу *zauzeto* на *true*, док је метод *pomeriFiguruSaPolja()* поставља на *false*.
Метод *figuraNaPolju()* враћа *true*, ако се на пољу налази фигура, у супротном *false*.

Класа *Figura* је апстрактна класа којом се описује фигура на шаховској табли. Инстанцне чланице су боја фигуре, типа *int* и поље на коме се налази фигура, типа *Polje*.
Класа садржи и статичке константе KRALJ, LOVAC, TOP којим се одређује врста фигуре, као и статичке константе BELA, CRNA којим се одређује боја фигуре.
У конструктору, приликом иницијализације чланице *polje*, не треба правити копију поља које је задато као аргумент. Фигура треба да се постави на постојеће поље на шаховској табли, тако да

чланица *polje* треба да реферише на њега. Поље се означава заузетим позивом метода *postaviFiguruNaPolje()*.

Метод *pomeriFiguru(Polje)* помера фигуру на дато поље и враћа *true* ако је померање успело.
Процедура померања је иста за сваку фигуру и подразумева најпре проверу коректности потеза, тј. проверу да ли је дозвољено померити фигуру на поље *p*. Ако потез није дозвољен, враћа се *false*. У супротном, проверава се да ли на пољу *p* већ постоји нека фигура и, ако постоји, исписује се порука *"Figure se tuku"* и враћа *false*. Дакле, фигура се у овом случају не помера, јер би то захтевало имплементацију операције уклањања фигуре са шаховске табле, што се није тражило у задатку. Ако је поље p слободно, фигура се помера тако што се ослободи поље на коме се тренутно налази, а ново поље на коме је фигура, је управо поље *p;* поље *p* постаје заузето.
Пошто је процедура померања иста за све фигуре, метод је дефинисан у базној класи. Једино што је специфично за сваку фигуру јесте провера коректности потеза, те је стога метод *korektanPotez()* апстрактан. Провера да ли су координате поља *p* коректне врши се у тест класи.

Метод *toString()* дефинисан је у базној класи употребом оператора *instanceof* за проверу типа фигуре.

У класама *Kralj, Lovac* и *Top* дефинисане су статичке константе MAX_CRNIH и MAX_BELIH, које представљају максимални дозвољени број црних, односно, белих фигура те врсте. Такође, дефинисане су и статичке чланице *broj_belih* и *broj_crnih*, које броје беле, односно, црне фигуре те врсте. Класе немају инстанцне чланице.
У конструктору се позива конструктор наткласе да постави боју фигуре и поље на које се фигура поставља. Проверава се које је боје фигура и повећава одговарајући бројач.
Провера да ли је креиран максималан дозвољени број фигура дате врсте и одговарајуће боје врши се у тест класи, не у конструктору. Дакле, у тест класи се прво провери колико тренутно има фигура дате врсте у датој боји, па ако није достигнут максимум, креира се нова фигура.

Краљ може да се помера на на било које од суседних поља (хоризонтално, вертикално и дијагонално у односу на текући положај). Рачуна се апсолутна вредност разлике *x* и у координата текућег поља и поља на које треба померити краља. Потез је коректан ако добијене апсолутне вредности нису веће од 1 и нису истовремено једнаке 0.

Померање ловца је коректно ако су апсолутне вредности разлике *x* и *y* координата текућег поља и поља на које треба померити ловца једнаке, јер ловац може да се помера само дијагонално.

Топ може да се помера хоризонтално и вертикално у односу на тренутни положај. Коректан је онај потез где апслолутне вредности разлике *x* и *y* координата текућег поља и поља на које треба померити топа нису истовремено различите од нуле.

У тест класи се прво креира шаховска табла. Потом се са стандарног улаза задаје број фигура које ће бити распоређене на табли. Фигуре се случајно бирају и чувају се у низу. Прво се случајно бира поље на које треба поставити фигуру, а потом врста и боја фигуре. Проверава се да ли је достигнут максималан дозвољени број фигура изабране врсте у изабраној боји. Ако јесте, избор се понавља.
Потом се учитавају координате поља где фигура треба да буде постављена. Уколико координате поља нису коректне, учитавање се понавља.
Потом се из низа учитаних фигура случајно бирају 3 фигуре и врши се њихово померање на задато поље. Ако је померање фигуре успешно, испише се нова позиција фигуре, у супротном порука да је померање некоректно.

Решење:

Polje.java

```
package figureSah2;
```

```
public class Polje
{
   private int x;
   private int y;

   // Clanica koja oznacava da li se na polju nalazi figura.
   private boolean zauzeto;

   public Polje(int x, int y)
   {
      this.x = x;
      this.y = y;
   }

   public static boolean ispravneKoordinate(int x, int y)
   {
      return x >= 0 && x <= 7 && y >= 0 && y <= 7;
   }

   public void postaviX(int x)
   {
      this.x = x;
   }

   public void postaviY(int y)
   {
      this.y = y;
   }

   public void postaviFiguruNaPolje()
   {
      zauzeto = true;
   }

   public void pomeriFiguruSaPolja()
   {
      zauzeto = false;
   }

   public boolean figuraNaPolju()
   {
      return zauzeto;
   }

   public int vratiX()
   {
      return x;
   }

   public int vratiY()
   {
      return y;
   }

   public String toString()
   {
      String ispis = new String();
      char slovo = 'A';
      slovo += x;
      ispis += slovo;
      ispis += (y + 1);
      return ispis;
   }

}
```

Figura.java

```
package figureSah2;

public abstract class Figura
{
   // Staticke konstante koje odradjuju vrstu figure
   public static final int KRALJ = 0;
   public static final int LOVAC = 1;
   public static final int TOP = 2;

   // Staticke konstante koje odredjuju boju figure
   public static final int BELA = 0;
   public static final int CRNA = 1;

   private int boja;
   private Polje polje;

   public Figura(int boja, Polje p)
   {
      this.boja = boja;
      this.polje = p;
      p.postaviFiguruNaPolje();
   }

   public boolean pomeriFiguru(Polje p)
   {
      if(korektanPotez(p))
      {
         if(p.figuraNaPolju())
         {
            System.out.println("Postoji figura na polju: figure se tuku!");
            return false;
         }
         this.vratiPolje().pomeriFiguruSaPolja();
         this.postaviPolje(p);
         p.postaviFiguruNaPolje();

         return true;
      }
      return false;
   }

   public abstract boolean korektanPotez(Polje p);

   public int vratiBoju()
   {
      return boja;
   }

   public Polje vratiPolje()
   {
      return polje;
   }

   // Postavlja novo polje na kome se nalazi figura.
   public void postaviPolje(Polje polje)
   {
      this.polje = polje;
   }

   public String toString()
   {
      StringBuffer ispis = new StringBuffer();
      if(this instanceof Kralj)
         ispis.append("Kralj");
      else if(this instanceof Lovac)
```

```
        ispis.append("Lovac");
      else if(this instanceof Top)
        ispis.append("Top");

      if(boja == BELA)
        ispis.append("B");
      else
        ispis.append("C");

      ispis.append(" ");
      ispis.append(polje);
      return ispis.toString();
   }
}
```

Kralj.java

```
package figureSah2;

public class Kralj extends Figura
{
   // Maksimalan dozvoljeni broj belih i crnih kraljeva
   public static final int MAX_CRNIH = 1;
   public static final int MAX_BELIH = 1;

   /* Staticke clanice koje broje koliko je kreirano belih,
    * a koliko crnih kraljeva
    */
   public static int broj_crnih;
   public static int broj_belih;

   public Kralj(int boja, Polje p)
   {
      super(boja, p);
      if(boja == Figura.BELA)
        broj_belih++;
      else
        broj_crnih++;
   }

   // Provera korektnosti poteza
   public boolean korektanPotez(Polje p)
   {
      int x = p.vratiX() - vratiPolje().vratiX();
      int y = p.vratiY() - vratiPolje().vratiY();

      if(Math.abs(x) <= 1 && Math.abs(y) <= 1 &&
              (Math.abs(x) != 0 || Math.abs(y) != 0))
           return true;
      return false;
   }
}
```

Lovac.java

```
package figureSah2;

public class Lovac extends Figura
{
   // Maksimalan dozvoljeni broj belih i crnih topova
   public static final int MAX_CRNIH = 2;
   public static final int MAX_BELIH = 2;

   /* Staticke clanice koje broje koliko je kreirano belih,
    * a koliko crnih topova
    */
```

```
   public static int broj_crnih;
   public static int broj_belih;

   public Lovac(int boja, Polje p)
   {
      super(boja, p);
      if(boja == Figura.BELA)
        broj_belih++;
      else
        broj_crnih++;

   }

   // Provara korektnosti poteza
   public boolean korektanPotez(Polje p)
   {
      int x = p.vratiX() - vratiPolje().vratiX();
      int y = p.vratiY() - vratiPolje().vratiY();

      if(Math.abs(x) != Math.abs(y))
        return false;
      else
        return true;
   }

}
```

Top.java

```
package figureSah2;

public class Top extends Figura
{
   // Maksimalan dozvoljeni broj belih i crnih topova
   public static final int MAX_CRNIH = 2;
   public static final int MAX_BELIH = 2;

   /* Staticke clanice koje broje koliko je kreirano belih,
    * a koliko crnih topova
    */
   public static int broj_crnih;
   public static int broj_belih;

   public Top(int boja, Polje p)
   {
      super(boja, p);
      if(boja == Figura.BELA)
        broj_belih++;
      else
        broj_crnih++;
   }

   // Provera korektnosti poteza
   public boolean korektanPotez(Polje p)
   {
      int x = p.vratiX() - vratiPolje().vratiX();
      int y = p.vratiY() - vratiPolje().vratiY();
      if(x != 0 && y != 0)
        return false;
      else
        return true;
   }
}
```

SahovskaTabla.java

```
package figureSah2;

import java.util.Random;
import java.util.Scanner;

public class SahovskaTabla
{

   public static void main(String[] args)
   {
      Scanner skener = new Scanner(System.in);
      Random izbor = new Random();

      Polje[][] tabla = new Polje[8][8];
      // Kreira se sahovska tabla
      for(int i = 0; i < 8; i++)
         for(int j = 0; j < 8; j++)
            tabla[i][j]= new Polje(i, j);

      System.out.println("Uneti broj figura na sahovskoj tabli:");
      int n = skener.nextInt();

      Figura[] figure = new Figura[n];

      for(int i = 0; i < n; i++)
      {
         int x = 0, y = 0;
         while(true)
         {
            x = izbor.nextInt(8);
            y = izbor.nextInt(8);
            if(!tabla[x][y].figuraNaPolju())
               break;
         }

         int vrsta = izbor.nextInt(3);
         int boja = izbor.nextInt(2);

         if(vrsta == Figura.KRALJ)
         {
            if(boja == Figura.BELA)
            {
               if(Kralj.broj_belih == Kralj.MAX_BELIH)
               {
                  i--;
                  continue;
               }
            }
            else
            {
               if(Kralj.broj_crnih == Kralj.MAX_CRNIH)
               {
                  i--;
                  continue;
               }
            }

            figure[i] = new Kralj(boja, tabla[x][y]);
         }

         else if(vrsta == Figura.LOVAC)
         {
            if(boja == Figura.BELA)
            {
               if(Lovac.broj_belih == Lovac.MAX_BELIH)
               {
```

```
                i--;
                continue;
             }
          }
          else
          {
             if(Lovac.broj_crnih == Lovac.MAX_CRNIH)
             {
                i--;
                continue;
             }
          }

          figure[i] = new Lovac(boja, tabla[x][y]);
       }

       else if(vrsta == Figura.TOP)
       {
          if(boja == Figura.BELA)
          {
             if(Top.broj_belih == Top.MAX_BELIH)
             {
                i--;
                continue;
             }
          }
          else
          {
             if(Top.broj_crnih == Top.MAX_CRNIH)
             {
                i--;
                continue;
             }
          }

          figure[i] = new Top(boja, tabla[x][y]);
       }
    }

    System.out.println("Postavljene figure na tabli:");
    for(int i = 0; i < n; i++)
       System.out.println(figure[i]);

    Figura izabrana;
    for(int i = 0; i < 3; i++)
    {
       izabrana = figure[izbor.nextInt(figure.length)];
       System.out.println("Izabrana figura:");
       System.out.println(izabrana);
       System.out.println("Polje na koje se pomera:");
       int x = skener.nextInt();
       int y = skener.nextInt();
       if(Polje.ispravneKoordinate(x, y))
          if(izabrana.pomeriFiguru(tabla[x][y]))
             System.out.println("Nakon pomeranja: " + izabrana);
          else
             System.out.println("Nekorektno pomeranje");

    }
  }
}
```

10. *колоквијум 2009, Нумерологија.* Написати класу *Osoba* која садржи инстанцне промењиве: име, презиме и адресу становања (све типа *String*).
OsobaD је особа за коју се додатно зна датум рођења (типа *Datum*).

OsobaJMBG је особа за коју се додатно зна јединствени матични број грађана (ЈМБГ, типа *String*). ЈМБГ је тринаестоцифрени број у коме прве две цифре, када се посматрају као цео број, представљају дан рођења особе, наредне две цифре посматране као број представљају месец рођења особе, а ако посматрамо наредне 3 цифре, и испред додамо цифру 1, то је година рођења особе (претпоставља се да су све особе које се разматрају рођене у прошлом миленијуму). Преосталих 6 цифара нису нам од значаја. Нпр. ако је особа рођена 29.02.1984. њен ЈМБГ је облика 2902984******, док је ЈМБГ особе рођене 02.02.1984. облика 0202984******.
Класа *Datum* од инстанцних променљивих има: дан, месец и годину (све типа *int*).
У свакој од наведених класа имплементирати:

- конструктор за креирање објеката када су доступни сви неопходни подаци
- копи-конструктор
- неопходне *get*()* и *set*()* методе
- метод који враћа *String*-репрезентацију објекта класе (Датум исписати у формату дд.мм.гггг.)

Обезбедити да се полиморфно извршавају методи:

```
1.) int numeroloskiBroj();
```

који треба да израчуна и врати нумеролошки број особе.
Нумеролошки број особе рачуна се на следећи начин: саберу се све цифре које се јављају у датуму рођења. Ако тако добијени збир није једноцифрен, саберу се његове цифре. Поступак се понавља све док се као резултат не добије једноцифрени број.
Нпр. ако је особа рођена 29.02.1984. сабирањем цифара 2+9+0+2+1+9+8+4 добија се збир 35 који није једноцифрен. Сабирањем његових цифара 3+5 добија се 8 што јесте једноцифрени број и поступак се ту завршава. Нумеролошки број особе је 8.

```
2.) String metabolizam(final Datum d);
```

који треба да врати *String* који се даље користи за пребројавање "троуглића" које особа има задатог дана *d*. Тражени *String* добија се на следећи начин: из *String*-репрезентација датума рођења и задатог датума побришу се тачке и оно што преостане посматра се као неозначени цео број. Тако добијена два броја се саберу, а оно што функција треба да врати је *String*-репрезентација резултата записана у пољу ширине 8 (ако је резултат 7-цифрени број, дописати водећу нулу).
Нпр. ако је особа рођена 29.02.1984. а као аргумент метода је задат датум 01.04.2009., сабирамо бројеве 29021984 и 01042009. Резултат је број 30063993, а наш метод треба да врати *String* у коме пише "30063993".
Допуштено је у класе додавати и друге методе за којима се укаже потреба при решавању задатка.
Написати тест-класу која ради следеће:

- захтева да корисник са стандардног улаза унесе број објеката, а затим:
- у петљи очекује да корисник унесе име, презиме, адресу особе
- након тога 1 ако се зна датум рођења, или 2 ако се зна ЈМБГ особе
- ако је унето 1, уноси се датум рођења особе (у облику дд.мм.гггг.) и креира одговарајући објекат, а ако је унето 2, уноси се ЈМБГ особе и креира одговарајући објекат
- све креиране објекте потребно је сместити у један низ одговарајућег типа
- по завршетку петље уноси се данашњи датум
- главни програм затим исписује *String*-репрезентацију, нумеролошки број и *String* за пребројавање троуглића оних особа чији нумеролошки број није 7 и које немају нула у данашњем *String*-у за троуглиће.
- неопходно је по уношењу сваког датума или ЈМБГ-а проверити да ли је тај унос валидан (нпр. 29.02.2009. није валидан датум (2009 није преступна) или 23.2.2009. (није облика дд.мм.гггг.) или 3.02.2009.(није облика дд.мм.гггг.) или 32.10.2008.(10. месец има 31 дан) као ни 03.02.2009 (нема тачку на крају) итд. Година је преступна ако је дељива са 400 или је дељива са 4, а није дељива са 100.

Објашњење:

У класи *Datum*, поред уобичајених метода (конструктор, копи-конструктор, *get*()* и *set*()* методи, *toString()*), имплементирани су и статички методи: *prestupna()* – испитивање да ли је дата година

преступна, *daniUmesecu()* – враћа број дана задатог месеца задате године (користи претходни метод), *validan()* – за задати дан, месец и годину проверава да ли су то валидни подаци за креирање објекта типа *Datum* (користи претходни метод) и *string2datum()* – врши конвертовање *String*-репрезентације датума у одговарајући објекат типа *Datum* (користи претходни метод). Метод *string2datum()* врши све неопходне провере: да ли је *String* дужине 11, да ли се на одговарајућим позицијама налазе тачке, да ли се на осталим позицијама налазе *String*-репрезентације података валидних за креирање објекта типа *Datum*. У методу *toString()*, ако се ради о датуму чији је дан једноцифрен број, испред *String*-репрезентације тог броја дописује се карактер '0'. Исто се чини и када је месец једноцифрени број.

Базна класа, *Osoba*, декларисана је као апстрактна, а апстрактни су методи *numeroloskiBroj()* и *metabolizam()* које желимо да позивамо користећи полиморфизам. У класи је имплементиран и помоћни метод *zbirCifara()* са пакетним правом приступа, који ће бити наслеђен у изведеним класама *OsobaD* и *OsobaJMBG* које су смештене у истом пакету. Овај метод рачуна збир цифара датог броја тако што у петљи, све док се не обраде све цифре броја (тј. док број не постане 0), последњу цифру броја (остатак при дељењу броја са 10) додаје на текући збир цифара (који се иницијално поставља на 0), а затим се та цифра уклања из броја његовим целобројним дељењем са 10.

Кључни делови изведених класа *OsobaD* и *OsobaJMBG* су полиморфни методи. Метод *numeroloskiBroj()*, у обе класе, најпре срачуна збир цифара из датума рођења, а онда, по потреби, понавља поступак рачунања збира цифара добијеног резултата све док се као резултат не добије једноцифрен број. У класи *OsobaD* метод рачуна збир цифара из датума рођења сабирањем збира цифара дана, збира цифара месеца и збира цифара године, док у класи *OsobaJMBG* то чини сабирањем цифра по цифра првих 7 цифара ЈМБГ-а и додавањем јединице која се у ЈМБГ-у имплицитно подразумева. Метод *metabolizam()* у класи *OsobaD* понаособ израчуна збирове дана, месеци и година датума рођења особе и задатог датума, множењем одговарајућим степеном броја 10 позиционира их на места која треба да заузму у резултату и сабере тако добијене резултате. Збир дана се, множењем са 1000000, помера 6 места улево остављајући са десне стране 2 места за збир месеци и 4 места за збир година. Збир месеци се, множењем са 10000, помера 4 места улево остављајући простор за збир година. Збир година заузима крајње десне позиције у резултату, те нема потребе за његовим множењем степеном броја 10 (множи се са 10^0). Овако израчунат, резултат ће имати водећу нулу у делу који представља збир месеци када је тај збир једноцифрен, што је добро. Резултат се, затим, из типа *int* конвертује у *String*. Остаје још да се, ако је потребно, а то је у случају када је збир дана једноцифрен, допише водећа нула испред њега. У класи *OsobaJMBG* метод *metabolizam()* од *String*-а који представља ЈМБГ особе креира нови издвајањем само дела *String*-а који се односи на датум рођења и уметањем јединице на одговарајућу позицију, а од дана, месеца и године задатог датума, на исти начин као што то ради истоимени метод класе *OsobaD* (њиховим множењем степенима броја 10 и сабирањем резултата), рачуна одговарајући цео број неопходан за рачунање троуглића. Затим се врши сабирање овог и броја чију *String*-репрезентацију представља *String* добијен из ЈМБГ-а особе на описани начин, а потом се добијени резултат конвертује из типа *int* у тип *String* и, по потреби, дописује водећа нула.

У класи *OsobaJMBG* имплементиран је и статички метод *validan()* који проверава да ли се у датом *String*-у налази коректан запис ЈМБГ-а, односно да ли је *String* дужине 13, састављен само од цифара и да ли је првих 6 цифара такво да носи информацију о валидном датуму.

Тест-класа је имплементирана прилично праволинијски. Дефинише се низ типа базне класе у који се смештају референце на креиране објекте изведених класа. Након уношења данашњег датума пролази се кроз низ и исписују подаци о особама које задовољавају тражени критеријум. Провера да *String* за троуглиће не садржи нулу врши се позивом метода *indexOf()* класе *String* који у том случају треба да врати вредност -1.

Решење:

Datum.java

```
package numerologija;

public class Datum
{
   private int dan;
   private int mesec;
   private int godina;

   public Datum(int dan, int mesec, int godina)
   {

      this.dan = dan;
      this.mesec = mesec;
      this.godina = godina;

   }

   public Datum(final Datum d)
   {
      this(d.dan, d.mesec, d.godina);
   }

   // metod vraca 1 ako je godina prestupna, a 0 inace
   public static int prestupna(int godina)
   {
      if (godina % 400 == 0 || godina % 4 == 0 && godina % 100 != 0)
         return 1;
      return 0;
   }

   public static int daniUmesecu(int mesec, int godina)
   {
      switch (mesec)
      {
         case 1:
         case 3:
         case 5:
         case 7:
         case 8:
         case 10:
         case 12:
            return 31;
         case 4:
         case 6:
         case 9:
         case 11:
            return 30;
         case 2:
            return 28 + prestupna(godina);
         default:
            return 0;
      }
   }

   public static boolean validan(int g, int d, int m)
   {
      return g > 1900 && (m >= 1 && m <= 12)
          && (d >= 1 && d <= daniUmesecu(m, g));
   }

   /*
    * ako u String-u nije dobar unos,
    * metod ispisuje poruku o gresci i vraca null
    */
   public static Datum string2datum(String datumS)
   {
```

```
      if (datumS.length() != 11)
      {
         System.out.println("Pogresna duzina unosa");
         return null;
      }

      if (datumS.charAt(2) != '.' || datumS.charAt(5) != '.'
          || datumS.charAt(10) != '.')
      {
         System.out.println("Neispravan format unosa");
         return null;
      }

      int dan = 0;
      int mesec = 0;
      int godina = 0;

      try
      {
         dan = Integer.parseInt(datumS.substring(0, 2));
         mesec = Integer.parseInt(datumS.substring(3, 5));
         godina = Integer.parseInt(datumS.substring(6, 10));
      } catch (NumberFormatException e)
      {
         System.out.println("Neispravan format unosa");
         return null;
      }

      if (Datum.validan(godina, dan, mesec))
         return new Datum(dan, mesec, godina);
      else
      {
         System.out.println("Neispravna vrednost dana, meseca ili godine");
         return null;
      }

   }

   public int getDan()
   {
      return dan;
   }

   public int getMesec()
   {
      return mesec;
   }

   public int getGodina()
   {
      return godina;
   }

   public String toString()
   {
      String danS = (dan < 10) ? "0" + dan : "" + dan;
      String mesecS = (mesec < 10) ? "0" + mesec : "" + mesec;
      return danS + "." + mesecS + "." + godina + ".";
   }

}
```

Osoba.java

```
package numerologija;

public abstract class Osoba
```

```
{

   private String ime;
   private String prezime;
   private String adresa;

   public Osoba(String ime, String prezime, String adresa)
   {
      this.ime = ime;
      this.prezime = prezime;
      this.adresa = adresa;
   }

   // metodi koji ce se polimorfno pozivati
   public abstract int numeroloskiBroj();

   public abstract String metabolizam(final Datum d);

   int zbirCifara(int broj)
   {
      int suma = 0;
      do
      {
         suma += broj % 10;
         broj /= 10;
      } while (broj != 0);
      return suma;
   }

   public String getIme()
   {
      return ime;
   }

   public String getPrezime()
   {
      return prezime;
   }

   public String getAdresa()
   {
      return adresa;
   }

   public String toString()
   {
      return prezime + " " + ime + ", " + adresa;
   }

}
```

OsobaD.java

```
package numerologija;

public class OsobaD extends Osoba
{

   private Datum datum;

   public OsobaD(String ime, String prezime, String adresa, Datum d)
   {
      super(ime, prezime, adresa);
      datum = new Datum(d);
   }

   public OsobaD(final OsobaD o)
```

```
   {
      super(o.getIme(), o.getPrezime(), o.getAdresa());
      datum = new Datum(o.datum);
   }

   public int numeroloskiBroj()
   {
      int suma = zbirCifara(datum.getDan()) + zbirCifara(datum.getMesec())
          + zbirCifara(datum.getGodina());

      while (suma > 9)
         suma = zbirCifara(suma);

      return suma;
   }

   public String metabolizam(final Datum d)
   {

      int zbir = (datum.getDan() + d.getDan()) * 1000000 +
                 (datum.getMesec() + d.getMesec()) * 10000 +
                 datum.getGodina() + d.getGodina();

      String rez = String.valueOf(zbir);

      if (rez.length() == 7)
         rez = "0" + rez;

      return rez;
   }

   public String toString()
   {
      return super.toString() + "; numeroloski broj: " + numeroloskiBroj();
   }

}
```

OsobaJMBG.java

```
package numerologija;

public class OsobaJMBG extends Osoba
{
   private String JMBG;

   public OsobaJMBG(String ime, String prezime, String adresa, String JMBG)
   {
      super(ime, prezime, adresa);
      this.JMBG = JMBG;
   }

   public OsobaJMBG(final OsobaJMBG o)
   {
      this(o.getIme(), o.getPrezime(), o.getAdresa(), o.JMBG);
   }

   public static boolean validan(String JMBG)
   {
      if (JMBG.length() != 13)
         return false;
      for (int i = 0; i < JMBG.length(); i++)
         if (!Character.isDigit(JMBG.charAt(i)))
            return false;

      // Izdvajamo dan, mesec i godinu da bismo videli da li su validni
      int dan = Integer.parseInt(JMBG.substring(0, 2));
```

```
      int mesec = Integer.parseInt(JMBG.substring(2, 4));
      int godina = 1000 + Integer.parseInt(JMBG.substring(4, 7));

      if (!Datum.validan(godina, dan, mesec))
         return false;

      return true;
   }

   public int numeroloskiBroj()
   {
      int suma = 0;
      for (int i = 0; i < 7; i++)
      {
         suma += JMBG.charAt(i) - '0';
      }
      suma += 1; /* za ono izostavljeno 1 */

      while (suma > 9)
         suma = zbirCifara(suma);

      return suma;
   }

   public String metabolizam(final Datum d)
   {
      String datumS = JMBG.substring(0, 4) + "1" + JMBG.substring(4, 7);
      int dI = d.getDan() * 1000000 + d.getMesec() * 10000 + d.getGodina();

      String rez = String.valueOf(Integer.parseInt(datumS) + dI);
      if (rez.length() == 7)
         rez = "0" + rez;

      return rez;
   }

   public String toString()
   {
      return super.toString() + "; numeroloski broj: " + numeroloskiBroj();
   }

}
```

TestOsoba.java

```
package numerologija;

import java.util.Scanner;

public class TestOsoba
{

   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);

      System.out.println("Unesite broj osoba: ");
      int n = sc.nextInt();

      Osoba osobe[] = new Osoba[n];
      for (int i = 0; i < n; i++)
      {
         System.out.println("Unesite ime " + (i + 1) + ". osobe:");
         String ime = sc.next();
         System.out.println("Unesite prezime osobe: ");
         String prezime = sc.next();
         System.out.println("Unesite adresu osobe: ");
```

```
        String adresa = sc.next();

        System.out.println("Ako znate datum rodjenja osobe,"
            + "unesite 1,\n a ako znate JMBG unesite 2");
        int izbor = sc.nextInt();
        switch (izbor)
        {
           case 1:
              System.out.println("Unesite datum u obliku dd.mm.gggg.");
              String datumS = sc.next();

              Datum datum = Datum.string2datum(datumS);
              if (datum == null)
              {
                 i--;
                 break;
              }

              osobe[i] = new OsobaD(ime, prezime, adresa, datum);
              break;

           case 2:
              System.out.println("Unesite JMBG:");
              String JMBG = sc.next();
              if (OsobaJMBG.validan(JMBG))
              {
                 osobe[i] = new OsobaJMBG(ime, prezime, adresa, JMBG);
                 break;
              }

              System.out.println("Nije validan JMBG");
              i--;
              break;

           default:
              System.out.println("Pogresna opcija");
              i--;
              break;
        }

     }

     Datum datumDanasnji = null;
     while (datumDanasnji == null)
     {
        System.out.println("Unesite danasnji datum u formatu dd.mm.gggg.");
        String datumDanasnjiS = sc.next();

        datumDanasnji = Datum.string2datum(datumDanasnjiS);
     }

     System.out.println("Rezultati: ");
     for (int i = 0; i < n; i++)
        if (osobe[i].numeroloskiBroj() != 7
            && osobe[i].metabolizam(datumDanasnji).indexOf("0") == -1)
           System.out.println(osobe[i] + ", metabolizam: "
               + osobe[i].metabolizam(datumDanasnji));

  }

}
```

11. колоквијум 2009, Пријављивање испита. Написати програм који обрађује пријављивање испита у текућем испитном року за студенте који студирају по старом статуту.
Класа *Student* описује пријаву једног испита од стране једног студента. Инстанцне промењљиве класе су: категорија (може бити: "редовни", "самофинансирајући", "апсолвент"), име, презиме и смер

студента (све типа *String*), датум пријаве (типа *Datum*) и подаци о испиту који студент пријављује (типа *Ispit*).
За редовног студента се додатно зна његова година студија, као и који пут по реду пријављује испит.
За самофинансирајућег студента се додатно зна његова година студија.
Познато је да у роковима "јануар", "фебруар", "јун", "септембар" и "октобар" сви студенти могу полагати испите, а да у роковима "март", "април", "новембар" и "децембар" испите могу полагати само апсолвенти. За сваки од испитних рокова зна се крајњи рок до ког је могуће пријавити испите (нека то буде до 1-ог тог месеца).
Класа *Datum* од инстанцних променљивих има: дан, месец и годину (све типа *int*).
Класа *Ispit* од инстанцних променљивих има: име испита (типа *String*) и годину на којој се тај испит слуша (типа *int*).
У свакој од наведених класа имплементирати:

- конструктор за креирање објеката када су доступни сви неопходни подаци
- копи-конструктор
- неопходне *get*()* и *set*()* методе
- метод који враћа *String*-репрезентацију објекта класе (Датум исписати у формату дд.мм.гггг.)

Обезбедити да се полиморфно извршавају методи:

```
1.) int uplata();
```

који враћа суму новца коју је студент уплатио приликом пријаве испита.
Редовни студент не плаћа ништа ако испит пријављује мање од 3 пута, а плаћа 300 динара за испит који пријављује 3 или више пута. Самофинансирајући студенти и апсолвенти плаћају по 300 динара за сваки пријављени испит.

```
2.) boolean moze(String rok);
```

који треба да врати да ли студент може пријавити жељени испит у датом року.
Студент може пријавити само испите које је одслушао и то до крајњег рока за испитни рок у коме има право да полаже испите. (Апсолвенти су одслушали све испите.)
Допуштено је у класе додавати и друге методе за којима се укаже потреба при решавању задатка.
Написати тест-класу која ради следеће:

- захтева да корисник са стандардног улаза унесе број пријава испита, а затим:
- у петљи очекује да корисник унесе име, презиме, смер студента, датум пријаве испита (у облику дд.мм.гггг.), податке о испиту који се пријављује (име испита и година на којој се испит слуша)
- након тога "редовни" ако је студент редовни, "самофинансирајући" ако је самофинансирајући, а "апсолвент" ако је апсолвент.
- ако је унето "редовни", уноси се година студија студента и који пут он пријављује испит, па се креира одговарајући објекат, ако је унето "самофинансирајући", уноси се година студија студента и креира одговарајући објекат, а ако је унето "апсолвент", креира се објекат одговарајућег типа.
- све креиране објекте потребно је сместити у један низ одговарајућег типа
- по завршетку петље уноси се испитни рок за који је пријава у току
- главни програм затим пролази кроз све пријаве и исписује за сваку успешну пријаву податке о студенту и испиту као и уплаћену суму новца, а за сваку неуспешну пријаву податке о студенту и испиту као и текст "ПРИЈАВА НИЈЕ ПРОШЛА".
- неопходно је по уношењу сваког датума проверити да ли је тај унос валидан (нпр. 29.02.2009. није валидан датум (2009 није преступна) или 23.2.2009. (није облика дд.мм.гггг.) или 3.02.2009.(није облика дд.мм.гггг.) или 32.10.2008.(10. месец има 31 дан) као ни 03.02.2009 (нема тачку на крају) итд. Година је преступна ако је дељива са 400 или је дељива са 4, а није дељива са 100.

Објашњење:

Класа *Datum* се од истоимене класе из решења задатка ??? разликује само по томе што има још и метод *pre()* који испитује да ли је текући датум пре задатог.

Помоћна класа *Ispit* има једноставну дефиницију без специфичних детаља које би требало посебно објашњавати.

Базна класа *Student* декларисана је као апстрактна, а апстрактни су методи *uplata()* и *moze()* које желимо да позивамо користећи полиморфизам. У класи је дефинисана и константа *GOD* у којој се чува информација о текућој години, као и два статичка низа: *rokovi* – низ имена испитних рокова у којима сви студенти могу полагати испите и *krajnjiRok* – низ крајњих рокова до којих је могуће пријављивати испите у тим испитним роковима. Битно је истаћи да се име испитног рока налази на истој позицији у низу *rokovi* на којој се у низу *krajnjiRok* налази крајњи рок за пријављивање испита у том року. У вези са овим низовима дефинисана су и два статичка метода: *getRokovi()* – враћа низ имена свих рокова у којима сви студенти могу полагати испите и *getKrajnjiRok()* – враћа елемент низа *krajnjiRok* са позиције која се прослеђује као аргумент метода.

Изведена класа *Apsolvent* се од остале две изведене класе, *Redovni* и *Samofinansirajuci*, разликује по томе што има два статичка низа, попут оних из базне класе *Student*, који се односе на апсолвентске рокове.

Кључни делови изведених класа су полиморфни методи. Метод *uplata()* у класи *Redovni* враћа 300 ако је број пријављивања испита од стране текућег редовног студента мањи од 3, а иначе враћа 0. У класама *Samofinansirajuci* и *Apsolvent* метод *uplata()* има истоветну имплементацију – само враћа 300, јер студенти ових категорија плаћају сваки пријављени испит. С друге стране, метод *moze()* има истоветну имплементацију у класама *Redovni* и *Samofinansirajuci*: проверава се да ли студент има право да полаже испите у том року (проласком кроз низ имена допустивих рокова и тражењем у њему имена рока који је у току), па ако се установи да има, даље се проверава да ли је испит уредно пријавио (пре истека крајњег рока за пријаву испита у текућем испитном року) и, на крају, да ли је одслушао тај испит који пријављује. Будући да су апсолвенти одслушали све испите, а имају и додатне, апсолвентске рокове, метод *moze()* класе *Apsolvent* ради следеће: на сличан начин као истоимени методи класа *Redovni* и *Samofinansirajuci* проверава да ли се ради о редовном испитном року и да ли је апсолвент испит пријавио пре истека крајњег рока за пријављивање испита у том испитном року. Ако се не ради о редовном испитном року, испитује се о ком апсолвентском року је реч и да ли је испит пријављен пре крајњег рока за тај апсолвентски рок. И за апсолвентске и за регуларне испитне рокове, крајњи рок за пријављивање испита у испитном року одређиван је на основу имена рока коришћењем чињенице да се име рока у низу *rokovi* налази на истој позицији на којој се крајњи рок за пријаву испита у том испитном року налази у низу *krajnjiRok*.

Тест-класа је имплементирана прилично праволинијски. Дефинише се низ типа базне класе у који се смештају референце на креиране објекте изведених класа. Након тога, уноси се име испитног рока за који је пријава испита у току, а онда се пролази кроз низ, исписују подаци о текућој пријави испита и полиморфно позива метод *moze()* за испитивање да ли је пријава успела. Ако јесте, полиморфно се позива метод *uplata()* како би се видело коју своту новца је студент уплатио, а иначе се исписује порука о томе да пријава испита није прошла.

Решење:

Datum.java

```
package ispiti;

public class Datum
{
   private int dan;
   private int mesec;
   private int godina;

   public Datum(int dan, int mesec, int godina)
   {
```

```
   this.dan = dan;
   this.mesec = mesec;
   this.godina = godina;

}

public Datum(final Datum d)
{
   this(d.dan, d.mesec, d.godina);
}

// metod vraca 1 ako je godina prestupna, a 0 inace
public static int prestupna(int godina)
{
   if (godina % 400 == 0 || godina % 4 == 0 && godina % 100 != 0)
      return 1;
   return 0;
}

public static int daniUmesecu(int mesec, int godina)
{
   switch (mesec)
   {
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12:
         return 31;
      case 4:
      case 6:
      case 9:
      case 11:
         return 30;
      case 2:
         return 28 + prestupna(godina);
      default:
         return 0;
   }
}

public static boolean validan(int g, int d, int m)
{
   return g > 1900 && (m >= 1 && m <= 12)
       && (d >= 1 && d <= daniUmesecu(m, g));
}

/*
 * ako u String-u nije dobar unos,
 * metod ispisuje poruku o gresci i vraca null
 */
public static Datum string2datum(String datumS)
{
   if (datumS.length() != 11)
   {
      System.out.println("Pogresna duzina unosa");
      return null;
   }

   if (datumS.charAt(2) != '.' || datumS.charAt(5) != '.'
       || datumS.charAt(10) != '.')
   {
      System.out.println("Neispravan format unosa");
      return null;
```

```
      }

      int dan = 0;
      int mesec = 0;
      int godina = 0;

      try
      {
         dan = Integer.parseInt(datumS.substring(0, 2));
         mesec = Integer.parseInt(datumS.substring(3, 5));
         godina = Integer.parseInt(datumS.substring(6, 10));
      } catch (NumberFormatException e)
      {
         System.out.println("Neispravan format unosa");
         return null;
      }

      if (Datum.validan(godina, dan, mesec))
         return new Datum(dan, mesec, godina);
      else
      {
         System.out.println("Neispravna vrednost dana, meseca ili godine");
         return null;
      }

   }

   // da li je tekuci datum pre zadatog
   public boolean pre(final Datum d)
   {
      return godina < d.godina
          || (godina == d.godina &&
             (mesec < d.mesec || (mesec == d.mesec && dan < d.dan)));
   }

   public int getDan()
   {
      return dan;
   }

   public int getMesec()
   {
      return mesec;
   }

   public int getGodina()
   {
      return godina;
   }

   public String toString()
   {
      String danS = (dan < 10) ? "0" + dan : "" + dan;
      String mesecS = (mesec < 10) ? "0" + mesec : "" + mesec;
      return danS + "." + mesecS + "." + godina + ".";
   }

}
```

Ispit.java

```
package ispiti;

public class Ispit
{

   private String ispit;
```

```
   private int godina;

   public Ispit(String ispit, int godina)
   {
      this.ispit = ispit;
      this.godina = godina;
   }

   public Ispit(final Ispit i)
   {
      this(i.ispit, i.godina);
   }

   public String getIspit()
   {
      return ispit;
   }

   public int getGodina()
   {
      return godina;
   }

   public String toString()
   {
      return ispit + ", god. studija: " + godina;
   }

}
```

Student.java

```
package ispiti;

public abstract class Student
{
   public static final int GOD = 2009;
   private static String rokovi[] = { "januar", "februar", "jun",
                                      "septembar", "oktobar" };
   private static Datum krajnjiRok[] = { new Datum(1, 1, GOD),
       new Datum(1, 2, GOD), new Datum(1, 6, GOD),
       new Datum(1, 9, GOD), new Datum(1, 10, GOD) };

   private String ime;
   private String prezime;
   private String smer;

   private Datum datum;
   private Ispit ispit;

   private String kategorija;

   public Student(String kategorija, String ime, String prezime,
String smer, final Datum datum, final Ispit ispit)
   {
      this.kategorija = kategorija;
      this.ime = ime;
      this.prezime = prezime;
      this.smer = smer;
      this.datum = new Datum(datum);
      this.ispit = new Ispit(ispit);
   }

   public Student(final Student s)
   {
      this(s.kategorija, s.ime, s.prezime, s.smer, s.datum, s.ispit);
   }
```

```
   public abstract int uplata();

   public abstract boolean moze(String rok);

   public String getIme()
   {
      return ime;
   }

   public String getPrezime()
   {
      return prezime;
   }

   public String getSmer()
   {
      return smer;
   }

   public Datum getDatum()
   {
      return datum;
   }

   public Ispit getIspit()
   {
      return ispit;
   }

   public static String[] getRokovi()
   {
      return rokovi;
   }

   public static Datum getKrajnjiRok(int i)
   {
      return krajnjiRok[i];
   }

   public String toString()
   {
      return kategorija + " student: " + ime + " " + prezime + ", smer: "
+ smer + ", datum prijave: " + datum + ", ispit: " + ispit;
   }
}
```

Redovni.java

```
package ispiti;

public class Redovni extends Student
{

   private int godinaStudija;
   private int kojiPut;

   public Redovni(String ime, String prezime, String smer,
int godinaStudija, final Datum d, final Ispit i, int kojiPut)
   {
      super("redovni", ime, prezime, smer, d, i);
      this.godinaStudija = godinaStudija;
      this.kojiPut = kojiPut;
   }

   public Redovni(final Redovni r)
   {
```

```
      super("redovni", r.getIme(), r.getPrezime(),
             r.getSmer(), r.getDatum(), r.getIspit());
      godinaStudija = r.godinaStudija;
      kojiPut = r.kojiPut;
   }

   public int uplata()
   {
      if (kojiPut < 3)
        return 0;
      return 300;
   }

   public boolean moze(String rok)
   {
      String rokovi[] = Student.getRokovi();

      for (int i = 0; i < rokovi.length; i++)
        if (rok.equals(rokovi[i]) && getDatum().pre(getKrajnjiRok(i))
            && godinaStudija > getIspit().getGodina())
           return true;

      return false;

   }

   public String toString()
   {
      return super.toString() + ", ispit polaze: " + kojiPut + ". put"
          + "\ngodina studija studenta: " + godinaStudija;
   }
}
```

Samofinansirajuci.java

```
package ispiti;

public class Samofinansirajuci extends Student
{

   private int godinaStudija;

   public Samofinansirajuci(String ime, String prezime, String smer,
       int godinaStudija, final Datum d, final Ispit i)
   {
      super("samofinansirajuci", ime, prezime, smer, d, i);
      this.godinaStudija = godinaStudija;
   }

   public Samofinansirajuci(final Samofinansirajuci s)
   {
      super("samofinansirajuci", s.getIme(), s.getPrezime(),
             s.getSmer(), s.getDatum(), s.getIspit());
      godinaStudija = s.godinaStudija;
   }

   public boolean moze(String rok)
   {
      String rokovi[] = Student.getRokovi();

      for (int i = 0; i < rokovi.length; i++)
        if (rok.equals(rokovi[i]) && getDatum().pre(getKrajnjiRok(i))
            && godinaStudija > getIspit().getGodina())
           return true;

      return false;
```

```
   }

   public int uplata()
   {
      return 300;
   }

   public String toString()
   {
      return super.toString() + "\ngodina studija studenta: " + godinaStudija;
   }

}
```

Apsolvent.java

```
package ispiti;

public class Apsolvent extends Student
{
   private static String apsrokovi[] = { "mart", "april",
"novembar", "decembar" };
   private static Datum krajnjiRokAps[] = {
new Datum(1, 3, GOD), new Datum(1, 4, GOD),
new Datum(1, 11, GOD), new Datum(1, 12, GOD) };

   public Apsolvent(String ime, String prezime, String smer, final Datum d,
       final Ispit i)
   {
      super("apsolvent", ime, prezime, smer, d, i);
   }

   public Apsolvent(final Apsolvent a)
   {
      super("apsolvent", a.getIme(), a.getPrezime(), a.getSmer(),
           a.getDatum(), a.getIspit());
   }

   public int uplata()
   {
      return 300;
   }

   public boolean moze(String rok)
   {

      /*
       * ako je student prijavio ispit za apsolventski rok
       * pre isteka za njegovo prijavljivanje
       */
      int i;
      for (i = 0; i < apsrokovi.length; i++)
         if (rok.equals(apsrokovi[i]) && getDatum().pre(krajnjiRokAps[i]))
            return true;

      // slucaj neapsolventskog roka
      String rokovi[] = getRokovi();
      for (i = 0; i < rokovi.length; i++)
         if (rok.equals(rokovi[i]) && getDatum().pre(getKrajnjiRok(i)))
            return true;

      return false;

   }

}
```

Test.java

```
package ispiti;

import java.util.Scanner;

public class Test
{

   public static void main(String[] args)
   {

      Scanner sc = new Scanner(System.in);

      System.out.println("Unesite broj studentskih prijava: ");
      int n = sc.nextInt();

      Student studenti[] = new Student[n];

      for (int i = 0; i < n; i++)
      {
         System.out.println("Ime " + (i + 1) + ". studenta: ");
         String ime = sc.next();
         System.out.println("Prezime: ");
         String prezime = sc.next();
         System.out.println("Smer: ");
         String smer = sc.next();

         Datum datum = null;
         while (datum == null)
         {
// ponavlja se dok se ne unese ispravan datum
            System.out.println("Datum prijave, oblika dd.mm.gggg.");
            String datumS = sc.next();

            datum = Datum.string2datum(datumS);
         }

         System.out.println("Unesite ime ispita: ");
         String ispitS = sc.next();

         System.out.println("Unesite godinu studija na kojoj se slusa ispit: ");
         int godinaStudija = sc.nextInt();

         Ispit ispit = new Ispit(ispitS, godinaStudija);

         System.out.println("Unesite: \n");
         System.out.println("\t\"redovni\" - za redovnog studenta");
         System.out.println(
"\t\"samofinansirajuci\" - za samofinansirajuceg i ");
         System.out.println("\t\"apsolvent\" - za apsolventa");

         String kategorija = sc.next();

         if (kategorija.equalsIgnoreCase("redovni"))
         {
            System.out.println("Unesite godinu studija studenta: ");
            int godStud = sc.nextInt();
            System.out.println(
"Unesite koji put polaze ispit koji prijavljuje: ");
            int kojiPut = sc.nextInt();
            studenti[i] = new Redovni(ime, prezime, smer, godStud,
 datum, ispit, kojiPut);
         } else if (kategorija.equalsIgnoreCase("samofinansirajuci"))
         {
            System.out.println("Unesite godinu studija studenta: ");
            int godStud = sc.nextInt();
```

```
          studenti[i] = new Samofinansirajuci(ime, prezime,
smer, godStud, datum, ispit);
        } else if (kategorija.equalsIgnoreCase("apsolvent"))
        {
          studenti[i] = new Apsolvent(ime, prezime, smer, datum, ispit);
        } else
        {
          System.out.println("Pogresan izbor kategorije");
          i--;
        }
     }

     System.out.println("Unesite ispitni rok za koji je prijava u toku: ");
     String tekuciRok = sc.next();

     for (int i = 0; i < studenti.length; i++)
        if (studenti[i].moze(tekuciRok))
          System.out.println(studenti[i] + ", uplatiti: " +
 studenti[i].uplata() + " dinara.");
        else
          System.out.println(studenti[i] + ", PRIJAVA NIJE PROSLA.");

   }

}
```

12. колоквијум 2009, MTS. Написати програм за рад са *prepaid* и *postpaid* корисницима једног провајдера мобилне телефоније.

Класа *Korisnik* описује једног корисника мобилне телефоније. Инстанцна промењљива класе је телефонски број корисника (типа *String*). Бројеви свих корисника провајдера почињу са 064, 065 или 066 и дужине су 9 или 10 цифара. На све услуге, осим допуне кредита *prepaid* корисника, зарачунава се ПДВ од 18%.

За *Prepaid* корисника зна се још и његов кредит (типа *double*). Сваки *prepaid* корисник има 3 *friends&family* броја (који морају припадати истој мрежи). Цена разговора је 7.5 дин/мин, осим са *friends&family* бројевима за које важи цена од 4.5 дин/мин. Успостављање везе кошта 2.8 дин. Цена једног SMS-а је 0.9 дин. Све цене су приказане без урачунатог ПДВ-а. Први минут разговора се заокружује, а након тога позив се тарифира на 30 секунди (нпр. ако је разговор трајао 2 минута и 15 секунди, заокружује се на 2 минута, а ако је трајао 3 минута и 52 секунде, заокружује се на 4 минута).

Класа *Prepaid* треба да има и метод:

```
void dopuni kredit(int dopuna);
```

за допуну кредита износом допуна.

За *Postpaid* корисника зна се још и колики му је тренутно рачун (тип *double*). *Postpaid* корисници плаћају месечну претплату од 150 динара (иницијална вредност рачуна при креирању новог *postpaid* корисника). Минут разговора у мрежи кошта 2.8 дин, а минут разговора у националном саобраћају стаје 5.8 дин. Успостава везе кошта 2.5 дин. Обрачунски интервал за разговор је 1 секунда (нема заокруживања на целе минуте). Цена једног SMS-а је 2.2 дин. Све цене су приказане без урачунатог ПДВ-а.

Класа *Vreme* од инстанцних променљивих: има сат, минут и секунду (све типа *int*).

Класа *Razgovor* од инстанцних променљивих има: број (број са којим се разговара, типа *String*) и време (трајање разговора, типа *Vreme*).

У свакој од наведених класа имплементирати:

- конструктор за креирање објеката када су доступни сви неопходни подаци. (Приликом креирања *prepaid* корисника задају се и његови *friends&family* бројеви.)
- копи-конструктор
- неопходне *get*()* и *set*()* методе
- метод који враћа *String*-репрезентацију објекта класе (Време исписати у формату сат:минут:секунда. У *String*-репрезентацији *prepaid* корисника треба да се нађу и његови *friends&family* бројеви.)

Обезбедити да се полиморфно извршавају методи:

1.) `void azuriraj_racun_razgovor(Razgovor razgovor);`
који ажурира вредност кредита *prepaid* корисника, односно вредност рачуна *postpaid* корисника након обављеног разговора описаног параметром метода. Уколико *prepaid* корисник нема довољно кредита, кредит поставити на 0.
2.) `void azuriraj_racun_SMS();`
који ажурира вредност кредита *prepaid* корисника, односно вредност рачуна *postpaid* корисника након слања SMS-а. Уколико *prepaid* корисник нема довољно кредита, кредит поставити на 0.
Допуштено је у класе додавати и друге методе за којима се укаже потреба при решавању задатка.
Написати тест-класу која ради следеће:

- захтева да корисник са стандардног улаза унесе број корисника, а затим:
- у петљи очекује унос телефонског броја корисника, а након тога 1 за *prepaid*, а 2 за *postpaid* корисника.
- ако је унето 1, потом се уноси и кредит корисника и његови *friends&family* бројеви.
- креира се објекат одговарајућег типа.
- све креиране објекте потребно је смештати у један низ одговарајућег типа
- по завршетку петље 5 пута се случајно бира елемент низа и исписују подаци о њему. Ако је у питању *prepaid* корисник, нуде се услуге: "допуна", "разговор" и "SMS", а ако је у питању *postpaid* корисник само "разговор" и "SMS". Ако је потребно, уносе се подаци неопходни за изабрану услугу, извршава се услуга и исписују подаци о кориснику по извршењу услуге.
- неопходно је, по уношењу броја за креирање корисника, проверити да ли је тај број валидан (нпр. 0631234567 није валидан број за креирање корисника (за разговор је валидан!), јер не почиње са 064, 065 или 066; 064123 такође није валидан јер му је дужина 6, као ни 0641a23456 јер у себи има и карактер 'a'). Такође, и за унете податке о времену треба вршити провере исправности: 0 123 34 није исправно (123 минута!) или 2 23 67 (67 секунди!) (Нпр. време се уноси у формату: сати минута секунди).

Објашњење:

У помоћној класи, *Vreme*, осим уобичајених метода (конструктор, копи-конструктор, *get*()* и *set*()* методи, *toString()*) имплементиран је и метод *sledeci_minut(),* који се у класи *Prepaid* користи за заокруживање времена на следећи минут када је то потребно, као и статички метод *validan()* који за задате вредности броја сати, минута и секунди проверава да ли су то валидни подаци за креирање објекта типа *Vreme*.

Помоћна класа *Razgovor* има једноставну дефиницију без специфичних детаља које би требало посебно објашњавати.

Базна класа *Korisnik* декларисана је као апстрактна, а апстрактни су методи *azuriraj_racun_razgovor()* и *azuriraj_racun_SMS()* које желимо да позивамо користећи полиморфизам. И у овој класи имамо статички метод *validan()* који проверава да ли је у задатом *String*-у записан валидан телефонски број корисника, тј. да ли се број састоји искључиво од цифара, почиње са 064, 065 или 066 и дужине је 9 или 10. За ове провере користе се одговарајући методи класа *String* и *Character*.

У изведеним класама, *Prepaid* и *Postpaid*, дефинисане су константе које представљају цене услуга, без урачунатог ПДВ-а.

Од инстанцних променљивих, класа *Prepaid* има кредит корисника и низ његових *friends&family* бројева. Допуна кредита врши се додавањем износа допуне на текућу вредност кредита. У методу *azuriraj_racun_razgovor()* цена разговора по минуту се иницијално поставља на подразумевану, а касније се, ако се проласком кроз низ *friends&family* бројева установи да је разговор са неким од њих, цена ажурира вредношћу која важи за такве разговоре. Испитује се дужина трајања разговора и, по потреби, врши заокруживање на најближи претходни или следећи цео минут. Потом се израчунава цена разговора као сума цене успоставе везе и производа дужине разговора у минутима и цене

разговора по минуту, уз зарачунавање пореза. Уколико корисник има довољно кредита, он се умањује за израчунату цену разговора, а иначе, кредит се поставља на 0. Метод *azuriraj_racun_SMS()* умањује вредност кредита за цену *SMS*-а уз зарачунати порез. Уколико корисник нема довољно кредита за ову услугу, тј. претходним одузимањем је добијена негативна вредност, кредит се поставља на 0.

Метод *azuriraj_racun_razgovor()* класе *Postpaid* иницијално поставља цену разговора по минуту на цену која важи за националне разговоре, а уколико се ради о разговору са корисником исте мреже, цена се ажурира одговарајућом вредношћу. Цена разговора се рачуна као збир цене успоставе везе и производа дужине разговора у секундама и цене разговора по секунди (која се добија дељењем цене по минуту са 60), уз зарачунавање пореза. Рачун корисника ажурира се додавањем добијене цифре на текућу вредност рачуна. Метод *azuriraj_racun_SMS()* увећава вредност рачуна за цену *SMS*-а уз зарачунати порез.

У тест-класи дефинише се низ типа базне класе у који се смештају референце на креиране објекте изведених класа. Након што се случајно изабере неки од елемената низа, односно креираних корисника, применом оператора *instanceof* утврђује се његов стварни тип и у зависности од тога нуде расположиве услуге (допуна, разговор и SMS за *prepaid* корисника, а разговор и SMS за *postpaid* корисника). Методи који се тичу разговора и SMS–а позивају се полиморфно коришћењем промењиве типа базне класе, док је метод за допуну специфичан за класу *Prepaid* па је за његово позивање најпре неопходно извшити кастовање референце у стварни тип објекта (*Prepaid*).

Решење:

Vreme.java

```
package mts;

public class Vreme
{
   private int sat;
   private int minut;
   private int sekunda;

   public Vreme(int sat, int minut, int sekunda)
   {
      this.sat = sat;
      this.minut = minut;
      this.sekunda = sekunda;
   }

   public Vreme(final Vreme v)
   {
      this(v.sat, v.minut, v.sekunda);
   }

   public int getSat()
   {
      return sat;
   }

   public int getMinut()
   {
      return minut;
   }

   public int getSekunda()
   {
      return sekunda;
   }
```

```
   public Vreme sledeci_minut()
   {
      if (minut + 1 < 59)
         return new Vreme(sat, minut + 1, 0);
      else
         return new Vreme(sat + 1, 0, 0);
   }

   public static boolean validan(int sat, int minut, int sekunda)
   {
      return sat >= 0 && minut >= 0 && minut <= 59 && sekunda >= 0
          && sekunda <= 59;
   }

   public String toString()
   {
      return sat + ":" + minut + ":" + sekunda;
   }
}
```

Razgovor.java

```
package mts;

public class Razgovor
{
   private String broj;
   private Vreme vreme;

   public Razgovor(String broj, Vreme vreme)
   {
      this.broj = broj;
      this.vreme = new Vreme(vreme);
   }

   public Razgovor(final Razgovor r)
   {
      this(r.broj, r.vreme);
   }

   public String getBroj()
   {
      return broj;
   }

   public Vreme getVreme()
   {
      return vreme;
   }

   public String toString()
   {
      return "Razgovor sa: " + broj + ", trajao je: " + vreme;
   }

}
```

Korisnik.java

```
package mts;

public abstract class Korisnik
{
   private String broj;
   static final int PDV = 18;

   public Korisnik(String broj)
```

```
   {
      this.broj = broj;
   }

   public Korisnik(final Korisnik k)
   {
      this(k.broj);
   }

   public String getBroj()
   {
      return broj;
   }

   public abstract void azuriraj_racun_razgovor(Razgovor razgovor);

   public abstract void azuriraj_racun_SMS();

   public static boolean validan(String broj)
   {
      // Ako nisu sve cifre, broj nije validan
      for (int i = 0; i < broj.length(); i++)
         if (!Character.isDigit(broj.charAt(i)))
            return false;
      // treba da pocinje sa 064, 065 ili 066 i da je duzine 9 ili 10 (cifara)
      return (broj.startsWith("064") || broj.startsWith("065") ||
broj.startsWith("066"))
&& (broj.length() == 9 || broj.length() == 10);
   }

   public String toString()
   {
      return "Korisnik: " + broj;
   }
}
```

Prepaid.java

```
package mts;

public class Prepaid extends Korisnik
{

   private static final double RAZGOVOR = 7.5;
   private static final double FF = 4.5;
   private static final double USPOSTAVA = 2.8;
   private static final double SMS = 0.9;

   private double kredit;
   private String[] ff_brojevi = new String[3];

   public Prepaid(String broj, double kredit, String[] ff)
   {
      super(broj);
      this.kredit = kredit;
      for (int i = 0; i < ff_brojevi.length; i++)
         ff_brojevi[i] = ff[i];
   }

   public Prepaid(final Prepaid p)
   {
      super(p.getBroj());
      kredit = p.kredit;
      for (int i = 0; i < ff_brojevi.length; i++)
         ff_brojevi[i] = p.ff_brojevi[i];
   }
```

```
   public void dopuni_kredit(int dopuna)
   {
      kredit += dopuna;
   }

   /*
    * prvi minut razgovora se zaokruzuje, a nakon toga poziv se tarifira na 30
    * sekundi
    */
   public void azuriraj_racun_razgovor(Razgovor r)
   {
      // treba razlikovati f&f brojeve od ostalih
      double cena = RAZGOVOR;
      Vreme vreme = null;

      if (r.getVreme().getSat() == 0 && r.getVreme().getMinut() == 0)
         vreme = new Vreme(0, 1, 0);
      else if (r.getVreme().getSekunda() > 30)
         vreme = r.getVreme().sledeci_minut();
      else
         vreme = new Vreme(r.getVreme().getSat(), r.getVreme().getMinut(), 0);

      for (int i = 0; i < ff_brojevi.length; i++)
         if (r.getBroj().equals(ff_brojevi[i]))
         {
            cena = FF;
            break;
         }

      double cena_razgovora = (USPOSTAVA +
(vreme.getSat() * 60 + vreme.getMinut())* cena)
            * (100 + PDV) / 100.0;
      if (kredit < cena_razgovora)
         kredit = 0;
      else
         kredit -= cena_razgovora;

   }

   public void azuriraj_racun_SMS()
   {
      kredit -= SMS * (100 + PDV) / 100.0;
      if (kredit < 0)
         kredit = 0;
   }

   public String toString()
   {
      String ffString = "";
      for (int i = 0; i < ff_brojevi.length; i++)
         if (ff_brojevi[i] != null)
            ffString += ff_brojevi[i] + " ";
      return super.toString() + " kredit: " + kredit + "\n ff-brojevi: "
          + ffString;
   }
}
```

Postpaid.java

```
package mts;

public class Postpaid extends Korisnik
{

   private static final double PRETPLATA = 150;
   private static final double RAZGOVOR_U_MREZI = 2.8;
   private static final double RAZGOVOR_NACIONALNI = 5.8;
```

```
   private static final double USPOSTAVA = 2.5;
   private static final double SMS = 2.2;

   private double racun;

   public Postpaid(String broj)
   {
      super(broj);
      racun = PRETPLATA;
   }

   public Postpaid(final Postpaid p)
   {
      super(p.getBroj());
      racun = p.racun;
   }

   public void azuriraj_racun_razgovor(Razgovor r)
   {
      double cena = RAZGOVOR_NACIONALNI;
      if (r.getBroj().startsWith("064") || r.getBroj().startsWith("065")
          || r.getBroj().startsWith("066"))
         cena = RAZGOVOR_U_MREZI;

      double cena_razgovora = (USPOSTAVA + (r.getVreme().getSat() * 3600
          + r.getVreme().getMinut() * 60 + r.getVreme().getSekunda())
          * cena / 60)
          * (100 + PDV) / 100.0;

      racun += cena_razgovora;
   }

   public void azuriraj_racun_SMS()
   {
      racun += SMS * (100 + PDV) / 100.0;
   }

   public String toString()
   {
      return super.toString() + " racun: " + racun;
   }

}
```

TestMTS.java

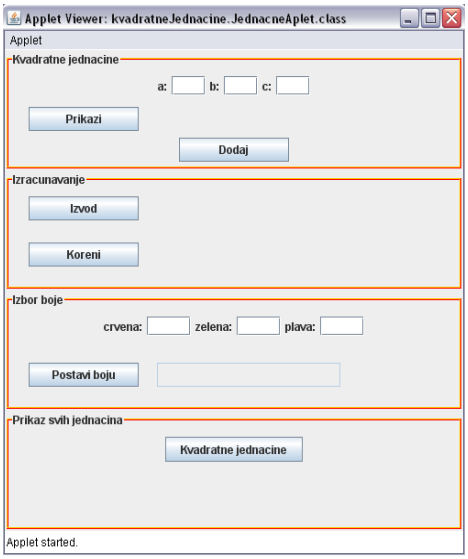
```
package mts;

import java.util.Scanner;
import java.util.Random;

public class TestMTS
{

   public static void main(String[] args)
   {

      System.out.println("Unesite broj korisnika");
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();

      Korisnik korisnici[] = new Korisnik[n];

      for (int i = 0; i < n; i++)
      {
         String broj = null;
         for (;;)
```

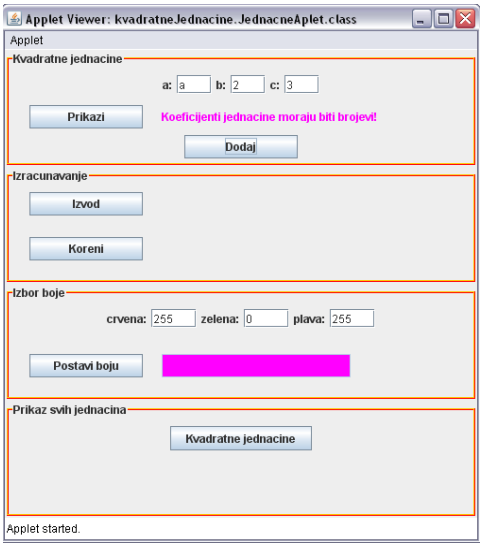
```
        {
           System.out.println("Unesite telefonski broj korisnika: ");
           broj = sc.next();
           if (Korisnik.validan(broj))
              break;
           System.out.println("Nekorektan unos, pokusajte ponovo.");

        }

        System.out.println(
"Unesite 1 za prepaid korisnika, 2 za postpaid korisnika");
        int izbor = sc.nextInt();

        switch (izbor)
        {
           case 1:
              System.out.println("Unesite kredit: ");
              int kredit = sc.nextInt();
              String ff[] = new String[3];
              for (int j = 0; j < ff.length; j++)
              {
                 System.out.println("Unesite " + (j + 1) + ". f&f broj: ");
                 ff[j] = sc.next();
                 if (!Korisnik.validan(ff[j]))
                 {
                   System.out.println(
"Broj nije validan ili ne pripada mrezi korisnika");
                   j--;
                 }
              }
              korisnici[i] = new Prepaid(broj, kredit, ff);
              break;
           case 2:
              korisnici[i] = new Postpaid(broj);
              break;
           default:
              System.out.println("Neispravna opcija.");
              i--;
              break;
        }

     }

     Random random = new Random();
     for (int i = 0; i < 5; i++)
     {
        Korisnik izabrani = korisnici[random.nextInt(korisnici.length)];
        System.out.println(izabrani);

        if (izabrani instanceof Prepaid)
        {
           System.out.println(
"Unesite vrstu usluge: \"dopuna\", \"sms\" ili \"razgovor\"");
           String izbor = sc.next();
           if (izbor.equalsIgnoreCase("dopuna"))
           {
              System.out.println("Unesite sumu:");
              ((Prepaid) izabrani).dopuni_kredit(sc.nextInt());
              System.out.println("Nakon usluge: " + izabrani);

           } else if (izbor.equalsIgnoreCase("sms"))
           {
              izabrani.azuriraj_racun_SMS();
              System.out.println("Nakon usluge: " + izabrani);

           } else if (izbor.equalsIgnoreCase("razgovor"))
           {
```

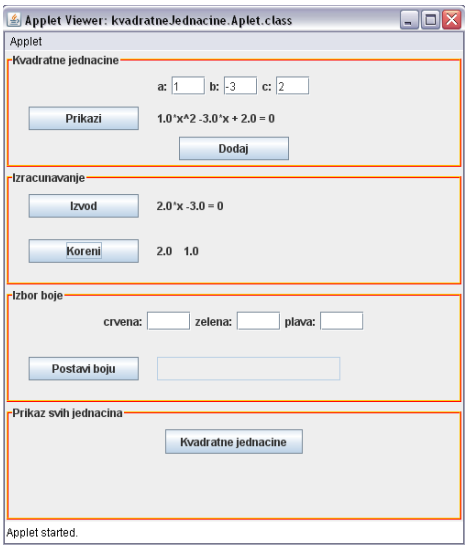
```
            System.out.println("Unesite broj koji se poziva:");
            String broj = sc.next();
            System.out.println(
"Unesite trajanje razgovora: sat minut sekunda");
            int sat = sc.nextInt();
            int minut = sc.nextInt();
            int sekunda = sc.nextInt();
            if (!Vreme.validan(sat, minut, sekunda))
            {
               System.out.println("Neispravno trajanje razgovora");
               i--;
               continue;
            }
            izabrani.azuriraj_racun_razgovor(
new Razgovor(broj, new Vreme(sat, minut, sekunda)));
            System.out.println("Nakon usluge: " + izabrani);
          } else
          {
            System.out.println("Pogresna opcija");
            i--;
            continue;
          }
        } else
        {
          System.out.println("Unesite sms ili razgovor");
          String izbor = sc.next();
          if (izbor.equalsIgnoreCase("sms"))
          {
            izabrani.azuriraj_racun_SMS();
            System.out.println("Nakon usluge: " + izabrani);
          } else if (izbor.equalsIgnoreCase("razgovor"))
          {
            System.out.println("Unesite broj koji se poziva:");
            String broj = sc.next();
            System.out.println(
"Unesite trajanje razgovora: sat minut sekunda");
            int sat = sc.nextInt();
            int minut = sc.nextInt();
            int sekunda = sc.nextInt();
            if (!Vreme.validan(sat, minut, sekunda))
            {
               System.out.println("Neispravno trajanje razgovora");
               i--;
               continue;
            }
            izabrani.azuriraj_racun_razgovor(
new Razgovor(broj, new Vreme(sat,minut, sekunda)));
            System.out.println("Nakon usluge: " + izabrani);

          } else
          {
            System.out.println("Pogresna opcija");
            i--;
          }

        }

     }

   }

}
```

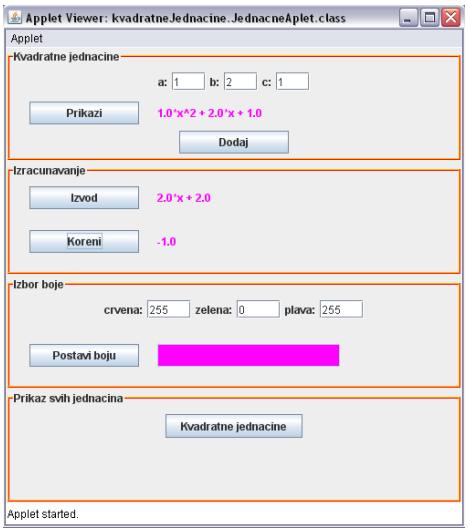
13. *Испит, јун 2009, Квадратне једначине*.
Написати аплет који представља интерфејс за рад са квадратним једначинама.
Аплет треба организовати у целине које садрже:

1. Три текстуална поља за унос коефицијената квадратне једначине. Дугме *Dodaj* за додавање учитане квадратне једначине у колекцију и дугме *Prikazi* за приказ квадратне једначине. Дугме *Prikazi* приказује последњу квадратну једначину смештену у колекцију. Уколико нема једначина у колекцији, ништа се не исписује. Извршити контролу вредности учитаних коефицијената.
2. Дугмета *Izvod* и *Koreni* за рачунање и испис извода одговарајуће квадратне функције и за рачунање и испис корена квадратне једначине. Исписати одговарајућу поруку ако једначина нема реалне корене. Ако једначина има два иста реална корена, исписати само један.
   Формат исписа квадратне једначине: a*x^2 + b*x + c = 0.
   Претпоставити да једначина има вредности различите од нуле за све коефицијенте.
   Обезбедити и коректан испис извода квадратне једначине.
3. Текстуална поља за унос вредности R, G и B компоненте боје која ће се користити за испис квадратне једначине, извода, корена и порука.
   За сваку компоненту се задаје њена вредност која припада интервалу [0, 255] .
   Кликом на дугме *Postavi boju* поставља се боја за испис на одговарајућу комбинацију датих компоненти. Изабраном бојом обојено је текстуално поље поред дугмета *Postavi boju* које не може да се едитује.
   У исто текстуално поље исписује се порука уколико вредност неке од компоненти не припада датом опсегу или је унешена вредност која није број.
4. Дугме *Kvadratne jednacine.* Кликом на дугме врши се испис свих успешно унешених квадратних једначина у фајл *izlaz.txt.*
   Формат исписа: сваки ред фајла садржи једну квадратну једначину иза које су наведени њени реални корени, уколико постоје, у супротном је исписана порука да једначина нема реалних корена.
   Пример: 1.0*x^2 -3.0*x + 2.0 = 0 2.0 1.0
   2.0*x^2 + 3.0*x + 4.0 = 0 Jednacina nema realnih korena!


Applet Viewer: kvadratneJednacine.JednacneAplet.class
Applet
Kvadratne jednacine
a: b: c:
Prikazi
Dodaj
Izracunavanje
Izvod
Koreni
Izbor boje
crvena: zelena: plava:
Postavi boju
Prikaz svih jednacina
Kvadratne jednacine
Applet started.


Applet Viewer: kvadratneJednacine.JednacneAplet.class
Applet
Kvadratne jednacine
a: a b: 2 c: 3
Prikazi
Koeficijenti jednacine moraju biti brojevi!
Dodaj
Izracunavanje
Izvod
Koreni
Izbor boje
crvena: 255 zelena: 0 plava: 255
Postavi boju
Prikaz svih jednacina
Kvadratne jednacine
Applet started.


Applet Viewer: kvadratneJednacine.Aplet.class
Applet
Kvadratne jednacine
a: 1 b: -3 c: 2
Prikazi
1.0*x^2 -3.0*x + 2.0 = 0
Dodaj
Izracunavanje
Izvod
2.0*x -3.0 = 0
Koreni
2.0 1.0
Izbor boje
crvena: zelena: plava:
Postavi boju
Prikaz svih jednacina
Kvadratne jednacine
Applet started.


Applet Viewer: kvadratneJednacine.JednacneAplet.class
Applet
Kvadratne jednacine
a: 1 b: 2 c: 1
Prikazi
1.0*x^2 + 2.0*x + 1.0
Dodaj
Izracunavanje
Izvod
2.0*x + 2.0
Koreni
-1.0
Izbor boje
crvena: 255 zelena: 0 plava: 255
Postavi boju
Prikaz svih jednacina
Kvadratne jednacine
Applet started.

Објашњење:

Класа *KvadratnaJednacina* садржи три инстанцне променљиве типа *double*, које представљају коефицијенте квадратне једначие.
Метод *izvod()* рачуна извод квадратне једначине који је квадратна једначина са коефицијентима *0, 2*a, b*.
Метод *koreni()* рачуна корене квадратне једначине и смешта их у низ. Методом *diskriminanta()* рачуна се дискриминанта квадратне једначине и на основу њене вредности закључује да ли једначина има реалне корене или не. Ако нема реалне корене, поставља се да су оба корена једнака 0. У супротном се рачунају корени по познатој формули.
Пошто се претпоставља да је коефицијент *a* различит од нуле, у методу *toString()* треба дефинисати и коректан испис извода квадратне једначине, када је коефицијент *a* једнак нули. Зато постоје два начина за формирање *String*-репрезентације.

За смештање квадратних једначина које се уносе користи се стек. Да би се приказала задата квадратна једначина, одредили извод и корени квадратне једначине, неопходно је једначину најпре додати у колекцију. Поменуте операције се увек извршавају над једначином која је последња додата колекцији, што указује на потребу за стеком.

Садржај аплета организован је у четири панела.
Панел *panel1* са оквирним насловом *Kvadratne jednacine* је организован у три потпанела употребом *GridLayout(0, 1)* распореда компоненти. Први обухвата поља за уношење коефицијената квадратне једначине. Други обухвата дугме за приказ квадратне јеначине и лабелу у којој се приказ врши, док трећи обухвата дугме за додавање квадратне једначине колекцији.
Панел *panel2* са оквирним насловом *Izracunavanje* је организован у два потпанела употребом *GridLayout(0, 1)* распореда компоненти. Први обухвата дугме *Izvod*, и лабелу за испис извода, а други дугме *Koreni* и лабелу за испис корена.
Панел *panel3* са оквирним насловом *Izbor boje* је организован у два потпанела употребом *GridLayout(0, 1)* распореда компоненти. Први обухвата поља за унос врености *R*, *G*, *B* компоненти боје, а други дугме *Postavi boju* и поље за приказ изабране боје.
Панел *panel4* са оквирним насловом *Prikaz svih jednacina* обухвата дугме *Kvadratne jednacine*.

Обраде догађаја:

- Клик на дугме *Dodaj*. Креира се квадратна једначина на основу вредности коефицијената задатих у текстуалним пољима, уколико су вредности коректне и додаје се на врх стека. Уколико је за вредност неког од коефицијената задата вредност која није број, избацује се изузетак типа *NumberFormatException* и у лабели се исписује одговарајућа порука у боји за испис. Не прекида се извршавање аплета.
- Клик на дугме *Prikazi*. Приказује се последња додата квадратна једначина из колекције, тј. једначина са врха стека у боји за испис. Ако је стек празан, не исписује се ништа.
- Клик на дугме *Izvod*. Рачуна се извод једначине са врха стека и исписује у боји за испис.
- Клик на дугме *Koreni*. Рачунају се корени једначине са врха стека и исписује њихова вредност у боји за испис, односно порука да једначина нема реалних корена.
- Клик на дугме *Postavi boju*. Проверава се да ли су коректно задате вредности компоненти боје. Морају бити бројеви и припадати интервалу [0, 255]. Креира се боја на следећи начин:
  ```
  bojaZaIspis = new Color((R<<16) ^ (G<<8) ^ B);
  ```
  Горње операције битског шифтовања у лево су неопходне, јер у битској репрезентацији вредности боје битови на позицијама 0-7 одговарају компоненти *B*, битови на позицијама 8-15 одговарају компоненти *G*, док битови на позицијама 16-31 одговарају компоненти *R*.
  Други начин је једноставнији:
  ```
  bojaZaIspis = new Color(R, G, B);
  ```
  Задата боја постаје боја за испис.

- Клик на дугме *Kvadratne jednacine*. Покушава се отварање датотеке *„C:\Temp\izlaz.txt“* и у случају неуспеха исписује одговарајућа порука. Ако је све у реду, редом се исписују квадратне једначине из стека и вредности њихових корена.

Решење:

KvadratnaJednacina.java

```
package kvadratneJednacine;

public class KvadratnaJednacina
{
   private double a;
   private double b;
   private double c;

   public KvadratnaJednacina(double a, double b, double c)
   {
      this.a = a;
      this.b = b;
      this.c = c;
   }

   public KvadratnaJednacina izvod()
   {
      return new KvadratnaJednacina(0, 2*a, b);
   }

   public double[] koreni()
   {
      double koreni[] = new double[2];
      double D = diskriminanta();

      if(!imaRealneKorene(D))
         koreni[0] = koreni[1] = 0;
      else if(D == 0)
         koreni[0] = koreni[1] = -b/(2*a);
      else
      {
         koreni[0] = (-b + Math.sqrt(D))/(2*a);
         koreni[1] = (-b - Math.sqrt(D))/(2*a);
      }
      return koreni;
   }

   public double diskriminanta()
   {
      return b*b - 4*a*c;
   }

   public boolean imaRealneKorene(double D)
   {
      return D >= 0 ? true : false;
   }

   public String toString()
   {
      String string = new String();
      if(a == 0)
      {
         string += b + "*x";
         string += c > 0 ? " + " + c : " " + c;
         string += " = 0";
      }
      else
```

```
      {
         string += a + "*x^2";
         string += b > 0 ? " + " + b + "*x" : " " + b + "*x";
         string += c > 0 ? " + " + c : " " + c + " = 0";
         string += " = 0";
      }
      return string;
   }

}
```

Aplet.java

```
package kvadratneJednacine;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Stack;

import javax.swing.BorderFactory;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class Aplet extends JApplet {

   public static final long serialVersionUID = 0;

   private JPanel panel1;
   private JPanel panel2;
   private JPanel panel3;
   private JPanel panel4;

   private JLabel la, lb, lc;
   private JTextField a, b, c;

   private JButton prikazi;
   private JLabel labela_ispis;

   private JButton dodaj;

   private JButton izvod;
   private JLabel labela_izvod;

   private JButton koreni;
   private JLabel labela_koreni;

   private JLabel lcrvena, lzelena, lplava;
   private JTextField crvena, zelena, plava;

   private JButton boja;
   private JTextField polje_boja;

   private JButton prikaziSve;

   private Color bojaZaIspis;
```

```
private Stack<KvadratnaJednacina> jednacine =
                            new Stack<KvadratnaJednacina>();

public void init() {
   SwingUtilities.invokeLater(
      new Runnable() {
         public void run(){
            kreirajGUI();
         }
      });
}

public void kreirajGUI()
{
   setSize(500, 500);
   setLayout(new GridLayout(0, 1));

   panel1 = new JPanel(new GridLayout(0, 1));
   panel1.setBorder(BorderFactory.createTitledBorder(
        BorderFactory.createEtchedBorder(Color.red, Color.orange),
                                            "Kvadratne jednacine"));

   JPanel panel1A = new JPanel();
   la = new JLabel("a:");
   a = new JTextField(3);

   lb = new JLabel("b:");
   b = new JTextField(3);

   lc = new JLabel("c:");
   c = new JTextField(3);

   panel1A.add(la);
   panel1A.add(a);
   panel1A.add(lb);
   panel1A.add(b);
   panel1A.add(lc);
   panel1A.add(c);
   panel1.add(panel1A);

   JPanel panel1B = new JPanel(new FlowLayout(FlowLayout.LEFT, 20, 5));
   prikazi = new JButton("Prikazi");
   prikazi.setPreferredSize(new Dimension(120, 25));
   labela_ispis = new JLabel("");
   panel1B.add(prikazi);
   panel1B.add(labela_ispis);
   panel1.add(panel1B);

   prikazi.addActionListener(
        new ActionListener() {
           public void actionPerformed(ActionEvent e)
           {
              if(jednacine.empty())
                labela_ispis.setText("");
              else
              {
                labela_ispis.setForeground(bojaZaIspis);
                labela_ispis.setText(jednacine.peek().toString());
              }
           }
        });

   JPanel panel1C = new JPanel();
   dodaj = new JButton("Dodaj");
   dodaj.setPreferredSize(new Dimension(120, 25));
   panel1C.add(dodaj);
   panel1.add(panel1C);
```

```
add(panel1);

dodaj.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
           try {
             double koef_a = Double.parseDouble(a.getText());
             double koef_b = Double.parseDouble(b.getText());
             double koef_c = Double.parseDouble(c.getText());

             jednacine.push(new KvadratnaJednacina(
                                 koef_a, koef_b, koef_c));
             labela_ispis.setText("");

           }
           catch(NumberFormatException nfe)
           {
             labela_ispis.setText("" +
                 "Koeficijenti jednacine moraju biti brojevi!");
             labela_ispis.setForeground(bojaZaIspis);
           }
        }
     });

panel2 = new JPanel(new GridLayout(0, 1));
panel2.setBorder(BorderFactory.createTitledBorder(
     BorderFactory.createEtchedBorder(Color.red, Color.orange),
                                        "Izracunavanje"));

JPanel panel2A = new JPanel(new FlowLayout(FlowLayout.LEFT, 20, 5));
izvod = new JButton("Izvod");
izvod.setPreferredSize(new Dimension(120, 25));
labela_izvod = new JLabel("");
panel2A.add(izvod);
panel2A.add(labela_izvod);
panel2.add(panel2A);

izvod.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
           labela_izvod.setText(jednacine.peek().izvod().toString());
           labela_izvod.setForeground(bojaZaIspis);
        }
     });

JPanel panel2B = new JPanel(new FlowLayout(FlowLayout.LEFT, 20, 5));
koreni = new JButton("Koreni");
koreni.setPreferredSize(new Dimension(120, 25));
labela_koreni = new JLabel("");
panel2B.add(koreni);
panel2B.add(labela_koreni);
panel2.add(panel2B);
add(panel2);

koreni.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
           labela_koreni.setForeground(bojaZaIspis);
           double[] resenja = jednacine.peek().koreni();
           if(resenja[0] != resenja[1])
             labela_koreni.setText("" + resenja[0] +
                                   "    " + resenja[1]);
           else if(resenja[0] == 0)
             labela_koreni.setText("Jednacina nema realnih resenja!");
```

```
          else
            labela_koreni.setText("" + resenja[0]);
        }
     });


panel3 = new JPanel(new GridLayout(0, 1));
panel3.setBorder(BorderFactory.createTitledBorder(
     BorderFactory.createEtchedBorder(Color.red, Color.orange),
                                          "Izbor boje"));

JPanel panel3A = new JPanel();
lcrvena= new JLabel("crvena:");
crvena = new JTextField(4);

lzelena = new JLabel("zelena:");
zelena = new JTextField(4);

lplava = new JLabel("plava:");
plava = new JTextField(4);

panel3A.add(lcrvena);
panel3A.add(crvena);
panel3A.add(lzelena);
panel3A.add(zelena);
panel3A.add(lplava);
panel3A.add(plava);
panel3.add(panel3A);

JPanel panel3B = new JPanel(new FlowLayout(
                                    FlowLayout.LEFT, 20, 5));
boja = new JButton("Postavi boju");
boja.setPreferredSize(new Dimension(120, 25));
polje_boja = new JTextField("");
polje_boja.setPreferredSize(new Dimension(200, 25));
polje_boja.setEditable(false);
panel3B.add(boja);
panel3B.add(polje_boja);
panel3.add(panel3B);
add(panel3);

boja.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
           try {
             int R = Integer.parseInt(crvena.getText());
             int G = Integer.parseInt(zelena.getText());
             int B = Integer.parseInt(plava.getText());

             if(R < 0 || R > 255 || G < 0 || G > 255 || B < 0 || B > 255)
             {
               polje_boja.setText("Dozvoljeni intenzitet boje: [0, 255]");
               return;
             }

             bojaZaIspis = new Color((R<<16) ^ (G<<8) ^ B);
             polje_boja.setBackground(bojaZaIspis);
           }
           catch(NumberFormatException nfe)
           {
             polje_boja.setText("Intenzitet boje je numericki podatak!");
           }
        }
     });

panel4 = new JPanel();
```

```
      panel4.setBorder(BorderFactory.createTitledBorder(
           BorderFactory.createEtchedBorder(Color.red, Color.orange),
                                              "Prikaz svih jednacina"));
      prikaziSve = new JButton("Kvadratne jednacine");
      panel4.add(prikaziSve);
      add(panel4);

      prikaziSve.addActionListener(
           new ActionListener() {
              public void actionPerformed(ActionEvent e)
              {
                 PrintWriter izlaz;

                 try
                 {
                   izlaz = new PrintWriter(new FileWriter("C:\\Temp\\izlaz.txt"));
                 }
                 catch(IOException IOe)
                 {
                   System.out.println("Neuspela operacija pripreme za upis " +
                                      "u fajl izlaz.txt");
                   return;
                 }

                 for(KvadratnaJednacina jednacina : jednacine)
                 {
                   double[] koreni = jednacina.koreni();
                   izlaz.print(jednacina + "  ");
                   if(koreni[0] != koreni[1])
                     izlaz.print(koreni[0] + "    " + koreni[1]);
                   else if(koreni[0] == 0)
                     izlaz.print("Nema realnih resenja!");
                   else
                     izlaz.print(koreni[0]);
                   izlaz.println();
                   izlaz.flush();
                 }
              }
           });

      setVisible(true);
   }
}
```

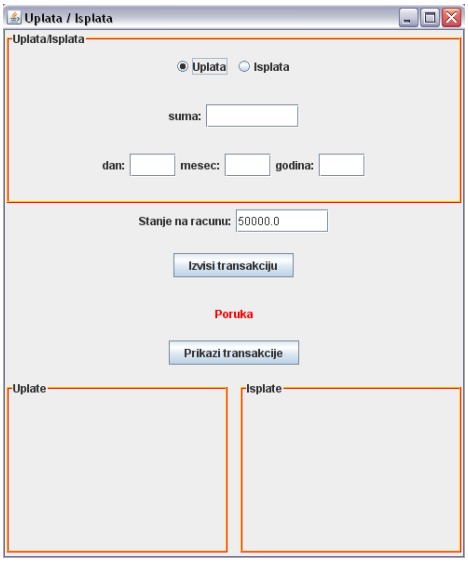
14. *Испит, јун 2009, Трансакције*.

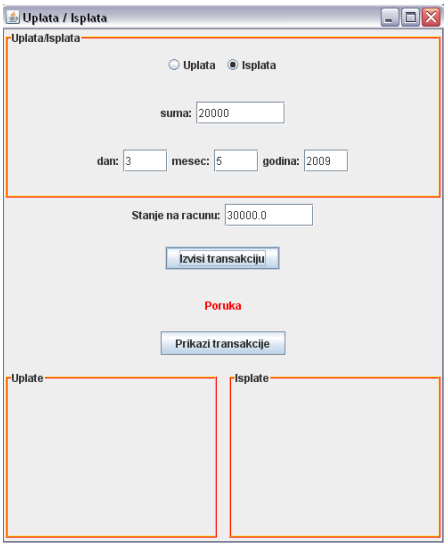
Написати апликацију која представља интерфејс за рад са трансакцијама у оквиру **једног** корисничког рачуна у банци.

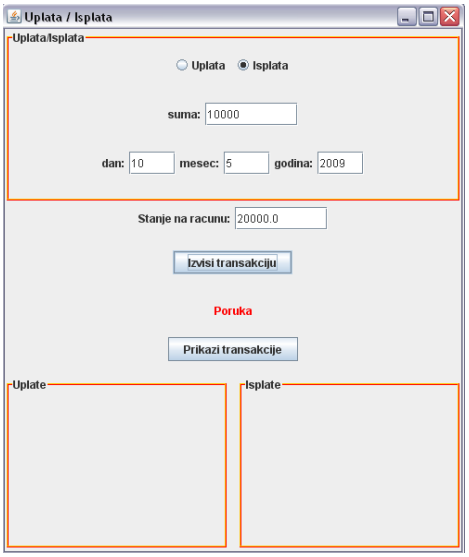
Прозор апликације има наслов *Uplata/Isplata* и треба га организовати у целине које садрже:

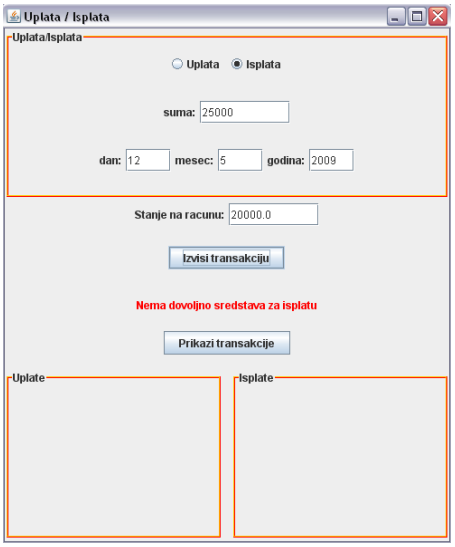
1. Два радио дугмета *Uplata* и *Isplata*. Кликом на дугме бира се тип трансакције који се врши.
   Иницијално је селектовано дугме *Uplata*. Текстуално поље за унос суме новца за трансакцију.
   Три текстуална поља за унос датума (дан, месец, година) транскакције.
   Обавезно извршити све видове провере коректности података имајући у виду природу самих података и оно што подаци представљају.
   Трансакцију дефинисати на погодан начин, без коришћења наслеђивања.
2. Текстуално поље у коме је приказано текуће стање на рачуну.
   Дугме *Izvrsi transakciju*. Кликом на дугме реализује се изабрани тип трансакције. Уколико трансакција може да се реализује, подаци о трансакцији смештају се у одговарајућу колекцију и ажурира се текуће стање на рачуну. Неуспешне трансакције се не додају у колекцију!
   У случају неуспешне трансакције у лабели *Poruka* исписати одговарајућу поруку.

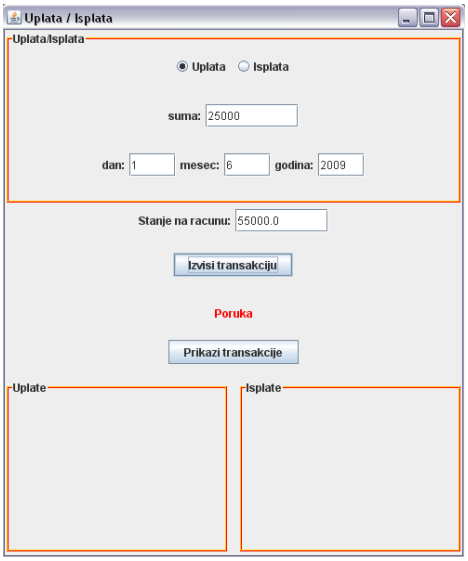
3. Дугме *Prikazi transakcije*. Кликом на дугме приказују се трансакције у одвојеним групама – посебно уплате, посебно исплате.
   Трансакције су приказане употребом *check-boxova*.
   Формат приказа трансакције: *Suma novca Datum transakcije*
   Све трансакције које су извршене после првог у текућем месецу треба да буду чекиране.
   Онемогућити кориснику било какву промену стања приказаних *check-boxova*!

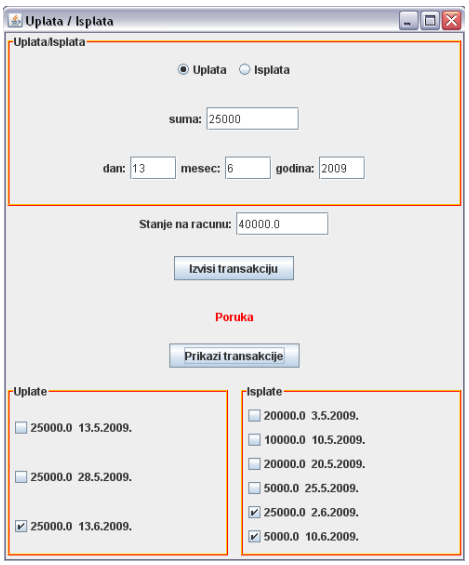





Uplata / Isplata
Uplata/Isplata
Uplata
Isplata
suma: 10000
dan: 10 mesec: 5 godina: 2009
Stanje na racunu: 20000.0
Izvisi transakciju
Poruka
Prikazi transakcije
Uplate
Isplate


Uplata / Isplata
Uplata/Isplata
Uplata
Isplata
suma: 25000
dan: 12 mesec: 5 godina: 2009
Stanje na racunu: 20000.0
Izvisi transakciju
Nema dovoljno sredstava za isplatu
Prikazi transakcije
Uplate
Isplate


Uplata / Isplata
Uplata/Isplata
Uplata
Isplata
suma: 25000
dan: 1 mesec: 6 godina: 2009
Stanje na racunu: 55000.0
Izvisi transakciju
Poruka
Prikazi transakcije
Uplate
Isplate


Uplata / Isplata
Uplata/Isplata
Uplata
Isplata
suma: 25000
dan: 13 mesec: 6 godina: 2009
Stanje na racunu: 40000.0
Izvisi transakciju
Poruka
Prikazi transakcije
Uplate
25000.0 13.5.2009.
25000.0 28.5.2009.
25000.0 13.6.2009.
Isplate
20000.0 3.5.2009.
10000.0 10.5.2009.
20000.0 20.5.2009.
5000.0 25.5.2009.
25000.0 2.6.2009.
5000.0 10.6.2009.

Објашњење:

У класи *Datum* у методу *posle()* испитује се да ли датум који се дефинише долази после датума који је задат као аргумент.
Класа *Transakcija* дефинише трансакцију. Тип трансакције одређен је статичким константама UPLATA, ISPLATA. Све трансакције односе се на један исти рачун у банци, па је чланица која представља рачун статичка.
Свака трансакција карактерише се својим типом, сумом новца за реализацију трансакције и датумом трансакције.
Метод *postaviPocetnoStanje()* поставља иницијалну вредност за стање рачуна.
Метод *izvrsiTransakciju()* врши реализацију трансакције. Реализација зависи од типа трансакције. Ако је у питању UPLATA, сума новца се додаје на текуће стање рачуна. Ако је у питању ISPLATA, текуће стање се умањује за дату суму, при чему није дозвољено да стање на рачуну оде у минус. Ако је то случај, трансакција се неће извршити.

Садржај прозора апликације обухвата три панела.
Панел *panel1* са насловним оквиром *Uplata/Isplata* организован је у три потпанела. Први садржи два радио дугмета *Uplata* и *Isplata* за избор типа трансакције. Други обухвата поље за унос суме новца за трансакцију, а трећи поља за унос датума трансакције.
Панел *panel2* организован је у четири потпанела. Први садржи поље где је у сваком тренутку приказано текуће стање на рачуну. Други садржи дугме *Izvrsi transakciju*, трећи лабелу за испис поруке, а четврти дугме *Prikazi transakcije*.
Панел *panel3* организован је у два потпанела са насловним оквирима *Uplate*, односно, *Isplate*, који служе за приказ извршених уплата, односно, исплата.

Трансакције се смештају у колекцију. Овде је коришћена класа *java.util.Vector*. Може се користити и класа *ArrayList* која је јако слична класи *java.util.Vector*.
На почетку се поставља иницијално стање рачуна на неку вредност.

Обрада догађаја:

- Клик на дугме *Izvrsi transakciju*. Проверавају се вредности које корисник задаје – датум трансакције, сума и тип трансакције и на основу тога креира објекат који представља трансакцију. Ако трансакција не може да се изврши, исписује се у лабели одговарајућа порука. Таква трансакција се не сматра валидном, па неће бити ни додата у колекцију. Ако је трансакција успешно извршена, додаје се у колекцију и ажурира се вредност која је исписана у пољу за приказ текућег стања.
- Клик на дугме *Prikazi transakcije*. За сваку трансакцију из колекције креира се *check-box* дугме. Ако је трансакција извршена после 1.6.2009., дугме ће бити чекирано. Уплате се приказују на левом панелу, а исплате на десном. Уједно се обрађују догађаји ниског нивоа над *check-box* дугметом. Када се мишем уђе на површину дугмета, дугме постаје неактивно, чиме се онемогућује да корисник промени статус дугмета (селектовано/деселектовнао). Када се напусти површина дугмета, дугме поново постаје активно.

Решење:

Datum.java

```
package transakcije;

public class Datum {
   private int dan;
   private int mesec;
   private int godina;
```

```
   public Datum(int dan, int mesec, int godina)
   {
      this.dan = dan;
      this.mesec = mesec;
      this.godina = godina;
   }

   public Datum(final Datum d)
   {
      this.dan = d.dan;
      this.mesec = d.mesec;
      this.godina = d.godina;
   }

   public static boolean validan(int dan, int mesec, int godina)
   {
      if(dan < 1 || dan > 31)
         return false;
      if(mesec < 1 || mesec > 12)
         return false;
      return true;
   }

   public boolean posle(Datum d)
   {
      if(godina < d.godina)
         return false;
      else if (godina > d.godina)
         return true;
      else if(mesec < d.mesec)
         return false;
      else if(mesec > d.mesec)
         return true;
      else if(dan < d.dan)
         return false;
      else if(dan > d.dan)
         return true;
      else return false;
   }

   public String toString()
   {
      return "" + dan + "." + mesec + "." + godina + ".";
   }
}
```

Transakcija.java

```
package transakcije;

public class Transakcija {
   public static final int UPLATA = 0;
   public static final int ISPLATA = 1;

   public static double racun;

   private double suma;
   private Datum datum;
   private int tip;

   public Transakcija(double suma, int dan,
        int mesec, int godina, int tip)
   {
      this.suma = suma;
      datum = new Datum(dan, mesec, godina);
      this.tip = tip;
   }
```

```
   public Transakcija(double suma, Datum d, int tip)
   {
      this.suma = suma;
      datum = new Datum(d);
      this.tip = tip;
   }

   public static void postaviPocetnoStanje(double stanje)
   {
      racun = stanje;
   }

   public static double vratiTekuceStanje()
   {
      return racun;

   }

   public double suma()
   {
      return suma;
   }

   public int tip()
   {
      return tip;
   }

   public Datum datum()
   {
      return datum;
   }

   public boolean izvrsiTransakciju()
   {
      boolean ind = true;
      switch(tip)
      {
      case UPLATA:
         racun += suma;
         break;
      case ISPLATA:
         if(racun - suma >= 0)
         {
            racun -= suma;
            ind = true;
         }
         else ind = false;
         break;
      }
      return ind;
   }

   public String toString()
   {
      return "" + suma + "  " + datum;
   }
}
```

UplataIsplata.java

```
package transakcije;

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
```

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;

public class UplataIsplata {

   private static JFrame prozor =
                  new JFrame("Uplata / Isplata");
   private static JPanel panel1;
   private static JPanel panel2;
   private static JPanel panel3;
   private static JPanel panel3A;
   private static JPanel panel3B;

   private static JTextField suma;
   private static JRadioButton uplata;
   private static JRadioButton isplata;
   private static JTextField dan;
   private static JTextField mesec;
   private static JTextField godina;

   private static JTextField stanje;
   private static JLabel labela_poruka;

   private static Vector<Transakcija> transakcije =
                  new Vector<Transakcija>();

   public static void main(String[] args) {
      SwingUtilities.invokeLater(
           new Runnable() {
              public void run(){
                 kreirajGUI();
              }
           });
   }

   public static void kreirajGUI() {

      Transakcija.postaviPocetnoStanje(50000);

      Container sadrzaj = prozor.getContentPane();
      sadrzaj.setLayout(new GridLayout(0, 1));
      prozor.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      prozor.setBounds(50, 100, 500, 600);


      panel1 = new JPanel(new GridLayout(0,1));
      panel1.setBorder(BorderFactory.createTitledBorder(
           BorderFactory.createEtchedBorder(
                Color.RED, Color.ORANGE), "Uplata/Isplata"));

      JPanel panel1A = new JPanel();
```

```
uplata = new JRadioButton("Uplata", true);
isplata = new JRadioButton("Isplata", false);
ButtonGroup grupa = new ButtonGroup();
grupa.add(uplata);
grupa.add(isplata);
panel1A.add(uplata);
panel1A.add(isplata);
panel1.add(panel1A);

JPanel panel1B = new JPanel();
JLabel suma_labela = new JLabel("suma:");
suma = new JTextField("");
suma.setPreferredSize(new Dimension(100, 25));
panel1B.add(suma_labela);
panel1B.add(suma);
panel1.add(panel1B);

JPanel panel1C = new JPanel();
JLabel dan_labela = new JLabel("dan:");
dan = new JTextField("");
dan.setPreferredSize(new Dimension(50, 25));

JLabel mesec_labela = new JLabel("mesec:");
mesec = new JTextField("");
mesec.setPreferredSize(new Dimension(50, 25));

JLabel godina_labela = new JLabel("godina:");
godina = new JTextField("");
godina.setPreferredSize(new Dimension(50, 25));

panel1C.add(dan_labela);
panel1C.add(dan);
panel1C.add(mesec_labela);
panel1C.add(mesec);
panel1C.add(godina_labela);
panel1C.add(godina);

panel1.add(panel1C);
sadrzaj.add(panel1);

panel2 = new JPanel(new GridLayout(0, 1));
JPanel panel2A = new JPanel();
JLabel stanje_labela = new JLabel("Stanje na racunu:");
stanje = new JTextField(Double.valueOf(
       Transakcija.vratiTekuceStanje()).toString());
stanje.setPreferredSize(new Dimension(100, 25));
panel2A.add(stanje_labela);
panel2A.add(stanje);
panel2.add(panel2A);

JPanel panel2B = new JPanel();
JButton transakcija = new JButton("Izvisi transakciju");
panel2B.add(transakcija);
panel2.add(panel2B);

transakcija.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           try {
             double iznos = Integer.parseInt(suma.getText());
             int d = Integer.parseInt(dan.getText());
             int m = Integer.parseInt(mesec.getText());
             int g = Integer.parseInt(godina.getText());

             if(!Datum.validan(d, m, g))
               labela_poruka.setText("Nekorektan datum!");
```

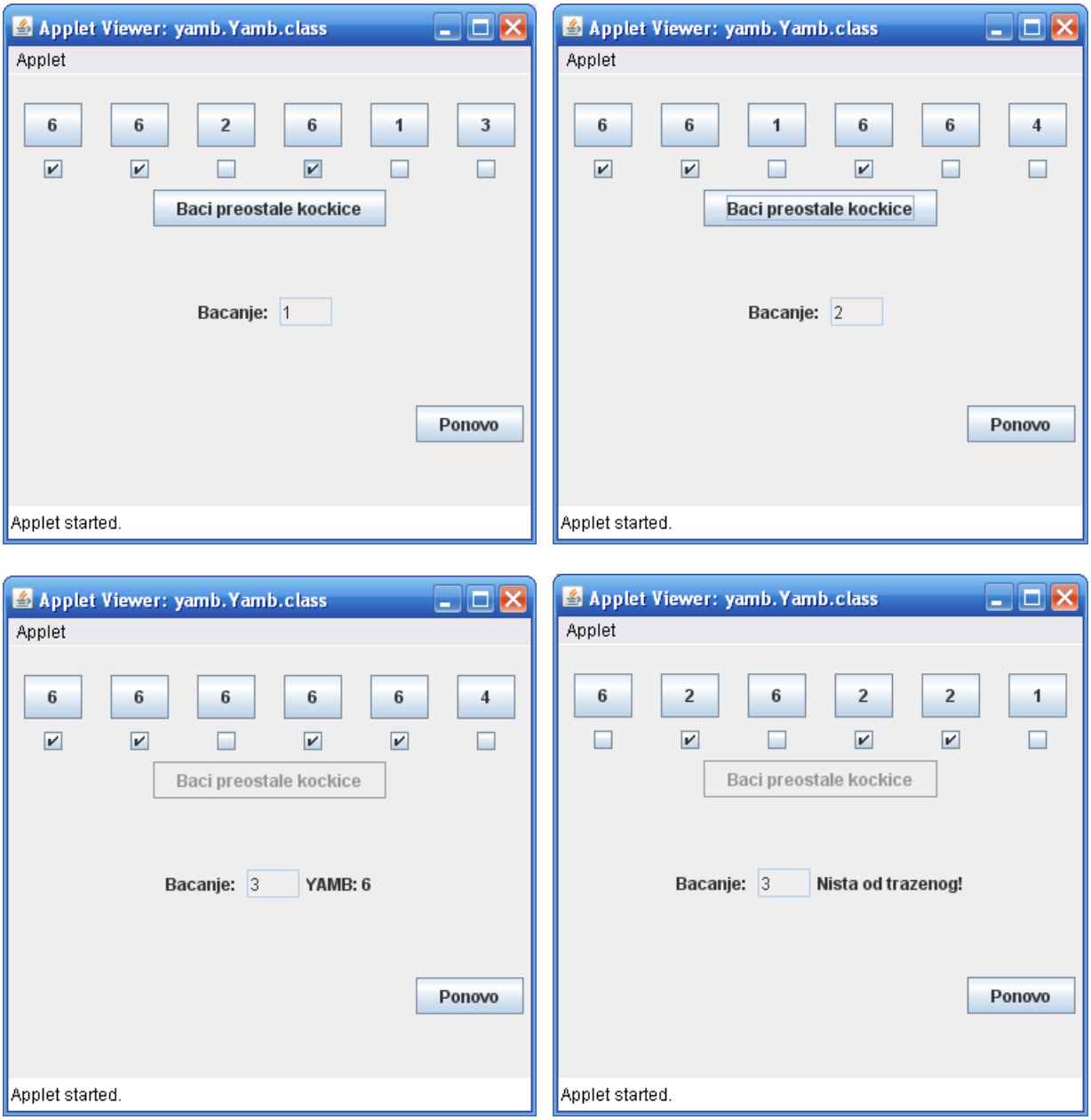
```
            Transakcija t;
            if(uplata.isSelected())
              t = new Transakcija(iznos, d, m, g, Transakcija.UPLATA);
            else
              t = new Transakcija(iznos, d, m, g, Transakcija.ISPLATA);

            if(!t.izvrsiTransakciju())
              labela_poruka.setText("Nema dovoljno sredstava za isplatu");
            else
            {
              transakcije.add(t);
              labela_poruka.setText("Poruka");
            }

            stanje.setText(String.valueOf(Transakcija.vratiTekuceStanje()));
          }
          catch(NumberFormatException nfe)
          {
            labela_poruka.setText("Uneseni podaci moraju biti brojevi");
          }
        }
      });

labela_poruka = new JLabel("Poruka");
labela_poruka.setForeground(Color.red);
labela_poruka.setHorizontalAlignment(SwingConstants.CENTER);
panel2.add(labela_poruka);

JPanel panel2C = new JPanel();
JButton prikazi = new JButton("Prikazi transakcije");
panel2C.add(prikazi);
panel2.add(panel2C);
prozor.add(panel2);

prikazi.addActionListener(
     new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          for(Transakcija t : transakcije)
          {
            JCheckBox cbox = new JCheckBox(t.toString());

            cbox.addMouseListener(
                new MouseAdapter()
                {
                  public void mouseEntered(MouseEvent e)
                  {
                    e.getComponent().setEnabled(false);
                  }

                  public void mouseExited(MouseEvent e)
                  {
                    e.getComponent().setEnabled(true);
                  }

                });

            if(t.datum().posle(new Datum(1, 6, 2009)))
              cbox.setSelected(true);

            if(t.tip() == Transakcija.UPLATA)
            {
              panel3A.add(cbox);
              panel3A.validate();
            }

            else
            {
```

```
                    panel3B.add(cbox);
                    panel3B.validate();
                }

            }

        }
    });

    panel3 = new JPanel(new GridLayout(1, 2, 10, 5));
    panel3A = new JPanel(new GridLayout(0, 1));
    panel3A.setBorder(BorderFactory.createTitledBorder(
        BorderFactory.createEtchedBorder(Color.RED, Color.ORANGE), "Uplate"));

    panel3.add(panel3A);

    panel3B = new JPanel(new GridLayout(0, 1));
    panel3B.setBorder(BorderFactory.createTitledBorder(
        BorderFactory.createEtchedBorder(Color.RED, Color.ORANGE), "Isplate"));
    panel3.add(panel3B);

    prozor.add(panel3);
    prozor.setVisible(true);
  }
}
```

15. *јун 2009, YAMB.* Написати аплет чији графички кориснички интерфејс треба да изгледа као на приказаним сликама.
У горњем делу прозора има 6 дугмади која представљају коцкице за YAMB.
Испод сваког дугмета налази се по један *check-box*. Ако је он чекиран, то значи да се одговарајућа коцкица не баца у наредном бацању. Притисак на дугме треба да промени статус одговарајућег *check-box*-а (чекиран/није чекиран).
Коцкице се бацају 3 пута (што се симулира притиском на дугме "Баци преостале коцкице"), а редни број бацања се бележи у једном текстуалном пољу у коме је онемогућен унос корисника.
Након трећег бацања, онемогућити даље догађаје притиском на дугме "Баци преостале коцкице", проверити да ли је добијена кента, каре, "јамб" или ништа од тога и исписати одговарајућу поруку о томе у једној лабели на прозору.
Притисак на дугме "Поново", које се налази у доњем делу прозора, треба да доведе аплет у првобитно стање како би играч могао поново да игра.

(**Мини упутство за YAMB**) Приликом испитивања добијене комбинације, једна коцкица се не узима у обзир.
Добили сте "јамб" ако на преосталих 5 коцкица имате исте бројеве. Ако су то нпр. 2-јке, одговарајућа порука коју треба исписати у лабели је: "YAMB: 2"
Каре је комбинација 4 иста броја на коцкицама. Ако имате четири 6-ице, испишите поруку "КАРЕ: 6".
Кента је комбинација 1,2,3,4,5 или 2,3,4,5,6 на коцкицама које се узимају у обзир. Одговарајуће поруке које треба исписати у лабели биле би "КЕНТА: 1 2 3 4 5" и "КЕНТА: 2 3 4 5 6". (Ако је комбинација баш 1,2,3,4,5,6 испишите било коју од ове две поруке).
Ако није добијена ни кента, ни каре, ни "јамб", у лабели исписати поруку "Ништа од траженог!".

Изглед прозора програма у разним тренуцима извршавања приказан је на сликама:



Објашњење:
Осим компоненти које учествују у обради догађаја, класа има и инстанцну променљиву *bacanje* типа *int* која означава редни број бацања и иницијализована је вредношћу 1.

Layout manager аплета је *GridLayout manager* који има једну колону, а број врста је 0, што значи да ће их бити довољно да се смести сав садржај који буде додат. На површ аплета затим се, редом, додају следећи панели:

- *kockiceSve*, са потпанелима: *kockicePanel[0]*,…,*kockicePanel[brojKockica-1]*. Панел *kockicePanel[i], i=1,..,brojKockica-1* има *GridLayout manager* са 2 врсте и 1 колоном. У првој врсти је дугме *kockice[i]* које представља *i*-ту коцкицу, а у другој је панел *kockicica[i]*, у који је смештен одговарајући *check-box kockiceCB[i]*. Лабеле на дугмади која представљају коцкице иницијализују се *String*-репрезентацијама случајних бројева из интервала [1-6].
- *sledeceBacanjePanel* (само дугме *sledece*)

- *bacanjePanel* (лабела и текстуално поље за испис редног броја бацања, у коме је позивом метода *setEditable()* са аргументом *false* онемогућен унос корисника, као и лабела за испис резултујуће поруке)
- *ponovoPanel* (само дугме *ponovo*)

Објашњења обрада догађаја (сви ослушкивачи догађаја су објекти одговарајућих анонимних класа):

- дугме *kockice[i]*: потребно је променити статус *i*-тог *check-box*-а. Како је *i* локална променљива у класи аплета, у анонимној класи ослушкивача догађаја немамо приступ до ње, те њену вредност одређујемо на следећи начин: користећи метод *getSource()* дохватамо референцу на дугме које је извор догађаја, тј. *i*-то дугме, а затим, проласком кроз низ *kockice* свих дугмади и поређењем текућег елемента низа са добијеном референцом на дугме које је извор догађаја налазимо вредност за *i*. Потом очитамо статус *i*-тог *check-box*-а и променимо га.
- дугме *sledece*: пролазимо кроз низ дугмади који представља коцкице и, уколико одговарајући *check-box* за дугме није чекиран, што значи да коцкица учествује у бацању, симулирамо њено бацање тако што на дугмету исписујемо *String*-репрезентацију случајног броја из интервала [1-6]. Потом се инкрементира бројач бацања и ажурирана вредност исписује у одговарајућем текстуалном пољу на површини аплета. Затим се проверава критеријум заустављања, па ако је испуњен, тј. ако смо имали 3 бацања, позивом метода *setEnabled()* са аргументом *false* онемогућујемо дугме *sledece* и крећемо у испитивање добијене комбинације. За то нам је потребан помоћни низ, *brojaci*, у коме памтимо колико којих коцкица имамо. Једним пролазом кроз низ утврђујемо имамо ли "јамб" или каре. Ако имамо бар 5 истих коцкица, добили смо "јамб". Иначе, ако имамо 4 исте коцкице, добили смо каре. Нема потребе проверавати остатак низа, јер ни "јамб" ни каре не можемо имати са неким другим бројем, кад нам 4 коцкице отпадају на овај са којим смо већ добили каре. За испитивање да ли смо добили кенту имамо 3 варијанте. Променљива *kenta1* означава комбинацију 1, 2, 3, 4, 5, док променљива *kenta2* означава комбинацију 2, 3, 4, 5, 6. Добили смо неку од ове две комбинације ако се сваки од бројева који учествује у комбинацији појављује на бар једној од коцкица. Прва и трећа варијанта испитују овај услов коришћењем логичког оператора &&, а друга коришћењем чињенице да је производ ненегативних бројева 0 акко је бар један од чинилаца 0. Ако добијена комбинација није ни "јамб", ни каре, ни кента, у одговарајуће текстуално поље на површи аплета уписује се порука да нисмо добили ништа од траженог.
- дугме *ponovo*: уписивањем празног *String*-а у текстуално поље *rezultat* "брише" се порука о претходно добијеној комбинацији, бројач бацања се ресетује на 1 и та вредност се уписује у одговарајуће текстуално поље на површи аплета, омогућује се дугме *sledece*, а на низ дугмади *kockice* поставља нова комбинација бројева.

Решење:

YAMB.java

```
package yamb;

import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
```

```
public class YAMB extends JApplet
{

   final static int brojKockica = 6;
   private JButton kockice[] = new JButton[brojKockica];
   private JCheckBox kockiceCB[] = new JCheckBox[brojKockica];
   private Random random = new Random();

   private JButton sledece = new JButton("Baci preostale kockice");

   private int bacanje = 1;
   private JTextField bacanjeTF = new JTextField(3);

   private JLabel rezultat = new JLabel("");

   public void init()
   {
      SwingUtilities.invokeLater(new Runnable()
      {
         public void run()
         {
            createGUI();
         }
      });
   }

   public void createGUI()
   {

      setSize(600, 400);

      Container content = getContentPane();
      content.setLayout(new GridLayout(0, 1));

      FlowLayout flow = new FlowLayout(FlowLayout.CENTER, 20, 20);
      JPanel kockiceSve = new JPanel();
      kockiceSve.setLayout(flow);

      JPanel[] kockicePanel = new JPanel[brojKockica];
      JPanel[] kockicica = new JPanel[brojKockica];
      for (int i = 0; i < brojKockica; i++)
      {
         kockicePanel[i] = new JPanel();
         kockicePanel[i].setLayout(new GridLayout(2, 1));
         kockice[i] = new JButton();
         kockice[i].addActionListener(new ActionListener()
         {
            public void actionPerformed(ActionEvent e)
            {
               JButton dugme = (JButton) e.getSource();
               for (int i = 0; i < brojKockica; i++)
                  if (kockice[i] == dugme)
                  {
                    kockiceCB[i].setSelected(!kockiceCB[i].isSelected());
                    return;
                  }
            }
         });
         kockicePanel[i].add(kockice[i]);
         kockiceCB[i] = new JCheckBox();
         kockicica[i] = new JPanel();
         kockicica[i].add(kockiceCB[i]);
         kockicePanel[i].add(kockicica[i]);
         kockice[i].setText(String.valueOf(1 + random.nextInt(brojKockica)));

         kockiceSve.add(kockicePanel[i]);
```

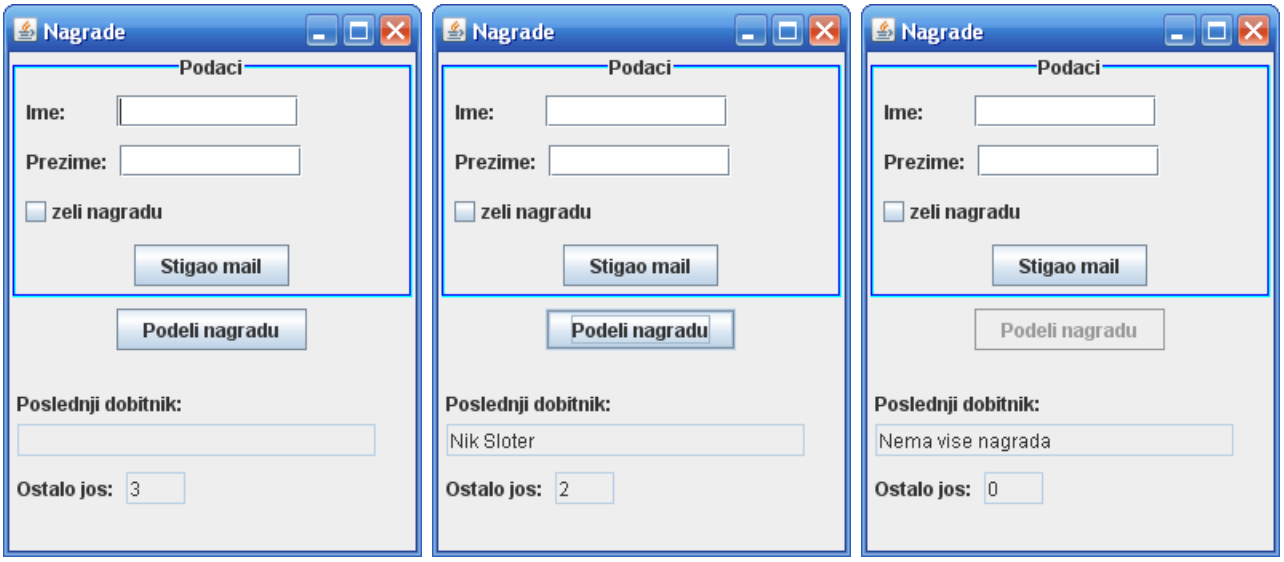
```
      }

      content.add(kockiceSve);

      JPanel sledeceBacanjePanel = new JPanel();
      sledeceBacanjePanel.add(sledece);
      sledece.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent e)
        {
          for (int i = 0; i < brojKockica; i++)
            if (!kockiceCB[i].isSelected())
              kockice[i].setText(
String.valueOf(1 + random.nextInt(brojKockica)));
          bacanje++;
          bacanjeTF.setText(bacanje + "");

          if (bacanje == 3)
          {

            sledece.setEnabled(false);

            // prebrojavamo koliko kojih kockica imamo
            int brojaci[] = new int[brojKockica];
            for (int j = 0; j < brojKockica; j++)
              brojaci[Integer.parseInt(kockice[j].getText()) - 1]++;

            // imamo li yamb ili kare?
            for (int j = 0; j < brojKockica; j++)
              if (brojaci[j] >= 5)
              {
                rezultat.setText("YAMB: " + (j + 1));
                return;
              } else if (brojaci[j] == 4)
              {
                rezultat.setText("KARE: " + (j + 1));
                return;
              }

            // imamo li kentu?
            boolean kenta1 = true;
            boolean kenta2 = true;

            for (int j = 0; (kenta1 || kenta2) && j < brojKockica - 1; j++)
            {
              kenta1 = kenta1 && (brojaci[j] > 0);
              kenta2 = kenta2 && (brojaci[j + 1] > 0);
            }

            if (kenta1)
            {
              rezultat.setText("KENTA: 1 2 3 4 5");
              return;
            }

            if (kenta2)
            {
              rezultat.setText("KENTA: 2 3 4 5 6");
              return;
            }
/*
            // Alternativni nacin za kentu
            int kenta1 = 1, kenta2 = 1;
            for (int j = 0; j < brojKockica - 1; j++)
            {
              kenta1 *= brojaci[j];
              kenta2 *= brojaci[j + 1];
```

```
            }
            if (kenta1 != 0)
            {
               rezultat.setText("KENTA: 1 2 3 4 5");
               return;
            }
            if (kenta2 != 0)
            {
               rezultat.setText("KENTA: 2 3 4 5 6");
               return;
            }
*/

/*
            // Kenta, 3. nacin
            if (brojaci[0] >= 1 && brojaci[1] >= 1 && brojaci[2] >= 1
                && brojaci[3] >= 1 && brojaci[4] >= 0)
            {
               rezultat.setText("KENTA: 1 2 3 4 5");
               return;
            }

            if (brojaci[1] >= 1 && brojaci[2] >= 1 && brojaci[3] >= 1
                && brojaci[4] >= 0 && brojaci[5] >= 1)
            {
               rezultat.setText("KENTA: 2 3 4 5 6");
               return;
            }
*/
            rezultat.setText("Nista od trazenog!");
         }

      }
   });
   content.add(sledeceBacanjePanel);

   JPanel bacanjePanel = new JPanel();
   bacanjePanel.add(new JLabel("Bacanje: "));
   bacanjePanel.add(bacanjeTF);
   bacanjeTF.setText(bacanje + "");
   bacanjeTF.setEditable(false);

   bacanjePanel.add(rezultat);

   content.add(bacanjePanel);

   JPanel ponovoPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
   JButton ponovo = new JButton("Ponovo");
   ponovo.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
      {
         rezultat.setText("");
         bacanje = 1;
         bacanjeTF.setText(bacanje + "");
         sledece.setEnabled(true);
         for (int i = 0; i < brojKockica; i++)
         {
            kockice[i].setText(
String.valueOf(1 + random.nextInt(brojKockica)));
            kockiceCB[i].setSelected(false);
         }
      }
   });
   ponovoPanel.add(ponovo);
   content.add(ponovoPanel);
```

```
    }

}
```

16. *јун 2009, Награде*. Једна радио-станица добила је од спонзора одређени број награда које треба да подели својим слушаоцима. Количина награда записана је у фајлу "`nagrade.txt`".
Слушаоци се за награде пријављују *mail*-ом.
За један *mail* битно је име и презиме пошиљаоца и да ли је у питању слушалац који се пријавио за награду, или *mail* нема везе са доделом награда.
Симулирање пристизања *mail*-ова врши се притиском на дугме које се налази у горњем делу прозора.
За поделу награда задужен је радник који свој посао обавља на следећи начин: он уђе у *inbox* и отвара *mail*-ове "одозго надоле", почев од последњег примљеног који није већ прочитао. Ако је у питању *mail* слушаоца који жели награду, радник му је додељује (додељивање награде симулира се притиском на дугме "Подели награду"), враћа се у *inbox* и наставља да отвара *mail*-ове на исти начин.
У једном текстуалном пољу, у коме је онемогућен унос корисника, треба исписивати име и презиме слушаоца коме је управо додељена награда, а у другом текстуалном пољу, у коме је такође онемогућен унос корисника, у сваком тренутку треба да пише број тренутно неподељених награда.
Ако у *inbox*-у тренутно нема *mail*-ова слушалаца који желе награде, а притиснуто је дугме "Подели награду", у пољу за приказ срећног добитника исписати поруку "Тренутно нема пријављених кандидата". Ако су све награде подељене, у истом текстуалном пољу исписати поруку "Нема више награда" и онемогућити даље догађаје притиска на дугме "Подели награду". У случају да је притиснуто дугме којим се симулира пристизање *mail*-а, а није попуњено поље за име или презиме пошиљаоца, у том пољу исписати поруку "ПОПУНИТИ ПОЉЕ! "
Након што су све награде подељене, фајл "`dobitnici.txt`" треба да има следећи садржај: најпре треба да буде наведен списак добитника, оним редом којим им је радник додељивао награде, а затим, након линије у којој пише "Нису добили награде: " и списак свих оних који нису добили награде, почев од "првог испод црте", па до оног који би последњи добио награду, да их је било довољно.

Изглед прозора програма у разним тренуцима извршавања приказан је на сликама:



Објашњење:

Класа *Osoba* је помоћна и од инстанцних променљивих има само *ime* и *prezime*. Нема потребе да се уводи и променљива која одређује да ли особа жели или не жели награду, јер ћемо у колекцију додавати само оне особе које желе награду, док ћемо остале игнорисати.

У класи *Prozor* урађен је главни део посла. Због начина на који радник обрађује *mail*-ове, подаци о њима смештају се на стек који је назван *kandidati*. У инстанцној променљивој *broj* налази се број тренутно неподељених награда чија иницијална вредност се чита из задатог фајла. Инстанцна променљива *izl* служи за исписивање коначних резултата у фајл. Остале инстанцне променљиве представљају компоненте које учествују у обради догађаја. Једини метод класе је конструктор. Након постављања наслова прозора и подешавања подразумеване акције при затварању прозора, из фајла "`nagrade.txt`" помоћу објекта класе *Scanner* чита се колико је награда добијено од спонзора. Приликом креирања овог објекта могуће је да дође до избацивања изузетка типа *FileNotFoundException*, па је тај део кода ограђен одговарајућим *try-catch* блоком. Како је наведено само име фајла, без путање, фајл би требало да се налази у текућем директоријуму пројекта. Прочитани број награда се уписује у одговарајуће текстуално поље на површи прозора. Потом се креира објекат за писање у фајл "`dobitnici.txt`". Опет, код је ограђен *try-catch* блоком јер може довести до избацивања изузетка типа *IOException*. Врло је битно да се овај код не налази у методу за обраду догађаја дугмета *podeli*, јер би се онда сваким притиском на то дугме изнова креирао празан фајл, а то није оно што овде желимо. Следи код за креирање графичког корисничког интерфејса.

Layout manager прозора апликације је *GridLayout manager* који има једну колону, а број врста је 0, што значи да ће их бити довољно да се смести сав садржај који буде додат. На површ прозора затим се, редом, додају следећи панели:

- *podaci* (*GridLayout manager* и *TitledBorder* граница), са потпанелима *imePanel* (лабела и текстуално поље за унос имена), *prezimePanel* (лабела и текстуално поље за унос презимена) и *mailPanel* (само дугме *mail*). Након додавања потпанела *prezimePanel*, а пре додавања потпанела *mailPanel*, у панел *podaci* се непосредно, без претходног уметања у сопствени панел, додаје и *check-box nagrada*.
- *podeliNagraduPanel* (*GridLayout manager*) са потпанелима *podeliPanel* (само дугме *podeli*), *dobitnikPanel* (лабела и текстуално поље за испис последњег добитника у коме је позивом метода *setEditable()* са аргументом *false* онемогућен унос корисника) и *preostaloPanel* (лабела и текстуално поље за испис броја тренутно неподељених награда у коме је позивом метода *setEditable()* са аргументом *false* онемогућен унос корисника).

Објашњења обрада догађаја (оба ослушкивача догађаја су објекти одговарајућих анонимних класа):

- дугме *mail*: ако пошиљалац жели награду (*check-box nagrada* је селектован), очитавамо име и презиме које је корисник унео, од тих података креирамо објекат типа *Osoba* и смештамо га на врх стека *kandidati*. Затим враћамо текстуална поља и *check-box* у првобитно (празно) стање. У случају да корисник није унео име или презиме, у одговарајућем текстуалном пољу се исписује порука "ПОПУНИТИ ПОЉЕ" и завршава обрада догађаја. Када пошиљалац *mail*-а не жели награду, не ради се ништа.
- дугме *podeli*: ако нема више награда, позивом метода *setEnabled()* са аргументом *false* дугме *podeli* се онемогућује, у текстуално поље за испис последњег добитника уписује се порука "Нема више награда" и, све док се не испразни стек, скида се један по један објекат са његовог врха и подаци о кандидату који није добио награду уписују у излазни фајл. Ако има још награда и има кандидата који их желе (стек није празан), додељује се једна награда: број расположивих награда се декрементира и ажурирана вредност се исписује у текстуалном пољу *preostalo*. Скида се објекат са врха стека – кандидат који је добио награду – и подаци о њему исписују и у текстуалном пољу *dobitnik* и у излазном фајлу. Иначе, тј. када награда има, али је стек празан, у пољу *dobitnik* исписује се порука "Тренутно нема пријављених кандидата".

У тест-класи се, на уобичајен начин, креира и приказује прозор апликације.

Решење:

`Osoba.java`

```
package nagrade;

public class Osoba
{
   private String ime;
   private String prezime;

   public Osoba(String ime, String prezime)
   {
      this.ime = ime;
      this.prezime = prezime;
   }

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

Prozor.java

```
package nagrade;

import java.awt.Color;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import java.util.Stack;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Prozor extends JFrame
{

   private JButton mail = new JButton("Stigao mail");
   private JButton podeli = new JButton("Podeli nagradu");
   private JTextField ime = new JTextField(10);
   private JTextField prezime = new JTextField(10);
   private JCheckBox nagrada = new JCheckBox("zeli nagradu");
   private JTextField dobitnik = new JTextField(20);
   private JTextField preostalo = new JTextField(3);

   private int broj = 0;

   private Stack<Osoba> kandidati = new Stack<Osoba>();

   private PrintWriter izl;
```

```
   public Prozor(String title)
   {
      setTitle(title);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // fajl iz koga citamo inicijalni broj nagrada
      Scanner sc = null;
      try
      {
         sc = new Scanner(new File("nagrade.txt"));

      } catch (FileNotFoundException e)
      {
         System.out.println("Nema fajla nagrade.txt");
         System.exit(1);
      }

      if (sc.hasNextInt())
         broj = sc.nextInt();

      sc.close();

      preostalo.setText(broj + "");

      // fajl u koji upisujemo dobitnike i kandidate koji nisu dobili nagrade
      try
      {
         izl = new PrintWriter(new FileWriter("dobitnici.txt"));
      } catch (IOException IOe)
      {
         System.out.println(
              "Neuspela operacija pripreme za upis u fajl dobitnici.txt");
         System.exit(1);
      }

      // Kreiranje GUI-ja
      Container content = getContentPane();
      content.setLayout(new GridLayout(0, 1));

      JPanel podaci = new JPanel(new GridLayout(0, 1));
      content.add(podaci);

      podaci.setBorder(
BorderFactory.createTitledBorder(
new EtchedBorder(Color.CYAN, Color.BLUE),
 "Podaci", TitledBorder.CENTER,
                         TitledBorder.DEFAULT_POSITION));

      JPanel imePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
      imePanel.add(new JLabel("Ime:         "));
      imePanel.add(ime);

      podaci.add(imePanel);

      JPanel prezimePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
      prezimePanel.add(new JLabel("Prezime: "));
      prezimePanel.add(prezime);

      podaci.add(prezimePanel);
      podaci.add(nagrada);

      JPanel mailPanel = new JPanel(new FlowLayout());
      mailPanel.add(mail);
      mail.addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent e)
```

```
   {
      if (nagrada.isSelected())
      {
         String imeS = ime.getText();
         String prezimeS = prezime.getText();
         if (imeS.equals(""))
         {
            ime.setText("POPUNITI POLJE!");
            return;
         } else if (prezimeS.equals(""))
         {
            prezime.setText("POPUNITI POLJE!");
            return;
         } else
            kandidati.push(new Osoba(imeS, prezimeS));
      }
      ime.setText("");
      prezime.setText("");
      nagrada.setSelected(false);

   }
});
podaci.add(mailPanel);

JPanel podeliNagraduPanel = new JPanel(new GridLayout(0, 1));

JPanel podeliPanel = new JPanel();
podeliPanel.add(podeli);
podeli.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
      if (broj == 0)
      {
         podeli.setEnabled(false);
         dobitnik.setText("Nema vise nagrada");
         izl.println("\n\nNisu dobili nagrade: ");
         while (!kandidati.isEmpty())
            izl.println(kandidati.pop().toString());
         izl.close();

      } else if (!kandidati.isEmpty())
      {
         broj--;
         preostalo.setText(broj + "");
         Osoba osoba = kandidati.pop();
         dobitnik.setText(osoba.toString());
         izl.println(osoba);
      } else
         dobitnik.setText("Trenutno nema prijavljenih kandidata");

   }
});
podeliNagraduPanel.add(podeliPanel);

JPanel dobitnikPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
dobitnikPanel.add(new JLabel("Posledнji dobitnik: "));
dobitnik.setEditable(false);
dobitnikPanel.add(dobitnik);

podeliNagraduPanel.add(dobitnikPanel);

JPanel preostaloPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
preostaloPanel.add(new JLabel("Ostalo jos: "));
preostalo.setEditable(false);
preostaloPanel.add(preostalo);
```

```
      podeliNagraduPanel.add(preostaloPanel);

      content.add(podeliNagraduPanel);

   }

}
```

Nagrade.java

```
package nagrade;

import java.awt.Dimension;
import java.awt.Toolkit;

import javax.swing.SwingUtilities;

public class Nagrade
{

   private static Prozor window;

   public static void main(String[] args)
   {
      SwingUtilities.invokeLater(new Runnable()
      {
         public void run()
         {
            createGUI();
         }
      });

   }

   static void createGUI()
   {
      window = new Prozor("Nagrade");
      Toolkit theKit = window.getToolkit();
      Dimension wndSize = theKit.getScreenSize();

      window.setBounds(wndSize.width / 4, wndSize.height / 4,
                       wndSize.width / 2, wndSize.height / 2);

      window.pack();
      window.setVisible(true);

   }

}
```

17. YOP. Написати аплет чији графички кориснички интерфејс треба да изгледа као на приказаним сликама.

У горњем делу прозора поставити 2 текстуална поља у која корисник уноси по један декадни цео број из интервала 0-255.

Притиском на дугме "Сабери", ако су у горња поља унете исправне вредности, у прве 2 лабеле у доњем делу прозора треба исписати записе унетих бројева у бинарној основи у пољима ширине 8, а затим је потребно извршити сабирање добијених бројева у систему са основом 2 и утврдити да ли је дошло до прекорачења (резултат такође представити 8-битним бинарним бројем). Добијени резултат, као и назнаку појаве прекорачења (карактер '*' испред резултата) исписати у трећој лабели у доњем делу прозора.

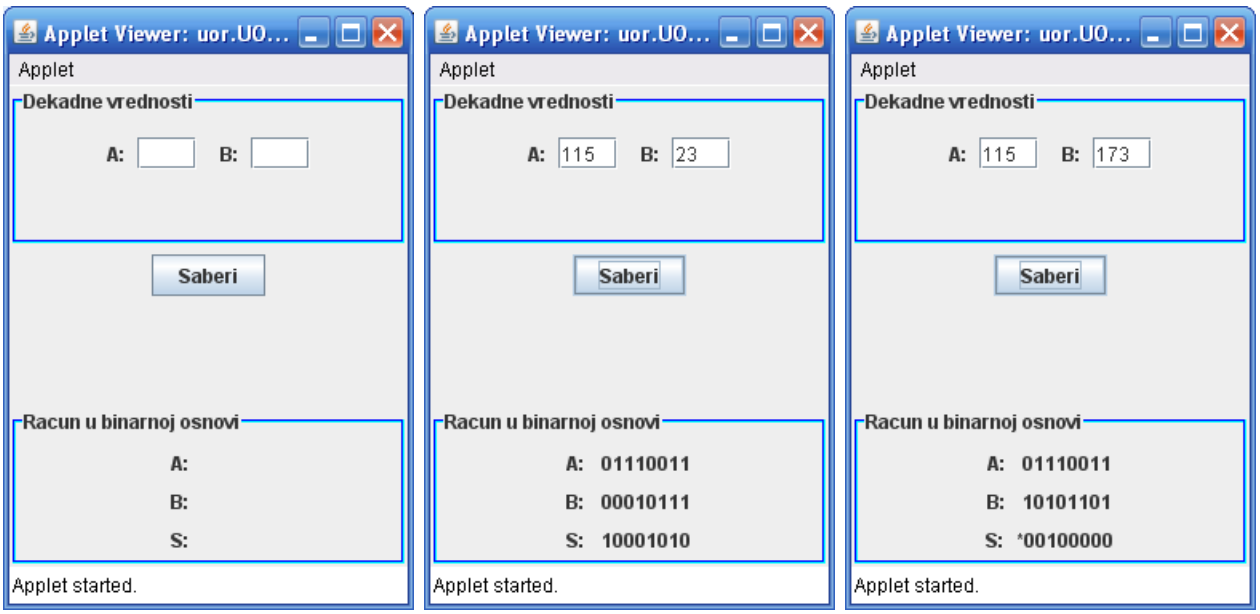
Ако нису унете исправне вредности у горња 2 поља, у лабели предвиђеној за приказ резултата исписати поруку:

"Број мора бити из опсега 0-255".

(Мини подсетник из УОР-а)
Претварање броја из декадне у бинарну основу: број се дели са 2 и памти се остатак. Ако је добијени количник једнак 0, поступак се завршава, а иначе се количник даље дели са 2 и памти се остатак…. Када у неком тренутку добијени количник постане 0, прочитају се унатрашке добијени остаци и то представља тражени запис броја у бинарној основи.

Сабирање бројева у бинарној основи врши се тако што се у сваком кораку сабирају цифре исте тежине сабирака и пренос са претходне позиције, при чему се добија цифра резултата на позицији те тежине, као и пренос за рачунање следеће цифре резултата. Креће се од позиције најмање тежине. Пренос на место најмање тежине је 0. До прекорачења је дошло ако постоји пренос са позиције највеће тежине.

Изглед прозора програма у разним тренуцима извршавања приказан је на сликама:



Објашњење:

Layout manager аплета је *GridLayout manager* који има једну колону, а број врста је 0, што значи да ће их бити довољно да се смести сав садржај који буде додат. Најпре се креира панел *brojeviPanel*, у који ће бити смештене компоненте за унос бројева у декадном систему. Панелу се, затим, додаје *TitledBorder* граница. За сваки број се креира посебан панел у који се додаје одговарајућа лабела и текстуално поље за унос броја. Коначно, панел *brojeviPanel* се додаје на површ аплета. *GridLayout manager* ће га сместити у прву врсту. У другу врсту биће смештен панел *saberiPanel*, у коме се налази само дугме *saberi*. Централни део програма представља обрада догађаја овог дугмета. У трећу врсту се смешта панел *binarnoPanel* са *TitledBorder* границом и *GridLayout manager*-ом са једном колоном и 0 врста, у који су додате компоненте за испис унетих бројева и њиховог збира у бинарној основи. Поново, сваки пар лабела/текстуално поље најпре је смештен у сопствени панел, који се затим додаје у *binarnoPanel*.

За потребе обраде догађаја дугмета *saberi* имплементиран је помоћни метод *dek2bin()* који прима један целобројни аргумет и, под претпоставком да се ради о броју из интервала [0, 255], о чему води рачуна позивајући метод *actionPerformed()* ослушкивача догађаја дугмета *saberi*, израчунава и враћа бинарну репрезентацију броја. Услов да је број из горњег интервала обезбеђује да његова бинарна

репрезентација може да се запише у пољу ширине 8. Бинарна репрезентација броја се одређује алгоритмом описаним у тексту задатка: све док број не постане нула, дели се бројем 2 и добијени остатак памти у низу. На крају се добијени низ остатака обрне и то постаје повратна вредност метода. На овај начин, за разлику од задатка ??? (велики бројеви), цифра најмање тежине броја бива смештена у последњи елемент низа.

Најзад, обрада догађаја дугмета *saberi* врши се на следећи начин: очитају се уноси корисника из одговарајућа два текстуална поља и методом *Integer.parseInt()* из њих издвоје унети цели бројеви. Потом се, помоћним методом *dek2bin(),* одређује њихова бинарна репрезентација, која се онда исписује у одговарајућа текстуална поља на површи аплета. Одговарајући *String* се од бинарне репрезентације добија надовезивањем у *for*-петљи *String*-репрезентација појединачних битова, почев од бита највеће тежине. Применом описаног алгоритма за сабирање бројева у бинарној основи, израчунава се тражени збир и одређује да ли је дошло до прекорачења. Коначно, резултат се уписује у одговарајућу лабелу.

За вежбу, модификовати решење задатка тако да цифра најмање тежине броја буде први елемент низа који представља његову бинарну репрезентацију.

Решење:

UOR.java

```
package uor;

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.border.EtchedBorder;

public class UOR extends JApplet
{

   private JTextField aTF = new JTextField(3);
   private JTextField bTF = new JTextField(3);

   private JButton saberi = new JButton("Saberi");

   private JLabel aL = new JLabel("         ");
   private JLabel bL = new JLabel("         ");

   private JLabel rezL = new JLabel("          ");

   public void init()
   {
      SwingUtilities.invokeLater(new Runnable()
      {
         public void run()
         {
            createGUI();
         }
      });
   }
```

```
   public void createGUI()
   {

      setSize(300, 300);
      Container content = getContentPane();

      content.setLayout(new GridLayout(0, 1));

      JPanel brojeviPanel = new JPanel();
      brojeviPanel.setBorder(
BorderFactory.createTitledBorder(
new EtchedBorder(Color.CYAN, Color.BLUE), "Dekadne vrednosti"));

      JPanel aBrojPanel = new JPanel();
      aBrojPanel.add(new JLabel("A: "));
      aBrojPanel.add(aTF);

      brojeviPanel.add(aBrojPanel);

      JPanel bBrojPanel = new JPanel();
      bBrojPanel.add(new JLabel("B: "));
      bBrojPanel.add(bTF);

      brojeviPanel.add(bBrojPanel);

      content.add(brojeviPanel);

      JPanel saberiPanel = new JPanel();
      saberiPanel.add(saberi);

      saberi.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent e)
        {
           int aBroj = 0;
           int bBroj = 0;
           try
           {
              aBroj = Integer.parseInt(aTF.getText().trim());
              bBroj = Integer.parseInt(bTF.getText().trim());

              if (aBroj < 0 || aBroj > 255 || bBroj < 0 || bBroj > 255)
              {
                 rezL.setText("Broj mora biti iz opsega 0-255");
                 return;
              }

              /*
               * ako je sve u redu, treba prevesti brojeve u binarni sistem,
               * sabrati ih, i utvrditi da li je doslo do prekoracenja
               */
              int a[] = dek2bin(aBroj);
              int b[] = dek2bin(bBroj);

              // upisujemo bin. reprezentaciju brojeva u odgovarajuce labele
              String aString = " ";
              for (int cifra : a)
                 aString += String.valueOf(cifra);

              String bString = " ";
              for (int cifra : b)
                 bString += String.valueOf(cifra);

              aL.setText(aString);
              bL.setText(bString);
```

```
            /*
* sabiranje i odredjivanje pojave prekoracenja,
             * krece se od mesta najmanje tezine!
             */
            int rez[] = new int[8];
            int prenos = 0;

            for (int i = 7; i >= 0; i--)
            {
              int suma = a[i] + b[i] + prenos;
              rez[i] = suma % 2;
              prenos = suma / 2;
            }

            /*
             * upisujemo rezultat u odgovarajucu labelu,
             * do prekoracenja je doslo ako je prenos sa mesta najvece tezine 1
             */
            String rezString = (prenos == 1) ? "*" : " ";
            for (int cifra : rez)
              rezString += String.valueOf(cifra);

            rezL.setText(rezString);

          } catch (NumberFormatException nfe)
          {
            rezL.setText("Broj mora biti iz opsega 0-255");
          }

        }
      });

      content.add(saberiPanel);

      JPanel binarnoPanel = new JPanel(new GridLayout(0, 1));
      binarnoPanel.setBorder(BorderFactory.createTitledBorder(new EtchedBorder(
          Color.CYAN, Color.BLUE), "Racun u binarnoj osnovi"));

      JPanel aBinarnoPanel = new JPanel();
      aBinarnoPanel.add(new JLabel("A: "));
      aBinarnoPanel.add(aL);

      binarnoPanel.add(aBinarnoPanel);

      JPanel bBinarnoPanel = new JPanel();
      bBinarnoPanel.add(new JLabel("B: "));
      bBinarnoPanel.add(bL);

      binarnoPanel.add(bBinarnoPanel);

      JPanel rezPanel = new JPanel();
      rezPanel.add(new JLabel("S: "));
      rezPanel.add(rezL);

      binarnoPanel.add(rezPanel);

      content.add(binarnoPanel);

    }

    private int[] dek2bin(int a)
    {
      int abin[] = new int[8];
      int i = 0;
      while (a != 0)
      {
        abin[i++] = a % 2;
```

```
        a /= 2;
      }
      /*
       * sada smo dobili cifre broja, ali u obrnutom poretku,
       * pa jos treba obrnuti dobijeni niz
       */
      for (i = 0; i < 4; i++)
      {
        int pom = abin[i];
        abin[i] = abin[7 - i];
        abin[7 - i] = pom;
      }

      return abin;

    }

}
```