Vježbe 3 - klase i sučelja

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

24. listopada, 2022.

Klase

Deklaracije klase omogućavaju **definiranje novih referenciranih tipova** i **opisivanje njihove implementacije**.

```
1  abstract class Tocka {//sadrzi apstraktnu funkciju
2      int x = 1, y = 1; //klasa mora biti abstract
3     static int brInstanci;
4      void pomakni(int pomakX, int pomakY) {
5          x += pomakX;
6          y += pomakY;
7          obavijesti();
8      }
9      abstract void obavijesti();//apstraktna funkcija
10     static void broji(){brInstanci++;}
11 }
12 abstract class ObojanaTocka extends Tocka {
13     int boja;  //nasljeduje apstraktnu funkciju
14 }              //isto mora biti apstraktna
```

Deklaracija klase

Ugnježđena klasa je klasa koja je deklarirana unutar tijela neke druge klase ili sučelja.

Klase

Klasa najvišeg nivoa je klasa koja nije ugnježđena.

```
1 class SredisteKruznice extends Tocka {
2  @Override void obavijesti() { //implementira funkciju
3    System.out.println("Srediste kruznice je u tocki"+"( "+x+
     ","+y+" )");
4       }
5  static void broji(){brInstanci+=2;}
6 }
```

Deklaracija klase

Klasa `SredisteKruznice` sadrži implementaciju funkcije `obavijesti`.
Kažemo da klasa **nadjačava** (eng. *override*) funkciju nadklase. To se može napraviti ukoliko je potpis funkcija u klasi i nadklasi (tip izlazne vrijednosti, ime, broj i tip formalnih argumenata) **identičan**, ili je potpis **identičan potpisu funkcije s fiksiranim tipovima**.
Ukoliko potpis nije identičan, ali je identično ime funkcije u klasi i nadklasi, tada kažemo da klasa **predefinira** (eng. *overload*) funkciju nadklase.

Klase

Kada klasa implementira ili promijeni implementaciju statičke funkcije nadklase (uz pretpostavku istih prototipova), tada kažemo da ta klasa **skriva** funkciju (isto se događa i s elementima članovima). Primjer je funkcija `broji` u klasama `Tocka` i `SredisteKruznice`.

```
1  class TockaNaPravcu extends Tocka {
2    float x, float y; //skriva elemente clanove
3   @Override void obavijesti() { //implementira funkciju
4     System.out.println("Tocka je: "+"( "+x+","+y+" )");
5        }
6   //predefinira funkciju
7   void pomakni(float pomakX, float pomakY){
8      x+=pomakX;
9      y+=pomakY;
10     obavijesti();
11  }
12
13 }
```

Deklaracija klase

Klase

```
1  final class Ishodiste extends Tocka {
2
3      Ishodiste(){
4          x = 0; y = 0;
5      }
6
7      @Override void obavijesti(){
8          System.out.println("Ishodiste je uvijek: "+" ( "+x+"
       ,"+y+" )");
9      }
10 }
```

Deklaracija klase

Klase koje su deklarirane kao `final` se **ne mogu** naslijediti. Možemo koristiti ključnu riječ `strictfp` i nad klasama (da bi osigurali **strogi način** računanja s realnim brojevima). Strogi način se uvijek koristi od Java verzije 17 nadalje.

Klase

Od Java verzije 17, moguće je definirati točno koje klase **smiju nasljeđivati** pojedinu klasu. To se postiže korištenjem ključne riječi `sealed`.

```
1  abstract public sealed class RijetkaTocka permits
       RijetkaImenovanaTocka {
2       HashMap<Integer,Double> vrijednosti;
3
4      public RijetkaTocka(){ vrijednosti = new HashMap<>(); }
5  }
6  //Klasa koja nasljeduje sealed klasu mora biti sealed (
       zasticena od nasljedivanja), non-sealed (nezasticena od
       nasljedivanja) ili final (nije moguce naslijediti).
7  public non-sealed class RijetkaImenovanaTocka extends
       RijetkaTocka {
8       String ime = "";
9
10      public RijetkaImenovanaTocka(String n){
11         super(); ime = n; }
12 }
```

Deklaracija klase

Ugniježdene klase

```
1  public class OpisanaTocka extends Tocka {
2
3      @Override void obavijesti(){
4          System.out.println("Tocka je: "+" ( "+x+","+y+" )");
5      }
6
7      public OpisanaTocka(){ x = -2; y = 2;  }
8
9    public class Opis{
10         String zapisao = "Marko Markic";
11         String pomaknuo  = "Pero Peric";
12         int xPocetna = -2, yPocetna = 2;
13
14   public Opis(String z, String p, int xP, int yP){
15     zapisao = z; pomaknuo = p; xPocetna = xP; yPocetna = yP;
16         }
17         .
18         .
19         .
```

Deklaracija ugniježdenih klasa

Ugniježdene klase

```
1           .
2           .
3           public void ispisiOpis(){
4                System.out.println("Zapisao: "+zapisao);
5                System.out.println("Pomaknuo: "+pomaknuo);
6                System.out.println("Pocetna lokacija: ( "+
    xPocetna+","+yPocetna+" )");
7           }
8       }
9
10      @Override void pomakni(int pomakX, int pomakY){
11          x+=pomakX; y+=pomakY;
12      }
13  }
```

Deklaracija ugniježdenih klasa

Ugniježdene klase

```
1 public class main {
2     public static void main(String args[]){
3           OpisanaTocka o = new OpisanaTocka();
4           o.obavijesti();
5
6          OpisanaTocka.Opis opis1 = o.new Opis("Duro Duric", "
    Petar Kresimir", 10, 15);
7          opis1.ispisiOpis();
8   }
9 }
```

Glavni program

Klasa `Opis` je **ugniježdena klasa** klase `OpisanaTocka` zato što je definirana **unutar tijela** klase `OpisanaTocka`. Ugniježdena klasa koja nije `static` se naziva **unutarnja klasa** (**ne može deklarirati** `static` elemente koji nisu konstanta). Ugniježdena klasa koja je `static` **nema referencu na klasu** u čijem tijelu se nalazi, stoga može referencirati **samo statičke** elemente te klase. Statička ugniježdena klasa može imati **statičke** i **ne-statičke** elemente članove i metode.

Record

Od Java verzije 16, uvedena je posebna verzije klase: `record`. Služi reprezentaciji klasa koje su jednostavan **skup agregiranih podataka**. Može biti osnovna klasa, član klase ili lokalna klasa. Ugniježdena klasa toga tipa je implicitno `static`.

```
1 public record NDimenzionalnaTocka(double[] koordinate,
      String ime) {
2
3 }
```

Klasa record

Ova jednostavna dekaracija će proizvesti zaštićena polja za elemente članove klase koordinate i ime, definirati funkcije za postavljanje i dohvaćanje vrijednosti tih elemenata, definirati konstruktor koji prima dva parametra tipa `double[]` i `String`, nadjačati funkcije `toString()`, `hashCode()`, `equals(Object o)`.
Dodatni konstruktori ili metode se moraju definirati i implementirati unutar definicije `recorda`.

Identifikatori pristupa

Ugniježdene klase, konstruktori, metode i elementi članovi mogu biti `default` (bez identifikatora), `private`, `protected` i `public` dok **klase** mogu biti samo `default` ili `public`.

- **Privatnim** objektima smije se pristupati **samo iz klase u kojima se nalaze**.
- **Default** identifikator označava da **nije definiran** niti jedan identifikator dozvole pristupa. Objekt je dohvatljiv **unutar klase** i **klasa smještenih u isti paket kao i dana klasa**. Podklase ne mogu pristupati elementima članovima i metodama nadklase osim ako nije definirana u istom paketu.
- **Protected** identifikator označava **ista dopuštenja pristupa** kao i **default** uz **dodatak da podklase mogu pristupati metodama i elementima članovima** čak i **ako se ne nalaze u istom paketu**.
- **Public** identifikator označava da se elementima članovima, konstruktorima ili metodama **smije pristupati iz svih klasa bez obzira u kojem paketu se nalaze**.

Pristupanje članovima nedostupne klase

```
1  package tocka;
2  public class Tocka {
3      public int x = 1, y = 1;
4
5  }
6
7  package vdtocka;
8  class Tocka3d extends tocka.Tocka{
9    public int z;
10 }
11
12 public class NovaTocka {
13     public static tocka.Tocka stvoriTocku() {
14         return new Tocka3d();
15     }
16 }
```

Kod paketa

Elementi x i y iz klase `tocka.Tocka` **su dohvatljivi** podklasi `vdtocka.Tocka3d`.

Pristupanje članovima nedostupne klase

```
1 package glavni;
2 class Glavna{
3   public static void main(String args[]){
4       tocka.Tocka t = vdtocka.NovaTocka.stvoriTocku();
5       System.out.println("Koordinate: "+t.x+" "+t.y);
6 }
```

Kod paketa koji sadrži main

Unutar paketa `glavni` **nije dohvatljiva** klasa `vdtocka.Tocka3d`. Unatoč tome, **možemo dohvatiti neke elemente** te klase (ne možemo dohvatiti element z).

Moguće je zabraniti poziv konstruktora osim iz tijela klase koji ga sadrži stavljanjem identifikatora pristupa ***private***, time **nije dozvoljeno instanciranje klase** osim iz **statičkih metoda** te iste klase.

Sučelja

Sučelja su **potpuno apstraktne klase** koje sadrže **prototipove funkcija i deklaracije konstanti**. Prototipove funkcija mogu implementirati klase.

```
1
2  interface Pomak{ //apstraktne funkcije
3    void simetrijaOkoX();
4    void simetrijaOkoY();
5    void premjesti(float xK, float yK);
6    void projecirajNaX();
7    void projecirajNaY();
8    final int MAX_X = 100; //konstante
9    final int MAX_Y = 100;
10 }
```

Primjer sučelja

Sve apstraktne funkcije u sučelju su javno dostupne (***public***). Svaka funkcija koja nadjačava neku funkciju mora imati **jednak identifikator dopuštenja pristupa ili veći** od nadjačane funkcije (stoga u klasi sve funkcije naslijeđene iz sučelja moraju biti ***public***).

Sučelja

```
1
2  class TockaNaPlatnu extends Tocka implements Pomak{
3
4    float x, y;
5
6       @Override void obavijesti(){
7          System.out.println("Tocka je: "+" ( "+x+","+y+" )");
8       }
9
10   @Override public void simetrijaOkoX(){ y = -y;}
11   @Override public void simetrijaOkoY(){ x = -x;}
12   @Override public void premjesti(float xK, float yK){x = xK;
         y = yK;}
13   @Override public void projecirajNaX(){y = 0;}
14   @Override public void projecirajNaY(){x = 0;}
15
16 }
```

Klasa koja implementira sučelje

Sučelja

Od *Java* verzije 8, dopušteno je korištenje `default` i `static` metoda unutar sučelja. Time je omogućeno da **sučelja sadrže funkcije s definiranom implementacijom**. Glavni razlog uvođenja `default` funkcija je mogućnost naknadnog ubacivanja funkcionalnosti u sučelja (bez potrebe re-implementacije svih klasa koje ga implementiraju). Kao posljedica, moguće je koristiti i privatne metode unutar sučelja, međutim ona se **ne mogu naslijediti i implementirati** unutar klasa (dostupna su samo unutar sučelja). **Funkcijska sučelja** su sučelja koja sadrže **samo jednu apstraktnu metodu**.

```
1
2 interface StandardniPomak{ //apstraktne funkcije
3   public static void ispisKomentara(){System.out.println("
      Pomak za jedan u desno");}
4   .
5   .
6   .
```

Sučelje s implementiranom metodom.

Sučelja

```
1  public default void ispis(Tocka t){
2  System.out.println("( "+t.x+","+"+t.y"+" )");}
3
4  private static void pomakniObije(Tocka t){
5    t.x+=1; t.y+=1;
6  }
7
8  private static void pomakniX(Tocka t){
9    t.x+=1;
10 }
11
12 private static void pomakniY(Tocka t){
13    t.y+=1;
14 }
15
16 .
17 .
18 .
```

Sučelje s implementiranom metodom.

Sučelja

```
1   public static void pomakni(Tocka t, int smijer){
2     if(smijer == 0)
3         pomakniX(t);
4     else if(smijer == 1)
5         pomakniY(t);
6     else pomakniObije(t);
7   }
8 }
9
10 public class main {
11     public static void main(String args[]){
12       OpisanaTocka o = new OpisanaTocka();
13       o.obavijesti();
14         StandardniPomak.pomakni(o, 2);
15         o.obavijesti();
16     }
17  }
```

Sučelje s implementiranom metodom.

Klase i sučelja

Višestruko nasljeđivanje sučelja **nije dozvoljeno**.

```
1 interface I<T> {}
2 class B implements I<Integer> {}
3 class C extends B implements I<String> {}
```

Višestruko nasljeđivanje sučelja.

Klasa *C* nasljeđuje `I<Integer>` (jer nasljeđuje klasu *B*) i nasljeđuje `I<String>` jer implementira sučelje *I* s argumentom tipa *String*. Kod takvog nasljeđivanja dolazi do greške pri prevođenju.

Višestruko nasljeđivanje metoda istog prototipa i elemenata članova **je dozvoljeno**. Nadjačana metoda **nadjačava obije nasljeđene metode** istog prototipa. Elementima članovima se **mora pristupati preko punog imena identifikatora** inače dolazi do greške pri prevođenju.

Sučelje se **može nalaziti unutar tijela klase**, međutim **klasa ne može pristupiti tom sučelju niti implementirati njegove funkcije**. Sučelju se **može pristupiti iz unutarnje klase** ili **preko imena klase unutar koje se nalazi**.

Klase i sučelja

```
1  public interface PomakZaDva {
2      public void pomakni(); final int p = 2;
3  }
4
5  public interface PomakZaTri {
6      public void pomakni(); final int p = 3;
7  }
8
9  public class AnimiranaTocka extends Tocka implements
       PomakZaDva, PomakZaTri {
10   @Override protected void obavijesti(){
11       System.out.println("Tocka je: "+" ( "+x+","+y+" )"); }
12           //x+=p, greska pri prevodenju
13     @Override public void pomakni(){x+=2; y+=2;}
14     public void moguciPomaci(){
15         System.out.println("1: "+PomakZaDva.p);
16         System.out.println("2: "+PomakZaTri.p); }
17 }
```

Višestruko nasljeđivanje metoda i elemenata članova.

Static, transient, volatile

static - nije potrebna instanca klase za korištenje, vidljiva svim instancama klase. Statička varijabla se **u nekim slučajevima može koristiti prije deklaracije** (zato što je inicijalizirana na početku pokretanja programa).

```
1  int w = x = 3; // ok - x s lijeve strane pridruzivanja
2  int p = x; // ok - inicijalizacija moze pristupiti static
       elementima
3
4  static int u =(new Object(){int bar(){ return x; } }).bar();
       // ok - dogada se u drugoj klasi
5
6      static int x;
7      static int vratiVrijednost() { return j; }
8      static int i = vratiVrijednost(); //i ima vrijednost 0
9      static int j = 1;
10
```

Korištenje static varijable prije deklaracije.

Elementi članovi (ne static) se **mogu koristiti unutar konstruktora iako su deklarirani ispod deklaracije konstruktora**.

Static, transient, volatile

transient - označava da varijabla nije dio trajnog stanja objekta. Varijabla neće biti spremljena ukoliko se objekt trajno spremi u memoriju.
volatile - koristi se kod višedretvenog programiranja. *Java virtualni stroj* osigurava da sve dretve u svakom trenutku vide konzistentnu vrijednost ***volatile*** varijabli.

```
1 class Tocka {
2     int x, y;
3     transient float rho, theta;
4     static volatile int brojac = 0;
5 }
6
```

Korištenje transient i volatile varijabli.

Korištenje tipa *enum*

```
1  public class Test {
2      enum GodisnjaDoba { ZIMA, PROLJECE, LJETO, JESEN }
3
4      public static void main(String[] args) {
5          for (GodisnjaDoba s : GodisnjaDoba.values())
6              System.out.println(s);
7      }
8  }
9
10 enum Operation {
11 //unutarnje klase nadjacavaju apstraktnu funkciju rac enum-a
       Operation, moze se pozivati izvan klase Operation
12 PLUS { double rac(double x, double y) { return x + y; } },
     //konstante s tijelom klase (unutarnje klase)
13 MINUS { double rac(double x, double y) { return x - y; }},
14 PUTA {double rac(double x, double y) { return x * y; }},
15 PODIJELJENO {double rac(double x, double y){ return x / y;}
16     };
```

Tip enum.

Korištenje tipa *enum*

```
1                .
2                .
3     abstract double rac(double x, double y);
4
5     public static void main(String args[]) {
6         double x = Double.parseDouble(args[0]);
7         double y = Double.parseDouble(args[1]);
8         for (Operation op : Operation.values())
9             System.out.println(x + " " + op + " " + y +
10                                " = " + op.rac(x, y));
11    }
12 }
```

Tip enum.

Enum je posebna vrsta tipa ***Class*** koja ne smije biti `abstract`, `sealed`, `non-sealed` niti `final`. U tijelu tipa *enum* se mogu nalaziti **konstante**, **konstruktori**, **kod za inicijaliziranje statičkih i ne-statičkih elemenata članova**, **metode**. Enum konstante **mogu imati tijelo klase**.

Tip *enum*

Tip *enum* postaje implicitno `final` ukoliko njegova deklaracija **ne sadrži** enum konstante koje imaju tijelo klase. Ukoliko kod deklaracije tipa *enum* postoje konstante koje imaju tijelo klase, tada je tip implicitno `sealed`. Jedine klase koje smiju naslijediti *enum* su tada anonimne klase definirane enum konstantama.

Ugniježdeni element tipa *enum* je implicitno `static`.

Zadatak za zadaću

Zadatak 1.

Definirajte jednu hijerarhiju generičkih (parametriziranih) klasa. Definirajte pripadna sučelja od kojih barem 2 trebaju imati *default* funkcije. Koristite tip presjeka da bi definirali tip koji sadrži razne kombinacije funkcija iz definiranih sučelja. Funkcionalnost demonstrirajte u glavnom programu koji se nalazi u različitom paketu od definirane hijerarhije klasa i sučelja.