3 Uvod u klase

3.1 Naša prva klasa: Knjiga

U ovom poglavlju ćemo ilustrovati elementarne radnje sa objektima klasnog tipa. U tom smislu, kreiraćemo klasu `Knjiga`, koja za članove podatke ima string `ime` i celobrojnu promenljivu `brojStrana`. Da bismo testirali rad sa objektima klase `Knjiga`, kreiraćemo dodatnu klasu `KnjigaTest`. Ove dve klase su date u nastavku.

U knjizi ćemo definicije klasa iz istog projekta razdvajati isprekidanom crtom.

```
// Deklaracija klase Knjiga

public class Knjiga {
  // Podaci klase
  private String ime;
  private int brojStrana;

  // Metode klase
  // Metoda getIme vraća ime knjige
  public String getIme() {
    return ime;
  }

  // Metoda setIme postavlja ime knjige
  public void setIme(String imeKnjige) {
    ime = imeKnjige;
  }

  // Metoda getBrojStrana vraća broj strana knjige
  public int getBrojStrana() {
    return brojStrana;
  }

  // Metoda setBrojStrana postavlja broj strana knjige
  public void setBrojStrana(int brStr) {
    brojStrana = brStr;
  }

  // Metoda prikaziKnjigu štampa ime knjige i broj strana
  public void prikaziKnjigu() {
```

```
    System.out.printf("Ime knjige je %s i ima %d strana.\n",
                      getIme(), getBrojStrana());
  }
}
```

```
// Deklaracija klase KnjigaTest

import java.util.Scanner;

public class KnjigaTest {

  // Klasa koja testira klasu Knjiga

  public static void main(String[] args) {
    Scanner unos = new Scanner(System.in);
    // Kreiranje objekta (instance) klase Knjiga
    Knjiga knjiga = new Knjiga();

    // Prikaz podrazumevanog imena knjige i broja strana
    System.out.println("Podrazumevano ime knjige i broj strana: ");
    knjiga.prikaziKnjigu();

    // Unos imena knjige i broja strana
    System.out.println("Unesite ime knjige i broj strana: ");
    String s = unos.nextLine();   // Čitanje linije teksta
    int n = unos.nextInt();       // Unos broja strana
    knjiga.setIme(s);             // Postavljanje imena knjige
    knjiga.setBrojStrana(n);      // Postavljanje broja strana knjige
    knjiga.prikaziKnjigu();
  }
}
```

```
Podrazumevano ime knjige i broj strana:
Ime knjige je null i ima 0 strana.
Unesite ime knjige i broj strana:
Robinzon Kruso
234
Ime knjige je Robinzon Kruso i ima 234 strana.
```

Prisetimo se da svaka klasa koja započinje ključnom rečju `public` mora biti smeštena u posebnom fajlu koji ima isto ime kao i klasa, i ekstenziju `java`. U našem primeru to znači da klase `Knjiga` i `KnjigaTest` moraju biti smeštene u fajlovima `Knjiga.java` i `KnjigaTest.java`, respektivno.

promenljiva izlazi iz opsega, pa se štampanje opet odnosi na podatak klase. U metodi `treca`, parametar metode zasenjuje podatak klase.

U Java metodi se, za razliku od C i C++, ne mogu deklarisati dve istoimene promenljive u ugnježdenim blokovima.

U skladu sa prethodnom napomenom, pokušaj deklaracije promenljive `x` u metodi `druga` bi doveo do greške jer ta promenljiva već postoji u `for` petlji.

3.10.6 Preklapanje metoda

Preklapanje metoda (eng. *method overloading*) je situacija kada više metoda jedne klase ima isto ime. Ovo je dozvoljeno, što smo videli u slučaju klase `Knjiga` testiranoj u klasi `KnjigaTest2` (poglavlje 3.9), kada smo imali preklopljene konstruktore. Metode se mogu preklapati pod uslovom da imaju različitu listu parametara. Listu parametara određuje broj, tip i redosled parametara. Pri pozivu preklopljene metode, kompajler, na osnovu broja, tipova i redosleda prosleđenih argumenata, bira odgovarajuću metodu. Preklapanje metoda se obično koristi za kreiranje metoda koje obavljaju iste ili slične zadatke, ali sa različitim argumentima. Na primer, metoda `abs` iz klase `Math` je preklopljena u četiri verzije:

```
public static int abs(int a)
public static long abs(long a)
public static float abs(float a)
public static double abs(double a)
```

Kombinacija imena metode sa brojem, tipom i redosledom parametara predstavlja *potpis metode* (eng. *method's signature*). Kompajler pravi razliku između preklopljenih metoda na osnovu potpisa metoda, jer se samo na osnovu imena ne može napraviti razlika. Interno, kompajler koristi duža imena metoda koja uključuju ime metode, tip i redosled parametara. Ako se ovakva produžena imena u potpunosti slažu, kompajler zaključuje da je došlo do konflikta pri preklapanju. Zato je važan i redosled parametara. U tom smislu, metode čiji su potpisi

```
int fun(int a, double b)
int fun(double a, int b)
```

se međusobno razlikuju i dozvoljene su sa stanovišta preklapanja.

Pri odlučivanju da li je došlo do preklapanja, ne posmatra se tip vraćene vrednosti, tj. preklopljene metode se ne mogu razlikovati na osnovu tipa vraćene vrednosti. U tom smislu, deklaracija metoda

```
int fun(int a, double b)
double fun(int a, double b)
```

nije dozvoljena.

Kako JVM razrešava preklapanje metoda? Posmatrajmo sledeće preklapanje metoda:

```
int fun(int a, double b)
int fun(double a, int b)
```

Poziv `fun(3, 4)` bi predstavljao sintaksnu grešku jer JVM ne može da odredi koju od metoda da pozove. Pošto nema potpunog poklapanja tipova argumenata sa parametrima, gleda se da li postoji parametar koji je istog tipa za sve metode. Ukoliko takav ne postoji, dolazi do greške. U prethodnom slučaju, prvi i drugi parametar metoda su različitog tipa, te dolazi do greške. Ukoliko postoji parametar koji je istog tipa za sve metode, on se eliminiše iz razmatranja i posmatraju se ostali parametri kako bi se razrešilo koju metodu treba zvati. Ukoliko nema potpunog poklapanja sa ostalim parametrima, procedura se ponavlja, tj. gleda se da li postoji parametar koji je istog tipa za sve metode itd.

Ilustrujmo ovo na sledećem primeru:

```
int fun(int a, double b, double c)
int fun(int a, int b, double c)
int fun(double a, int b, double c)
```

Poziv `fun(3, 4, 5)` bi se odnosio na drugu metodu (`int-int-double`). Pošto je treći parametar istog tipa kod sve tri metode, `double`, on se eliminiše iz razmatranja i posmatraju se ostala dva parametra. Kod druge metode, ostala dva se potpuno poklapaju sa tipom argumenata (dva `int`-a) i time je razrešen poziv preklopljene metode.

Sa druge strane, u slučaju metoda

```
int fun(int a, double b, double c)
int fun(int a, int b, double c)
int fun(double a, double b, int c)
```

poziv `fun(3, 4, 5)` dovodi do greške, jer ne postoji parametar istog tipa za sve tri metode.

3.11 Ključna reč `this`

Svi objekti jedne klase dele istu kopiju metoda klase. Sa druge strane, svaki objekat ima svoju kopiju podataka, isključujući statičke podatke koji su jedinstveni za čitavu klasu. Ključna reč `this`, pomoću koje objekat može referencirati samog sebe, pruža efikasan način da metoda zna koji tačno objekat je pozvao metodu, tj. sa čijim podacima da manipuliše. Naziva se još i referenca `this`. Kada se pozove nestatička metoda za određeni objekat, telo metode implicitno koristi referencu `this` za pristup promenljivama objekta i drugim metodama klase.

3.11.1 `this` i zasenjivanje podataka klase

Ključna reč `this` može se koristiti na više načina. Prvi način je za pristup podacima objekta koji su zasenjeni lokalnim promenljivama ili parametrima metode. U tom smislu,

Iako se zasenjenim podacima klase može pristupiti pomoću reference `this`, treba izbegavati zasenjivanje podataka jer može produkovati teško uočljive greške.

3.11.2 `this` i preklapanje konstruktora

Ovde ćemo kreirati klasu `Tacka` sa preklopljenim konstruktorima, metodama `set` i `get`, metodom za postavljanje koordinate tačke, metodama za računanje rastojanja tekuće od druge tačke, kao i metodom `toString`.

```
public class Tacka {

  // Potpuna implementacija klase Tacka

  private double x, y;

  public Tacka() {
    this(0, 0);
  }

  public Tacka(double x) {
    this(x, 0);
  }

  public Tacka(Tacka t) {
    this(t.getX(), t.getY());
  }

  public Tacka(double x, double y) {
    postaviTacku(x, y);
  }

  public void postaviTacku(double x, double y) {
    setX(x);
    setY(y);
  }

  public void postaviTacku(Tacka t) {
    this.postaviTacku(t.getX(), t.getY());
  }

  public double getX() {
    return x;
  }

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

  public double getY() {
```

```
    return y;
  }

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

  public double rastojanje(Tacka t) {
    return Math.sqrt((getX() - t.getX()) * (getX() - t.getX()) +
                     (getY() - t.getY()) * (getY() - t.getY()));
  }

  public String toString() {
    return String.format("(%.2f,%.2f)", getX(), getY());
  }
}
```

```
public class TackaTest {

  // Klasa koja testira klasu Tacka

  public static void main(String[] args) {
    Tacka t1 = new Tacka();
    Tacka t2 = new Tacka(3.4);
    Tacka t3 = new Tacka(2.3, -1.17);
    Tacka t4 = new Tacka(t3);

    System.out.printf("Prva tačka %s\n", t1.toString());
    System.out.printf("Druga tačka %s\n", t2.toString());
    System.out.printf("Treća tačka %s\n", t3.toString());
    System.out.printf("Četvrta tačka %s\n", t4.toString());

    t1.postaviTacku(4.5, -2.1);
    t2.postaviTacku(t3);
    t3.setX(11.59);
    t4.setY(-5.6);

    System.out.println("Prva tačka nakon izmene " + t1);
    System.out.println("Druga tačka nakon izmene " + t2);
    System.out.println("Treća tačka nakon izmene " + t3);
    System.out.println("Četvrta tačka nakon izmene " + t4);

    System.out.printf("Rastojanje prve i druge tačke %.2f\n",
                      t1.rastojanje(t2));
  }
}
```

```
Prva tačka (0.00,0.00)
Druga tačka (3.40,0.00)
```