Uvod u Java programski jezik

Ana Vignjević
Marko Čupić

Uvod u Java programski jezik

by Ana Vignjević and Marko Čupić

Abstract

Ovaj dokument namijenjen je polaznicima Java - tečaja i služi kao popratni materijal.

Table of Contents

List of Figures

List of Examples

Chapter 1. Uvod

Java programski jezik danas je jedan od najraširenijih programskih jezika, i spada u grupu objektno orijentiranih programskih jezika. Java osim samog programskog jezika ujedno definira i cijelo okruženje u kojem se programi izvode, nudeći pri tome vrlo velik broj gotovih biblioteka i funkcija. Zahvaljujući tome, pisanje programa može ići vrlo brzo i efikasno. Osim toga, Java programi ne prevode se direktno u instrukcije procesora na kojem se izvršava razvojna okolina, već u poseban međuformat poznat kao *byte-code*. Na svim popularnijim operacijskim sustavima danas postoje Java virtualni strojevi (engl. Java virtual machine) koji izvršavaju generirani byte-code. Programi pisani u Javi stoga se mogu smatrati potpuno platformski nezavisnima, i jednako se mogu izvršavati na primjerice Microsoft Windows te Linux operacijskim sustavima.

Struktura Java programa

Osnovna struktura Java programa prikazana je u primjeru the section called “Struktura Java programa” [1]

Example 1.1. Hello World program (datoteka HelloWorld.java)

```
package hr.fer.zemris.java.tecaj_0;
/**
 * @author Ana Vignjević
 * @version 1.0
 */
public class HelloWorld {

 /**
  * Metoda koja se poziva prilikom pokretanja programa.
  * Argumenti su objašnjeni u nastavku.
  * @param args Argumenti iz komandne linije.
  */
 public static void main(String[] args) {
        System.out.println("Hello World!");
 }

}
```

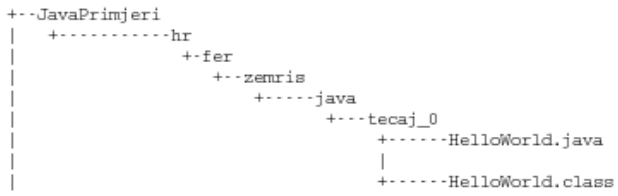
Primjer definira razred `HelloWorld` u kojem se nalazi metoda `main` (izvođenje programa započinje pozivom ove metode). Kako bi se smanjila kompleksnost koda, Java programski jezik uvodi pojam paketa kao način grupiranja više sličnih razreda. U liniji 1 nalazi se definicija paketa (ključna riječ *package*). Paketi predstavljaju hijerarhijsko grupiranje razreda, te i sami mogu sadržavati podpakete. U našem slučaju, razred `HelloWorld` smjestili smo u paket hr.fer.zemris.java.tecaj_0. Pri tome treba voditi računa da hijerarhijsko ime paketa mora odgovarati strukturi direktorija na disku, što znači da će datoteka HelloWorld.java biti smještena u direktorij hr/fer/zemris/java/tecaj_0.

Pretpostavimo da ćemo sve primjere spremati u direktorij C:\JavaPrimjeri. HelloWorld.java datoteku prevodimo tako da se pozicioniramo u direktorij JavaPrimjeri i izvršimo sljedeću naredbu:

```
JavaPrimjeri> javac hr/fer/zemris/java/tecaj_0/HelloWorld.java
```

Rezultat gornje naredbe je nova datoteka HelloWorld.class koja sadrži izvršivi kod našeg programa i nalazi se u istom direktoriju kao i izvorna .java datoteka.

Figure 1.1. Spremanje programa i izvršivog koda u isti direktorij



Program pokrećemo naredbom **java** nakon koje navedemo puno ime razreda bez ekstenzije .class:

```
JavaPrimjeri> java -cp .hr.fer.zemris.java.tecaj_0.HelloWorld

Hello World!
```

Na nekim sustavima Java je po defaultu konfigurirana tako da izvršive datoteke (datoteke s ekstenzijom .class) traži u trenutnom direktoriju. U tom slučaju pokretanje Java programa može se ostvariti i bez prekidača *-cp .*, dakle ovako:

```
JavaPrimjeri> java hr.fer.zemris.java.tecaj_0.HelloWorld
```

Ukoliko odlučimo odijeliti izvorni program od izvršivog koda tada prevođenje i izvođenje programa, te struktura direktorija izgleda ovako:

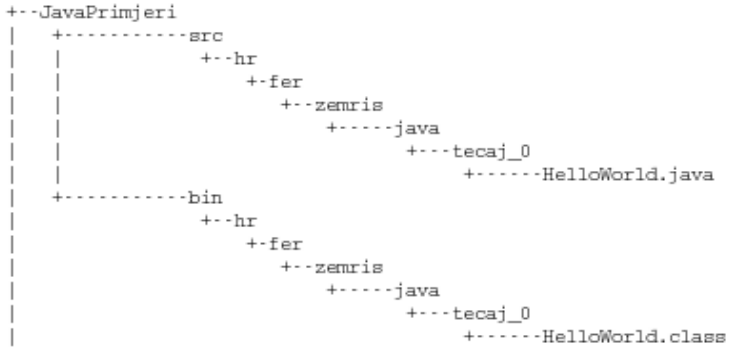
```
JavaPrimjeri> javac -d bin -sourcepath src src/hr/fer/zemris/java/tecaj_0/HelloWo
```

To će smjestiti izvršivi kod u direktorij bin sa pripadnom strukturom direktorija. Program se izvodi sljedećom naredbom:

```
JavaPrimjeri> java -cp bin hr.fer.zemris.java.tecaj_0.HelloWorld
```

Struktura direktorija za gore navedeni primjer izgleda ovako:

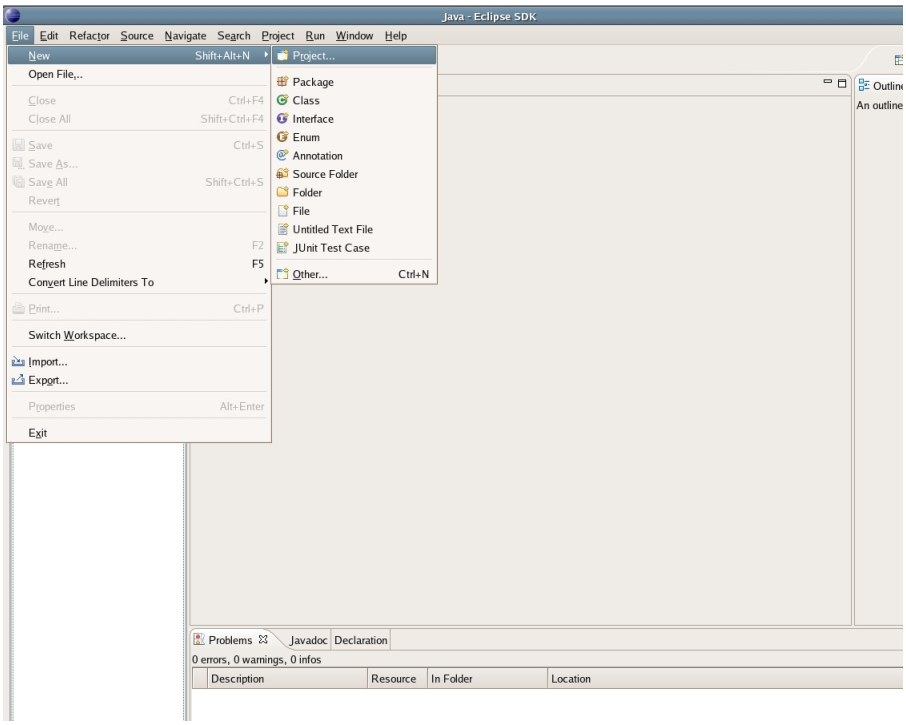
Figure 1.2. Spremanje programa i izvršivog koda u odvojene direktorije



Razvoj Java programa u razvojnom okruženju Eclipse

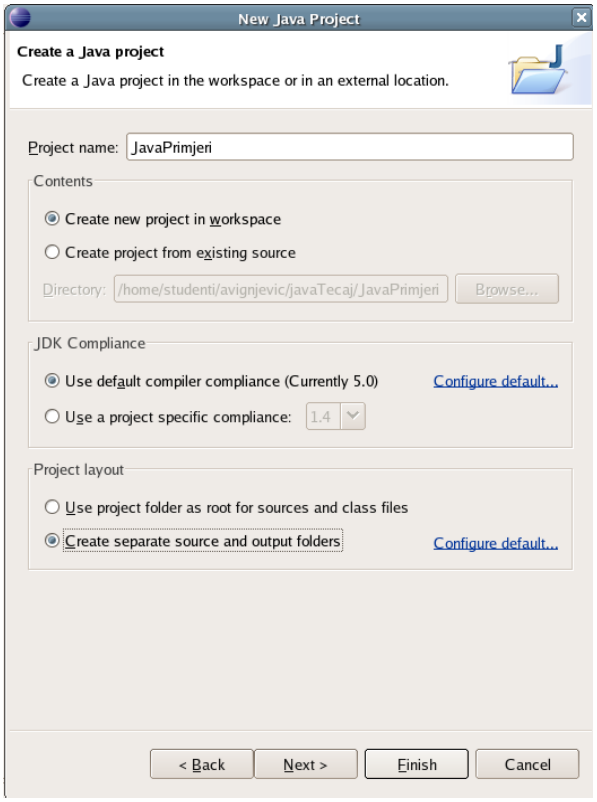
Razvojno okruženje Eclipse i detaljne upute za instalaciju i pokretanje moge se pronaći sa sljedećoj adresi: http://www.eclipse.org/ Slijedi opis procedure stvaranja i pokretanja HelloWorld programa pomoću Eclipsa. Prvo stvaramo novi projekt, odabravši u izborniku `File->New->Project` (Figure 1.3, "File->New->Project")

Figure 1.3. File->New->Project



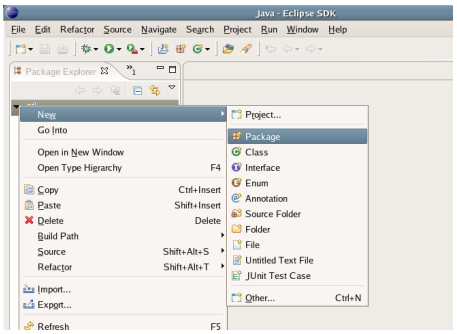
U prozoru nakon toga potrebno je odabrati opciju `Java Project` i nakon toga `next`. U dijalogu `New Java Project` (Figure 1.4, "Definiranje detalja novog projekta") treba unijeti ime projekta (JavaPrimjeri), odabrati opciju `Create new project in workspace` i `Create separate source and output folders`.

Figure 1.4. Definiranje detalja novog projekta



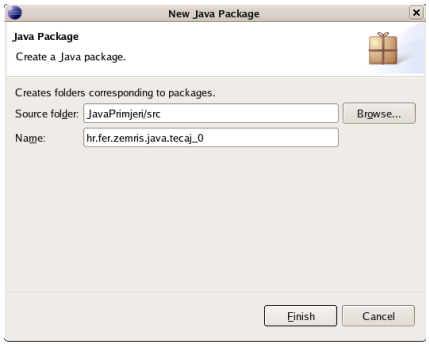
Nakon što smo definirali svoj projekt, deklariramo novi paket unutar njega, u kojem će se nalaziti svi naši razredi. Desnim klikom na naš projekt u izborniku odabiremo `New->Package` (Figure 1.5, “New->Package”)

Figure 1.5. New->Package



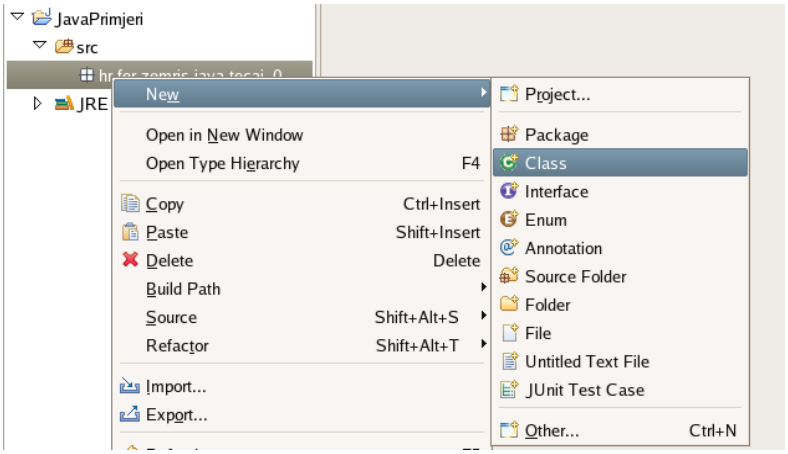
Slijedi imenovanje paketa `hr.fer.zemris.java.tecaj_0` (Figure 1.6, “Definiranje detalja novog paketa”) te potom `finish`.

Figure 1.6. Definiranje detalja novog paketa



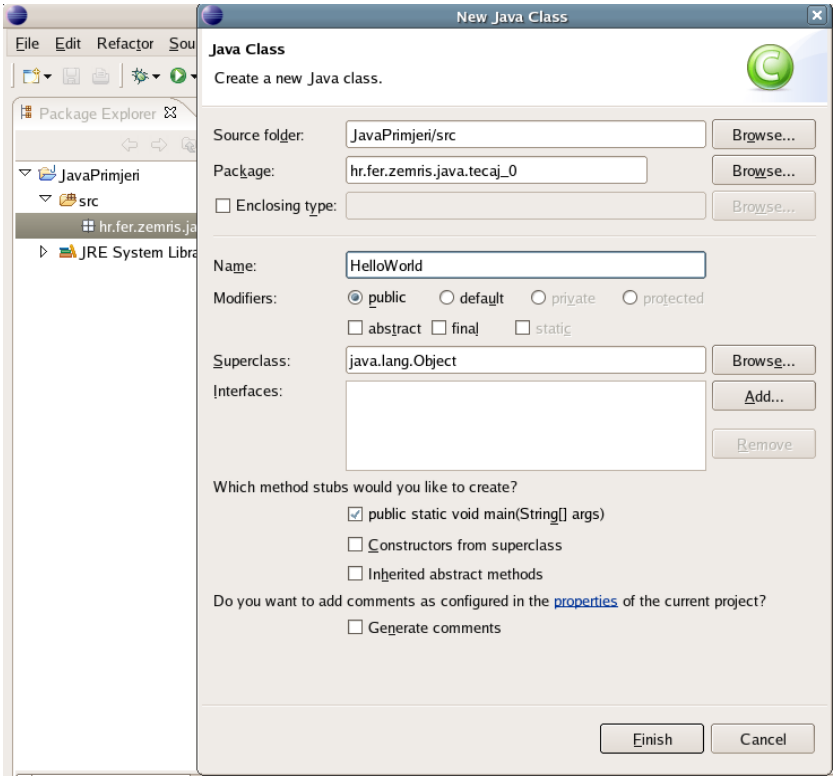
Nakon što je okolina za rad pripremljena, stvaramo svoju razred; desnim klikom na naš projekt, te zatim `New->Class`(Figure 1.7, “Novi razred”),

Figure 1.7. Novi razred



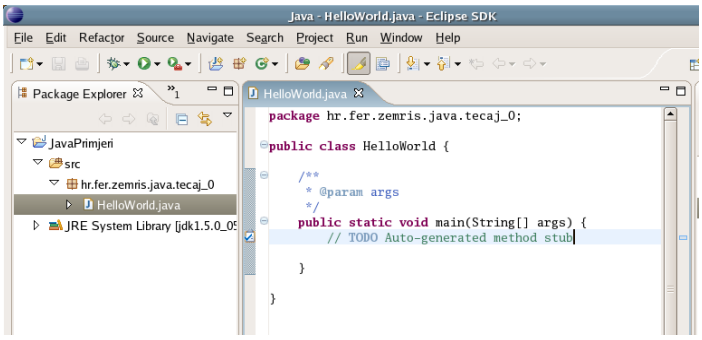
čime smo dobili novi izbornik u kojem definiramo sve potrebno za naš HelloWorld program (Figure 1.8, “Definiranje detalja HelloWorld razreda”).

Figure 1.8. Definiranje detalja HelloWorld razreda



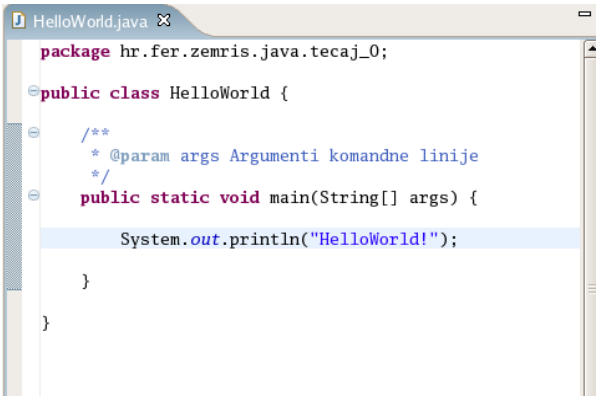
Time je kreirana naš razerd sa već postojećom main metodom, koju sada samo treba napisati.(Figure 1.9, “HelloWorld.java”)

Figure 1.9. HelloWorld.java



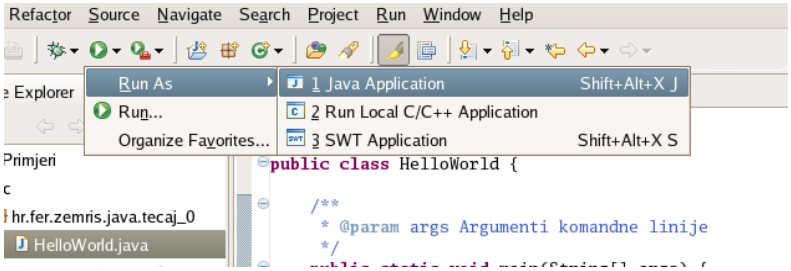
Na Figure 1.10, “Tijelo main() metode” možemo vidjeti napisani `void main()`. Iznad main metode se nalazi komentar pomoću kojeg ćemo kasnije moći generirati dokumentaciju (izbornik `Project->Generate Javadoc`).

Figure 1.10. Tijelo main() metode



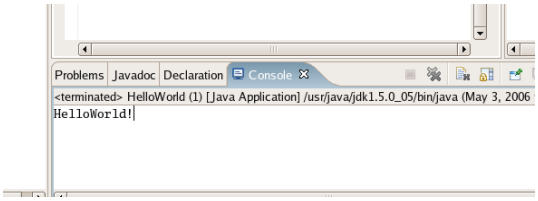
Još nam samo ostaje pokretanje HelloWorld programa, a to činimo na sljedeći način: Ispod `Source` izbornika odaberemo zeleni krug sa strelicom, te u dobivenom izborniku odaberemo `Run As->1 Java Application` (Figure 1.11, “Pokretanje Hello World programa”).

Figure 1.11. Pokretanje Hello World programa



Na sljedećoj slici Figure 1.12, “Rezultat izvođenja Hello World programa” se vidi ispis naše poruke u konzoli.

Figure 1.12. Rezultat izvođenja Hello World programa



Struktura Java datoteke

Svo programiranje u Javi se odvija u razredima. Ime razreda je ujedno i ime programa. Svaka Java datoteka sastoji se od tri dijela:

1.) Deklaracija paketa kojem razred priprada (`package`)

```
package hr.fer.zemris.java.tecaj_0;
```

pri čemu ime paketa predstavlja hijerarhijski prikaz strukture direktorija u kojem se nalazi naša datoteka.

2.)opcionalni popis paketa koje treba uključiti (*import*). U HelloWorld datoteci nije bilo potrebe za uključivanjem drugih razreda, no u primjeru sa ispisom decimalnih brojeva trebalo je uključiti razred `DecimalFormat`:

```
import java.text.DecimalFormat;
```

3.)deklaracija samog razreda

```
public class HelloWorld {
...
}
```

Java programski jezik podržava gniježđenje, što znači da unutar jednog razreda možemo deklarirati druge razrede.

```
package hr.fer.zemris.java.tecaj_0;

public class JavniRazred {
      ...
      public static class NekiDrugiRazred {
           ...
  }
}
        }
```

Također, unutar jedne Java datoteke možemo deklarirati proizvoljan broj razreda, no treba voditi računa da samo jedan može biti javan, a njegovo ime mora biti isto onome kojeg nosi sama datoteka.

```
package hr.fer.zemris.java.tecaj_0;

public class JavniRazred {
     ...
}
class NekiDrugiRazred {
     ...
}
```

Da bi se neki program mogao izvršiti on mora sadržavati metodu main(). U slučaju Hello World programa, main metoda je deklarirana na sljedeći način:

```
   public static void main(String[] args) {
      ...
   }
```

Deklaracija metode sadrži tri modifikatora koji će kasnije biti detaljnije objašnjeni:

• public - primjer jednog javnog razreda

• static - metoda pripada samom razredu, te nije potrebno stvariti instancu tog rezreda da bi se mogla koristiti

• void - metoda ne vraća ništa

Inačica Hello World programa za programski jezik C bi bila:

```
void main(int argc, char *args[]) {
    printf("Hello World!\n");
}
```

Komentari

Komentiranje koda je iznimno važno. Detaljni i jasni komentari olakšavaju praćenje toka programa te lociranje potencijalnih problema, kao i brži pristup onim dijelovima koje želimo modificirati ili optimizirati. Java stavlja poseban naglasak na komentare, te definira dvije vrste:

običan komentar:

// ovo je komentar

/* ovo je komentar */
Komentari koji započinju sa "//" su jednolinijski komentari, dok se "/* ...*/" mogu protezati kroz proizvoljan broj redaka, te su idealni za npr. opis metoda, detaljno komentiranje kompleksnijih dijelova koda itd.

javadoc komentar:

/** ovo je javadoc komentar */
Ovo su također višelinijski komentari, no oni se koriste pri generiranju dokumentacije.

Obični komentari se ne razlikuju od onih u C programskom jeziku, dok javadoc komentari omogućavaju programeru generiranje strukturirane dokumentacije pomoću posebnih oznaka sljedećeg oblika:
@naziv vrijednost:

@author ime_autora, npr. - @author Marko Čupić

@version verzija_razreda, npr. - @version 1.0

@param ime_argumenta opis, npr. - @param x broj čiji sinus treba izračunati

@return opis, npr. - @return vraća sinus zadanog broja

Komentari se koriste za opis metoda, ali i samih razreda (formalni komentari - npr. autor, verzija). Dokumentacija se generira odabirom izbornika **Project->Generate Javadoc**. Rezultat je lijepo strukturiran i jasan pregled opisa svih komentiranih metoda - što metode primaju kao argumente, uvjeti koji moraju biti zadovoljeni da bi metoda radila ispravno, što točno metode rade, te koji je rezultat njihovog izvođenja. Primjer jasnog i razumljivog komentara:

```
/**
* Metoda računa y-tu potenciju od broja x.
* @param x argument x
```

```
   * @param y argument y; mora biti nenegativan
   * @return vraća iznos izraza x^y
   */
   public static double pow(int x, int y) {
     ...
   }
```

Nekoliko jednostavnih primjera

Example 1.2. Ispis argumenata iz predanih preko komandne linije

Napisati program koji će na zaslon ispisati argumente koje dobiva prilikom pokretanja programa.

```
package hr.fer.zemris.java.tecaj_1;

/**
 * @author Marko Čupić
 * @version 1.0
 */
public class IspisArgumenata {

 /**
  * Metoda koja se poziva prilikom pokretanja
  * programa. Argumenti su objasnjeni u nastavku.
  * @param args Argumenti iz komandne linije.
  */
 public static void main(String[] args) {
  int brojArgumenata = args.length;

 for(int i = 0; i < brojArgumenata; i++) {
 System.out.println(
 "Argument " + (i+1) + ": " + args[i]
 );
 }
 }
}
```

Opis rješenja:

Metoda prima polje Stringova preko komandne linije (String je poseban tip podataka u Java programskom jeziku, nad kojim je definiran bogat skup funkcija za ispis, pretraživanje i spajanje). Polja kao objekti, imaju pripadnu varijablu *`length`* koja nam daje do znanja upravo duljinu tog polja. Također smo prije dodjeljivanja vrijednosti duljine polja u lokalnu varijablu *`brojArgumenata`* mogli provjeriti da li je polje uopće predano, te u skladu sa rezultatom upita eventualno zavrsiti izvođenje programa:

```
if(args==null || args.length==0) {
     return;
}
```

Ulazimo u for petlju te se na zaslon redom ispisuju predani argumenti u sljedećem formatu: "Argument (redni_broj_argumenta): vrijednost_argumenta". Primjetimo da poziv **System.out.println()** kao argument očekuje varijablu tipa String, a predane su varijable tipa String i int. U Javi se operator "+" koristi u operacijama sa Stringovima kao spajanje, a sam ispis je podržan za sve tipove varijabli, bez eksplicitnog navođenja formata ispisa (dok u je C programskom jeziko za polja znakova bilo nužno navesti "%s" za znakovne nizove ili "%d" za cjelobrojne tipove). Također, metoda i razred su prikladno komentirani.

Example 1.3. Računanje sume reda

Primjer: Napisati program koji će prilikom pokretanja primiti jedan argument (x), te izračunati koliko iznosi e^x razvojem u Taylorov red. Razvoj riješiti u zasebnoj funkciji. Program na zaslon mora ispisati rezultat.

Skica rješenja: Razred koji treba napraviti će sadržavati main() metodu, pomoću koje ćemo pokretati program, i posebnu metodu koja će računati e^x za parametar x kojeg joj treba predati kao argument. U Javi prevođenje programa nije uvjetovano redoslijedom definiranja metoda koje koristimo, što znači da metoda za računanje e^x može biti napisana ispred, ali i iza metode main() u kojoj se poziva. Također, deklaracija lokalnih variabli je moguća u bilo kojem trenutku, i bilo kojem mjestu u kodu, jedino treba voditi računa da onaj raspon unutar kojeg se varijabla deklarira uvjetuje njen ivotni vijek. Pokazalo se kao dobra praksa deklaraciju lokalnih varijabli vršiti upravo na onom mjestu gdje te varijable počinjemo koristiti, prvenstveno da bi se eliminirale moguće greške. Da bismo protumačili String kao broj koristimo metodu razreda Double koja kao argument prima String, te ga parsira u broj ukoliko je to moguće (možda uopće nije predan String koji sadržava broj), a vraća vrijednost protumačenog broja.

```
package hr.fer.zemris.java.tecaj_1;

/**
 * @author Marko Čupić
 * @version 1.0
 */
public class SumaReda {

 /**
  * Metoda koja se poziva prilikom pokretanja
  * programa. Argumenti su objašnjeni u nastavku.
  * @param args Argumenti iz komandne linije.
  */
 public static void main(String[] args) {

 if(args.length != 1) {
 System.err.println(
 "Program mora imati jedan argument!"
 );
 System.exit(1);
 }

 double broj = Double.parseDouble(args[0]);

 System.out.println("Računam sumu...");

 double suma = racunajSumu(broj);

 System.out.println("f(" + broj + ")=" + suma + ",");
 }



 /**
  * Računa e^x razvojem u Taylorov red, prema formuli:
  * e^x=1+x+(x^2/(2!))+(x^3/(3!))+(x^4/(4!))+...
  * @param broj argument funkcije e^x
  * @return iznos funkcije u točki x=broj dobiven kao
  *       suma prvih 10 clanova Taylorovog reda.
  */
 private static double racunajSumu(double broj) {
 double suma = 0.0;
 double potencija = 1.0;
 double faktorijela = 1.0;

 suma += 1.0;

 for(int i = 1; i < 10; i++) {
 potencija = potencija * broj;
 faktorijela = faktorijela * i;
 suma += potencija/faktorijela;
 }
 return suma;
 }
}
```

Example 1.4. Ispis decimalnih brojeva

Primjer: Napisati program koji će sadržavati funkciju koja prima polje double-ova, te ih ispisuje na zaslon po zadanom formatu. Napisati glavni program koji će ispisati brojeve:

- Najmanje tri mjesta za cijelobrojni dio, dva mjesta za decimalni
- Dva + dva mjesta s obaveznim ispisom predznaka

Skica rješenja:

Razred koji treba napraviti će sadržavati main() metodu, pomoću koje ćemo pokretati program, i posebnu metodu za ispis pbrojeva prema zadanom formatu. U ovom primjeru smo u našem razredu koristili i jedan drugi razred (`java.text.DecimalFormat`) kojeg smo uključili sa ključnom riječi **import** ispred punog imena tog razreda, a prije deklaracije našeg razreda. Format u kojem je potrebno ispisati brojeve se definira kao String; u prvom slučaju "000.00", što će rezultirati ispisom brojeva na (najmanje) tri mjesta za cjelobrojni, i dva mjesta za decimalni dio; u drugom "+00.00;-00.00" što će se interpretirati kao ispis na (najmanje) dva cjelobrojna, i dva decimalna mjesta, sa obaveznim ispisom predznaka. Prvi dio formata, prije separatora ";" definira znak ispisa za pozitivne, a drugi za negativne brojeve.

```
package hr.fer.zemris.java.tecaj_1;

import java.text.DecimalFormat;

/**
 * @author Marko Čupić
 * @version 1.0
 */
public class FormatiraniIspisDecBrojeva {

 /**
  * Metoda na standardni izlaz ispisuje polje decimalnih
  * brojeva prema zadanom formatu.
  * @param polje polje decimalnih brojeva koje treba ispisati.
  * @param format format koji govori kako polje treba ispisati.
  * @see DecimalFormat
  */
 public static void ispis(double[] polje, String format) {
 DecimalFormat formatter = new DecimalFormat( format );
 for(int i = 0; i < polje.length; i++) {

System.out.println("(" + i + "): [" + formatter.format(polje[i]) + "]" );

 }
    }
```

```
 /**
  * Metoda koja se poziva prilikom pokretanja
  * programa. Argumenti su objasnjeni u nastavku.
  * @param args Argumenti iz komandne linije.
  */
 public static void main(String[] args) {
 double[] brojevi = new double[] {3.712, 55.813, 55.816, -4.18};
 ispis(brojevi, "000.00");
 ispis(brojevi, "+00.00;-00.00");
 }
}
```

Example 1.5. Primjer čitanja s tipkovnice

Primjer: Napisati program koji će s tipkovnice čitati decimalni broj po broj i računati njihovu sumu, sve dok se upisuju nenegativni brojevi.

Skica rješenja: čitanje s tipkovnice je nešto kompliciranje, pa sljedeći kod može poslužiti kao šablona. Koristimo razrede `BufferedReader` i `InputStreamReader` iz paketa java.io (input/output) koje smo uključili ključnom riječi **import**. Instanca InputStreamReader-a prima argument *`System.in`*, a nju onda predajemo kao argument instanci BufferedReader-a jer upravo taj razred ima definirane one metode koje su nam potrebne za rješavanje ovog zadatka (metoda readLine()). U lokalnu varijablu "redak" spremamo ono sto je metoda `readLine()` dohvatila sa tipkovnice, te taj String tumačimo kao broj sa već spomenutom metodom `parseDouble` razreda `Double`. Ostatak rješenja je trivijalan. Nakon što smo završili sa radom, potrebno je pozvati metodu close() nad reader.om koja će kaskadno zatvoriti i instancu InputStreamReader-a. Instanca nekog razreda je deklalirani primjerak tog razreda, tj. objekt. Deklarirali smo primjerak BufferedReader-a sa sljedećom linijom koda (pozivanjem konstruktora):

```
BufferedReader reader = new BufferedReader(
       new InputStreamReader(System.in)
     );
```

baš kao što deklariramo i lokalne varijable, npr.:

```
String linija = new String(); // ili = "ovo je neki string";
...
int i = 0;
```

```
package hr.fer.zemris.java.tecaj_1;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * @author Marko Čupić
 * @version 1.0
 */
public class CitanjeSTipkovnice {

  /**
   * Metoda koja se poziva prilikom pokretanja
   * programa. Argumenti su objašnjeni u nastavku.
   * @param args Argumenti iz komandne linije.
   */
  public static void main(String[] args) throws IOException {
    System.out.println("Program za računanje sume pozitivnih brojeva.");

    System.out.println("Unosite brojeve, jedan po retku.");
    System.out.println("Kada unesete negativan broj, ispisat će se suma.");


    BufferedReader reader = new BufferedReader(
             new InputStreamReader(System.in)
     );

    double suma = 0.0;
    while(true) {
      String redak = reader.readLine();
      if(redak==null) break;
      double broj = Double.parseDouble(redak);
      if(broj<0) break;
      suma += broj;
    }
    System.out.print("Suma je: ");
    System.out.println(suma);

    reader.close();
  }
}
```

Rad sa stringovima

U Javi String nije polje znakova terminirano s "\0". Pošto sam način pohrane Stringova u memoriji nije bitno za samo programiranje, korištenje ovog razreda je uveliko olakšano .

Example 1.6. Nekoliko načina ispisa Stringova

Primjer: Nekoliko načina ispisa stringova sa procjenom efikasnosti. Prvo slijedi razred koji će nam poslužiti za demonstraciju efikasnosti korištenja operatora "+" te drugih operacija za manipulaciju Stringovima.

```
package hr.fer.zemris.java.tecaj_1;

/**
 * @author Marko čupić
 * @version 1.0
 */
public class RadSaStringovima {

 /**
  * Metoda koja se poziva prilikom pokretanja
  * programa. Argumenti su objasnjeni u nastavku.
  * @param args Argumenti iz komandne linije.
  */
   public static void main(String[] args) {
    ispis1();
    ispis2();
    ispis3();
    ispis4();
   }
   private static void ispis1() {...}
   private static void ispis2() {...}
   private static void ispis3() {...}
   private static void ispis4() {...}
}
}
```

Example 1.7. Spajanje teksta operatorom "+" kroz nekoliko linija

Metoda `ispis1()` prvo stvara novi String te ga inicijalizira na null vrijednost. Zatim se "dodavanje" teksta vrši običnim pridjeljivanjem i to operatorom "=". Pridjeljivanje se može obaviti i sa konstrukcijom "+=" baš kao što smo to činili sa numeričim podacima. Ovaj pristup je iznimno neefikasan zato što će korištenjem operatora "+" biti interno realociran novi String da bi duljinom odgovarao nadodanom tekstu, a to se čini kroz nekoliko naredbi.

```
/**
 * Demonstracija zbrajanja stringova.
 * Zbrajanje uporabom operatora + kroz vise naredbi.
 * Vrlo neefikasno!
 */
private static void ispis1() {
String tekst = null;
tekst = "The quick " + "brown ";
tekst += "fox jumps over ";
tekst += 3;
tekst += " lazy dogs.";
System.out.println(tekst);
}
```

Example 1.8. Spajanje teksta operatorom "+" u jednoj naredbi

Metoda `ispis2()` prvo stvara novi String te ga inicijalizira na null vrijednost. Premda se ovdje operator "+" koristi samo dva puta i u samoj jednoj naredbi, ovaj pristup je i dalje dosta neefikasan.

```
/**
 * Demonstracija zbrajanja stringova.
 * Zbrajanje operatorom + u jednoj naredbi. Efikasnije.
 */
private static void ispis2() {
String tekst = null;
int broj = 3;
tekst = "The quick brown fox jumps over " +broj+ " lazy dogs.";
System.out.println(tekst);
}
```

Example 1.9. Spajanje teksta uporabom StringBuffer razreda

Metoda `ispis3()` koristi drugačiji pristup, koristeći razred `StringBuffer`. Stvara se spremnik neke inicijalne veličine, te u njega dodaje tekst, no pošto su i ovdje bile potrebne tri realokacije da bi se spremnik adekvatno proširio, pristup je jednake efikasnosti kao prethodni primjer.

```
/**
 * Demonstracija zbrajanja stringova.
 * Zbrajanje uporabom StringBuffer objekta. Jednako efikasno
 * kao i primjer 2. Naime, inicijalno se stvara spremnik
 * velicine 16 koji se tri puta realocira kako bi se prosirio.
 * Napomena: od Java 5.0 koristiti StringBuilder koji je bitno
 * brzi (ali se mora koristiti unutar jedne dretve).
 */
private static void ispis3() {
String tekst = null;
StringBuffer sb = new StringBuffer();
sb.append("The quick ").append("brown ");
sb.append("fox jumps over ").append(3);
sb.append(" lazy dogs.");
tekst = sb.toString();
System.out.println(tekst);
}
```

Example 1.10. Spajanje teksta uporabom StringBuffer razreda - sa procjenom količine teksta

Metoda `ispis4()` je najefikasnija od navedenih. Pošto možemo procijeniti količinu teksta kojeg želimo upisati, unaprijed se alocira spremnik odgovarajuće veličine.

```
/**
 * Demonstracija zbrajanja stringova.<br>
 * Zbrajanje uporabom StringBuffer objekta. Najefikasnije
 * ako unaprijed znamo potrebnu velicinu spremnika. U primjeru
 * se alocira spremnik velicine 50 znakova.
 * Napomena: od Java 5.0 koristiti StringBuilder koji je bitno
 * brzi (ali se mora koristiti unutar jedne dretve).
 */
private static void ispis4() {
String tekst = null;
StringBuffer sb = new StringBuffer(50);
sb.append("The quick ").append("brown ");
sb.append("fox jumps over ").append(3);
sb.append(" lazy dogs.");
tekst = sb.toString();
System.out.println(tekst);
}
```

Od Jave 5.0 češće se koristi razred StringBuilder jer je znatno brži, no njegovo ograničenje je to da se mora koristiti unutar jedne dretve.

Chapter 2. Razredi i objekti

Ponekad potrebe nekog određenog programskog zadatka premašuju raspoloživ skup tipova podataka. Za izradu kompleksnih tipova podataka, C nudi strukture.

Primjer strukture u C-u

Struktura koja predstavlja geometrijski lik prikazan je u sljedećem primjeru.

Example 2.1. Struktura u C-u

```
typedef struct pravokutnik_str {
 int poz_x;
 int poz_y;
 int sirina;
 int visina;
 char *ime;
} pravokutnik;
```

Svojstva ovog pristupa su sljedeća: Svatko može mijenjati vrijednosti članova strukture, bez ikakve provjere da li postoje ovlasti koje to mijenjanje dopuštaju. Također, ne postoji nikakav način kontrole postavljanja legalnih, tj. smislenih vrijednosti članova strukture. Jedno od mogućih rješenja bi bila zabrana direktnog korištenja članova strukture ili razvijanje posebnog skupa funkcija koje će se zatim koristiti pri daljenjem radu sa strukturom. Slijede primjeri funkcija za stvaranje jednog primjerka te strukture, njegovo uništavanje te postavljanje/mijenjanje njegovih vrijednosti uz kontrolu valjanosti predanih vrijednosti:

Example 2.2. Rad sa strukturama u C-u

Funkcija `pravokutnik_novi` će pokušati alocirati prostor za jedan primjerak strukture pravokutnik, te njegovim članovima pridijeliti dostupne vrijednosti.

```
pravokutnik *pravokutnik_novi(int x, int y, int s, int v, char *ime) {
 pravokutnik *novi = (pravokutnik*)malloc(sizeof(pravokutnik));
 if(novi == NULL) return NULL;
 novi->ime = (char*)malloc(strlen(ime)+1);
 if(novi->ime == NULL) return NULL;
 strcpy(novi->ime, ime);
 novi->poz_x = x;
 novi->poz_y = y;
 novi->sirina = s;
 novi->visina = v;
 return novi;
}
```

Funkcija `pravokutnik_unisti` će osloboditi memoriju koju je zauzimao predani primjerak strukture pravokutnik.

```
void pravokutnik_unisti(pravokutnik *p) {
 if(p == NULL) return;
 if(p->ime != NULL) free(p->ime);
 free(p);
 return;
}
```

Funkcija `pravokutnik_postavi_sirinu` će se pobrinuti da se članskoj varijabli sirina pridijeli samo smislena vrijednost, ili nikakva.

```
void pravokutnik_postavi_sirinu(pravokutnik *p, int s) {
 if(s < 1) return;
 p->sirina = s;
 return;
}
```

Funkcija `pravokutnik_dohvati_sirinu` služi za dohvat vrijednosti članske varijable koja čuva širinu.

```
int pravokutnik_dohvati_sirinu(pravokutnik *p) {
 return p->sirina;
}
```

Ukoliko limitiramo rad nad strukturom samo na gornje funkcije prvenstveno se smanjuje mogućnost pogreške u kodu (npr. funkcija za stvaranje napisana je samo na jednom mjestu). Također, članovi neće biti postavljani na nekonzistentne vrijednosti. No nemoguće je osigurati se da svi koriste navedene funkcije. Postoji još jedan problem: Kako postići specijalizaciju strukture, ili dodavanje novih polja toj strukturi?

Java razredi

Java programski jezik uvodi poopćene "C-strukture" - razrede (engl. class). Na elementarnom nivou, razred je struktura, koja osim članskih varijabli ima i vlastite funkcije ("metode"), te nudi kontrolu pristupa (tko može pristupiti čemu). Da bi se stvorio primjerak jednog razreda (objekt), poziva se posebna metoda koja se zove konstruktor (ekvivalent funkcije pravokutnik_novi). Za uništavanje objekta u Javi ne postoji posebna metoda jer za to nema potrebe - ukoliko ponestane memorije, bit će pozvan garbage collector koji će osloboditi svu memoriju koju su zauzimali oni objekti na koje više ne postoje reference. Prije no što objekt bude uništen, on je finaliziran. Slijedi primjer definiranja razreda GeometrijskiLik:

Example 2.3. Razred GeometrijskiLik

Ovaj razred daje osnovan opis bilo kojeg geometrijskog lika. Ima svoje ime, koje čuva u privatnoj varijabli koja je po tipu String, nad kojom odmah definira i javnu metodu za dohvat tog imena (getter, engl.). Također, definira i svoj konstruktor, čijim pozivanjem se stvara instanca tog razreda, tj. objekt. Ovaj razred definira i dvije metode, od kojih jedna služi za dohvat opsega, a druga za dohvat površine tog geometrijskog lika.

```
public class GeometrijskiLik {
 /** Privatni element koji pohranjuje ime lika */
 private String ime;

 /** Konstruktor geometrijskog lika */
 public GeometrijskiLik(String ime) {
 this.ime = ime;
 }

 /** Dohvat imena geometrijskog lika */
 public String getIme() {
 return this.ime;
 }

 /** Dohvat opsega geometrijskog lika */
 public double getOpseg() {
 return 0.0;
 }
 /** Dohvat površine geometrijskog lika */
 public double getPovrsina() {
 return 0.0;
 }
}
```

Primjer stvaranja primjerka tog razreda:

```
GeometrijskiLik lik1 = new GeometrijskiLik("Lik1");
```

Varijabla *lik1* je po vrsti referenca (slično kao pokazivač u C-u). Operator **new** alocira u memoriji mjesto za jedan primjerak razreda i zatim zove odgovarajući konstruktor koji će inicijalizirati objekt. Rezultat tih operacija je vraćanje reference na novi objekt. U Javi ne postoje pokazivači u onom smislu kako smo ih definirali u C-u, niti je korisniku dopušteno da sam odlučuje "gdje" će i "koliko" memorije zauzeti. Java sama vodi računa o zauzimanju i oslobađanju memorije, te je time aliminirana mogućnost grešaka koje su se znale događati kada je korisnik pokušao pisati po dijelovima memorije koji mu nisu bili na raspolaganju.

Primjeri uporabe razreda GometrijskiLik

Example 2.4. Uporaba razreda GeometrijskiLik

U main metodi sljedećeg razreda stvaraju se dva primjerka razreda GeometrijskiLik (lik1 i lik2), te se na zaslon ispisuju njihova imena pozivanjem metode `getIme()`.

```
class Primjer1 {
   public static void main(String[] args) {
       GeometrijskiLik lik1 = new GeometrijskiLik("Lik1");
       GeometrijskiLik lik2 = new GeometrijskiLik("Lik2");
       System.out.println("Ime prvog lika je "+lik1.getIme());
       System.out.println("Ime drugog lika je "+lik2.getIme());
   }
}
```

Ispis:

```
Ime prvog lika je Lik1
Ime drugog lika je Lik2
```

Što ako želimo definirati razred koji će opisivati liniju? Linija je također geometrijski lik, no da bi je definirali trebaju nam i koordinate početka i kraja. Ovaj razred ćemo definirati tako što ćemo reći da Linija nasljeđuje (extends) osobine i metode koje posjeduje razred GeometrijskiLik, te mu dodati one osnovne osobine da bi primjerak tog razreda točno definirao liniju.

Example 2.5. Implementacija razreda Linija kao posebne vrste geometrijskog lika:

Nakon što pri deklaraciji razreda Linija definiramo da se nasljeđuje rezred GeometrijskiLik, ovom razredu je automatski dodijeljena varijabla *ime* te metode `getOpseg()` i `getPovrsina()`. Pored toga, mi sami definiramo jos dva para varijabli koje će nam služiti za definiranje početka i kraja linije. U konstruktoru ovog razreda primamo četiri numeričke vrijednosti koje postavljamo kao koordinate početne i završne točke, te pozivamo konstruktor od razreda GeometrijskiLik `super("Linija")` koji će postaviti ime ovog razreda na vrijednost predanog Stringa. Nadalje implementiramo metode za dohvat privatnih vrijednosti ovog razreda (engl. getter). Očito je da linija kao geometrijski lik nema površinu niti opseg, pa te metode nije potrebno mijenjati.

```
class Linija extends GeometrijskiLik {
   /** X koordinata početka linije. */
   private int x0;
   /** Y koordinata početka linije. */
   private int y0;
   /** X koordinata kraja linije. */
   private int x1;
   /** Y koordinata kraja linije. */
   private int y1;
   /**
    * Konstruktor linije
    */
   public Linija(int x0, int y0, int x1, int y1) {
     super("Linija");  // Poziv konstruktora od geom. lika
     this.x0 = x0;
     this.y0 = y0;
     this.x1 = x1;
     this.y1 = y1;
   }
  /**
   * Dohvat X-koordinate početka linije
   */
   public int getX0() {
      return x0;
   // isto sto i: return this.x0;
   }
  // ostale metode...
}
```

Nakon što smo implementirali razred Linija kao posebnu vrstu geometrijskog lika, moža bismo htjeli isto učiniti i za pravokutnik? Pravokutnik je geometrijski lik sa površinom, opsegom, te eventualnom definicijom jedne njegove točke u koordinatnom sustavu.

Example 2.6. Pravokutnik je poseban geometrijski lik:

Opet pri deklaraciji razreda kažemo da ovaj razred nasljeđuje GeometrijskiLik, te dodatno definiramo još četiri privatne varijable, od kojih će dvije definirati gornji lijevi vrh pravokutnika, a druge dvije pravokutnikovu širinu i visinu. Pošto pravokutnik ima i opseg i površinu, potrebno je te metode (definirane u razredu GeometrijskiLik) ovdje promijeniti (engl. override). Modifikacija se vrši tako da se metoda deklarira identično kao i u nadrazredu, te se implementira ona funkcionalnost koja će osigurati smislenost vraćenih vrijednosti. U konstruktoru ovog razreda primamo četiri numeričke vrijednosti koje postavljamo kao koordinate početne točke te širinu i visinu, te pozivamo konstruktor od razreda GeometrijskiLik `super("Pravokutnik")` koji će postaviti ime ovog razreda na vrijednost predanog Stringa. Nadalje implementiramo metode za dohvat privatnih vrijednosti ovog razreda

```
class Pravokutnik extends GeometrijskiLik {
  /** X koordinata gornjeg lijevog vrha. */
  private int vrhX;
  /** Y koordinata gornjeg lijevog vrha. */
  private int vrhY;
  /** Sirina pravokutnika. */
  private int sirina;
  /** Visina pravokutnika. */
  private int visina;
  /**
   * Dohvat X-koordinate gornjeg lijevog vrha
   */
   public int getVrhX() {
        return vrhX;
   }

  /**
* Konstruktor pravokutnika.
*/
 public Pravokutnik(
      int vrhX, int vrhY, int sirina, int visina
    ) {
 super("Pravokutnik"); // Poziv konstruktora od g. lika
 this.vrhX = vrhX;
 this.vrhY = vrhY;
 this.sirina = sirina;
 this.visina = visina;
 }

  /**
   * Izračun opsega pravokutnika; ova metoda prekriva
   * istu metodu definiranu u razredu GeometrijskiLik
   */
   public int getOpseg() {
        return 2*sirina + 2*visina;
   }
  /**
   * Izračun površine pravokutnika; ova metoda prekriva
   * istu metodu definiranu u razredu GeometrijskiLik
   */
    public int getPovrsina() {
           return sirina*visina;
    }
}
```

Java razred Object

Java definira razred Object kojeg svaki razred u Javi implicitno nasljeđuje i ji ima niz metoda. Sljedeće su nam od posebnog interesa:

- Object(); - konstruktor bez argumenata, služi stvaranju instance tog razreda

• void finalize() throws Throwable; - finalizator, metoda koja se poziva neposredno prije poziva garbage collector-a

• int hashCode(); - metoda koja računa hash vrijednost nekog razreda, treba je implementirati na takav način da je osiguran velik stupanj rasipanja za slične objekte

• boolean equals(Object o); - metoda kojom se primjerak trenutnog objekta uspoređuje s nekim drugim objektom

• String toString(); - vraća tekstualni opis objekta

Sada je potrebno razred GeometrijskLik proširiti metodama `equals()` i `toString()`.

Example 2.7. Dopuna razreda GeometrijskiLik

Prvo se pitamo da li je predani objekt primjerak (instanca) razreda GeometrijskiLik, za ovo koristimo ključnu riječ *instanceof*. Nakon toga je potrebno stvoriti primerak razreda GeomtrijskiLik te castati predani objekt u taj razred. Tek nakon toga se provjerava da li trenutni objekt i predani imaju isto ime, što smo definirali kao istovjetnost dva primjerka ovog razreda.

```
public boolean equals(Object obj) {
     if( !(obj instanceof GeometrijskiLik) ) return false;
     GeometrijskiLik drugi = (GeometrijskiLik)obj;
     return ime.equals(drugi.ime);
}
public String toString() {
      return "Lik "+ime;
}
```

Sada je potrebno nadopuniti i razred Linija istim metodama. Unutar metode `eqauls()` ovog razreda, poziva se i `super.equals()`, koja će nam vratiti vrijednost false samo ukoliko imena dvaju primjeraka nisu ista. Metodu `toString()` definiramo pozivom iste metode nadrazreda te dodavanjem ispisa ostalih elemenata ovog razreda.

Example 2.8. Dopuna razreda Linija

```
public boolean equals(Object obj) {
   if( !(obj instanceof Linija) ) return false;
   Linija druga = (Linija)obj;
   if(!(super.equals(druga))) return false;
   return x0==druga.x0 && y0==druga.y0 &&
   x1==druga.x1 && y1==druga.y1;
}

public String toString() {
   return super.toString() +
 "("+x0+"," +y0+","+x1+"," +y1+")";
}
```

Na isti način ćemo proširiti i razred Pravokutnik.

Example 2.9. Dopuna razreda Pravokutnik

```
public boolean equals(Object obj) {
   if( !(obj instanceof Pravokutnik) ) return false;
   Pravokutnik drugi = (Pravokutnik)obj;
   if(!(super.equals(drugi))) return false;
   return vrhX==drugi.vrhX && vrhY==drugi.vrhY &&
   sirina==drugi.sirina && visina==drugi.visina;
}

public String toString() {
  return super.toString() +
 "("+vrhX+","+vrhY+","+sirina+","+visina+")";
}
```

Slijedi nekoliko primjera uspređivanja objekata.

Example 2.10. Primjer usporedbe objekata

I linija i pravokutnik su geometrijski likovi, a pošto naši razredi Lnija i Pravokutnik nasljeđuju razred GeometrijskiLik, mi ih mozemo spremati u varijablu definiranu kao Geometrijski lik, pozivajući konstruktore samih razreda. Isto tako smo mogli umjesto

```
class Primjer2 {
 public static void main(String[] args) {
   GeometrijskiLik lik1 = new Pravokutnik(1,1,5,5);
   GeometrijskiLik lik2 = new Pravokutnik(1,1,5,5);

   System.out.println("lik1: "+lik1.toString());
   System.out.println("lik2: "+lik2);
   System.out.println("lik1==lik2 "+(lik1==lik2));
   System.out.println("lik1.equals(lik2) "+lik1.equals(lik2));

 }
}
```

Ispis za gornji kod izgleda ovako:

```
lik1: Pravokutnik(1,1,5,5)
lik2: Pravokutnik(1,1,5,5)
lik1==lik2 false
lik1.equals(lik2) true
```

Zasto je operator ekivalencije za dva, po definiciji jednaka primjerka razreda Pravokutnik, vratio false? Treba imati na umu da su lik1 i lik2 reference, a pošto se svakim pozivom konstruktora stvara novi objekt, a time i njegova referenca, one su jedinstvene. No metoda `equals()` će vratiti true, iz razloga što su sve osobine ta dva objekta potpuno jednake, po onim kriterijima koje smo zadali prilikom implementacije te metode.

Example 2.11. Primjer usporedbe Stringova

Kako uspoređujemo Stringove?

```
class Primjer3 {
 public static void main(String[] args) {
   String s1 = new String("Ovo je tekst.");
   String s2 = new String("Ovo je tekst.");
   System.out.println("s1==s2 "+(s1==s2));
   System.out.println("s1.equals(s2) "+s1.equals(s2));
 }
}
```

Ispis za gornji kod izgleda ovako:

```
s1==s2 false
s1.equals(s2) true
```

Očekivano, rezultat usporedbe je false kad smo Stringove uspoređivali operatorom ekivalencije, jer se radi o dvije različite reference. Metda `equals()` ne uspoređuje reference, već je u razredu Stringu override-ana tako da uspoređuje vrijednosti (dakle, tekst, znak po znak).

Example 2.12. Kvadrat je poseban geometrijski lik:

Kvadrat je poseban oblik pravokutnika. Pri deklaraciji ovog razreda kažemo da razred nasljeđuje Pravokutnik, a ne GeometrijskiLik, pošto sve što vrijedi za kvadrat već imamo definirano u Pravokutniku. Konstruktor ovog razreda definiramo kao metodu koja prima tri argumenta; dva vrha i jednu stranicu, te dobivene argumente proslijeđujemo u konstruktor Pravokutnika (poziv `super("Kvadrat", vrhX, vrhY, stranica, stranica);`).

```
public class Kvadrat extends Pravokutnik {
/**
 * Konstruktor kvadrata.
 */
public Kvadrat(int vrhX, int vrhY, int stranica) {
super("Kvadrat", vrhX, vrhY, stranica, stranica);
}

/**
 * Metoda za usporedbu kvadrata.
 * @return true ako su kvadrati jednaki, inace false
 */
public boolean equals(Object obj) {
if (!(obj instanceof Kvadrat))
return false;
Kvadrat drugi = (Kvadrat) obj;
return super.equals(drugi);
}

/**
 * Vraca tekstualni prikaz ovog kvadrata.
 * @return tekstualni prikaz
 */
public String toString() {

return "Kvadrat" + "(" + super.getVrhX() + "," + super.getVrhY() + ","
 + super.getSirina() + ")";
 }
}
```

No u razredu Pravokutnik očito ne postoji konstruktor koji prima 5 argumenata, a mi se moramo osigurati da ime novog objekta bude Kvadrat, a ne Pravokutnik. Potrebno je napraviti dodatan konstruktor u razredu Pravokutnik, koji će umjesto fiksne vrijednosti, kao ime objekta moći postaviti proizvoljan, predani String. Pošto taj konstruktor ima smisla samo kad se poziva iz razreda Kvadrat, potrebno je dodati mu modifikator *protected*. Konstruktor treba izgledati ovako:

```
/**
* Zasticeni konstruktor pravokutnika. Sluzi kao potpora za
* razrede koji nasljeduju ovaj razred i zele definirati svoje

* vlastito ime.
*/
protected Pravokutnik(
         String ime, int vrhX, int vrhY, int sirina, int visina

          ) {
super(ime); // Poziv konstruktora od g. lika
this.vrhX = vrhX;
this.vrhY = vrhY;
this.sirina = sirina;
this.visina = visina;
}
```

Na ovaj način će svaki razred koji nasljeđuje razred Pravokutnik moći postaviti svoje ime.