OBJEKTNO PROGRAMIRANJE 2

Oznaka predmeta: OP2

Predavanje broj: 03

Nastavna jedinica: JAVA

Nastavne teme:

Klasa Object, interfejs, paket, konstruktor, prenos vrednosti, statička polja, statički inicijalizacioni blokovi, statički metodi, ciklična statička inicijalizacija, datoteke, ulaz izlaz.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Eckel B., *Thinking in Java*, 2nd edition, Prentice-Hall, New Jersey 2000.

Cay S. Horstmann and Gary Cornell*: "Core Java, Advanced Features", Vol. 2, Prantice Hall, 2013.*

The Java Tutorial, Sun Microsystems 2001. *http://java.sun.com*

Branko Milosavljević, Vidaković M, *Java i Internet programiranje*, GInT, Novi Sad 2002.

main

- Metoda main mora biti public, static i void, te imati jedan argument tipa String[].
- Aplikacija može imati više main metoda, svaka klasa može imati jedan main metod.
- Stvarno korišćeni main je specificiran imenom klase pri pokretanju programa

Preporuka: svaka klasa treba da ima main metod za potrebe testiranja

Primer: ispisati parametre metode main

```
class Eho {
  public static void main(String[] argumenti) {
    for (int i=0; i<argumenti.length; i++)
      System.out.print(argumenti[i]+" ");
    System.out.println();
  }
}
```

primer poziva iz command prompta:

```
java Eho prvi drugi treci
```

rezultat će biti:

```
prvi drugi treci
```

Petlja: for-each

- Konvencionalna for iteracija:

```
public int sumaNiza(int niz[]){
  int suma=0;
  for (int i=0; i<niz.length;i++)
    suma+=niz[i];
  return suma;
}
```

- Ekvivalentni program sa iteracijom foreach:

```
public int sumaNiza(int niz[]){
  int suma=0;
  for (int e: niz)
    suma+=e;
  return suma;
}
```

Primer klase tačka

```
class Tacka{
  public double x;
  public double y;
  public static Tacka ishodiste=new Tacka();
                        //ishodiste je clan na nivou klase
  public Tacka() {
    inicijalizuj();
  }
  public Tacka(double x, double y){
    this.x=x;
    this.y=y;
  }
  public void inicijalizuj(){
    x=0;y=0;
  }
  public double rastojanje(Tacka t){
    double dx, dy;
    dx=x-t.x; dy=y-t.y;
    return Math.sqrt(dx*dx+dy*dy);// Math klasa -statičke metode
  }
}
```

Klasa Piksel

- Primeri poziva metode rastojanje iz metoda neke druge klase

```
Tacka prva = new Tacka(50.0,100.0);
//ili:  Tacka prva = new Tacka(); prva.x=50.0; prva.y=100.0;
double d=prva.rastojanje(Tacka.ishodiste);
Tacka druga = new Tacka(5.0,10.0);
d = druga.rastojanje(prva);
```

- Klasa Piksel: proširuje strukturu podataka dodavanjem atributa boja, menja ponašanje redefinisanjem metoda inicijalizuj i zadržava interfejs klase Tacka.

```
class Piksel extends Tacka{
  Boja boja; //referenca klase Boja
  public void inicijalizuj(){
    super.inicijalizuj();
    boja=null;
  }
}
```

- Objekti Piksel se mogu koristiti tamo gde se očekuju objekti Tacka

  Primer:

```
Tacka p = new Piksel();
p.inicijalizuj(); // poziva se inicijalizuj() klase Piksel
```

Klasa Object

- Klase koje ne proširuju eksplicitno druge klase, implicitno proširuju klasu Object.
- Object se nalazi u korenu hijerarhije klasa.
- Object je najopštija klasa za reference koje mogu da upućuju na objekat proizvoljne klase.

  Primer:

```
Object o = new Piksel();
o = "Petar Petrović“;
```

  legalno je referencu postaviti da upućuje na Piksel (u ovom primeru) i na String objekte.
- Klasu Object bi trebalo koristiti kada se želi da metoda prihvati referencu na bilo kakav objekat.
  - U ovom slučaju potrebno je proveriti kojoj klasi pripada stvarni argument kako bi se moglo baratati sa različitim stvarnim argumentima.

Interfejs

- Interfejs pretraživača (tipa koji omogućuje traženje po imenu)

```
interface Pretrazivac{
  /** Vraca objekat pridružen imenu ili null, ako nema takvog objekta */
  Object nadji(String ime);
}
```

- Klasa koja implementira interfejs Pretrazivac:

```
class JednostavanPretrazivac implements Pretrazivac {
  private String[] imena;
  private Object[] vrednosti;
  public Object nadji(String ime){
    for (int i=0; i<imena.length; i++)
      if (imena[i].equals(ime)) return vrednosti[i];
  }
}
```

- Kod koji koristi reference na objekat Pretrazivac

```
void obradiVrednosti(String[] imena, Pretrazivac tabela){
  for (int i=0;i<imena.length; i++){
    Object vrednost=tabela.nadji(imena[i]);
    if (vrednost != null)obradiKonkretnuVrednost(imena[i],vrednost);
  }
}
```

Korišćenje paketa

- Kada je potrebno koristiti neki deo nekog paketa može se:
  - koristiti njegovo puno ime:

```
class Datum1{
  public static void main(String[] args){
    java.util.Date danas=new java.util.Date();
    System.out.println(danas);
  }
}
```

  - uvesti (import) ceo paket ili neku klasu paketa:

```
import java.util.Date;
class Datum2{
  public static void main(String[] args){
    Date danas = new Date(); System.out.println(danas);
  }
}
```

Primer konstruktora

- Primer klase NebeskoTelo sa podrazumevanim konstruktorom:

```
class NebeskoTelo {
  public long id;
  public String ime = "<neimenovano>";
  public NebeskoTelo kurziOko = null;
  private static long sledeciID = 0;
  NebeskoTelo() {id = sledeciID++;}
}
```

Pozivi metoda:

```
NebeskoTelo sunce = new NebeskoTelo(); // id == 0
sunce.ime = "Sunce";
NebeskoTelo zemlja = new NebeskoTelo();// id == 1
zemlja.ime = "Zemlja";
zemlja.kruziOko = sunce;
```

- Primer drugog konstruktora koji ima parametre

```
NebeskoTelo(String imeTela, NebeskoTelo centar){
  this(); //eksplicitan poziv konstruktora bez parametara
  ime = imeTela;
  kruziOko = centar;}
```

Metod toString()

Pozivi metoda:

```
NebeskoTelo sunce = new NebeskoTelo ("Sunce", null);
NebeskoTelo zemlja = new NebeskoTelo ("Zemlja", sunce);
```

- Malo korišćenje metoda toString() (konverzija proizvoljnog objekta u String).

```
public String toString(){
  String opis = id + " (" + ime + ")";
  if (kruziOko != null)
    opis+= " centar rotacije: " + kruziOko;
  return opis;
}
```

Pozivi:

```
System.out.println("Telo " + sunce);
System.out.println("Telo " + zemlja);
```

Izlaz:

```
Telo 0 (Sunce)
Telo 1 (Zemlja) centar rotacije: 0 (Sunce)
```

Primer prenosa po referenci

Primer:

```
public static void main(String[] argumenti){
  NebeskoTelo venera = new NebeskoTelo("Venera", null);
  System.out.println("pre: " + venera);
  drugoIme(venera);
  System.out.println("posle: " + venera);
}
public static void drugoIme(NebeskoTelo telo){
  telo.ime = "Zvezda Danica";
  telo = null; // nema značaja
}
```

Izlaz:

```
pre: 0 (Venera)
posle: 0 (Zvezda Danica)
```

- Podsetiti se šta je rečeno za String.

Primer prenosa po vrednosti

Primer:

```
public static void main(String[] argumenti){
  double jedan = 1.0;
  System.out.println("pre: jedan=" + jedan);
  prepolovi(jedan);
  System.out.println("posle: jedan=" + jedan);
}
public static void prepolovi(double arg){
  arg /= 2.0; System.out.println("funkcija: pola="+arg);
}
```

Izlaz:

```
pre: jedan=1
funkcija: pola=0.5
posle: jedan=1
```

instanceof

- Reč instanceof čitajte kao da li referenca ukazuje na konkretan objekat date klase:

```
public class MainClass {
   public static void main(String[] a) {
      String s = "Hello";
      int i = 0;
      String g = null;
      if (s instanceof java.lang.String) {
         System.out.println("s is a String");
      }
      //if (i instanceof Integer) {
      //   System.out.println("i is an Integer");
      //} //nece dozvoliti ovu proveru
      if (g instanceof java.lang.String) {
         System.out.println("g is a String");
      }
}
```

- Izlaz je: `s is a String`
- Umesto *int i* probajte *Integer i* .

Statičko polje

- Statičko polje (promenljiva klase) ima samo jednu instancu po klasi.
- Statičko polje je tačno jedna promenljiva bez obzira na broj (čak 0) objekata klase.
- Statičko polje se inicijalizuje pre nego što se:
  - bilo koji statički član te klase koristi
  - bilo koji metod te klase počne izvršavanje
- Primer:

```
class Inicijalizacija{
  public static double x = 10.0;
  private double y;
  public void init() {
    y=x;
  }
} // x je definisano polje u trenutku korišćenja
```

Statički inicijalizacioni blokovi

- Služe za inicijalizaciju statičkih polja ili drugih stanja.

```
class ProstiBrojevi{
  public static int[] prviProstiBrojevi=new int[100];
  static {
    prviProstiBrojevi[0]=1;
    for (int i=1; i< prviProstiBrojevi.length; i++)
      prviProstiBrojevi[i]=sledeciProstBroj();
  }
  // ...
}
```

- Redosled statičke inicijalizacije: sleva-udesno i odozgo-naniže.
- Preporuka je da se u okviru statičkog inicijalizacionog bloka pozivaju statičke metode koje su sigurne (ne bacaju izuzetke).
- Nakon izvršenja inicijalizacionog statičkog bloka trebalo bi kao rezultat da budu pripremljene sve vrednosti statičkog člana koga inicijalizuje taj blok.
  - Npr. ako je u pitanju niz kao u datom primeru
- U static bloku može se dodeliti vrednost static final elementu ako već nije dodeljena (može i u konstruktoru). Razmislite šta znači *final class* po uzoru na *final* metoda?

Statički metodi

- Statički metod može obavljati opšti zadatak za sve objekte klase i može direktno pristupati samo statičkim poljima i statičkim metodima klase.
- Ne postoji this referenca (nema specifičnog objekta nad kojim se radi).
- Izvan klase – statičkom članu se pristupa koristeći ime klase i operator

```
prostBroj = ProstiBrojevi.sledeciProstBroj();
n = ProstiBrojevi.prviProstiBrojevi.length;
```

```
class PrimerStatInit{
  public static int[] nizFibonaci=new int[50];
  static {
      popuniFibonacijevNiz();
  }
  private static void popuniFibonacijevNiz(){
    nizFibonaci[0]= nizFibonaci[1]=1;
    for (int i=2; i< nizFibonaci.length; i++)
      nizFibonaci[i]= nizFibonaci[i-1] +
                      nizFibonaci[i-2];
  }
  public static void main(String arg[]){
    System.out.println("Peti element Fibonacijevog niza je " +
                        PrimerStatInit.nizFibonaci[4]);
  }}
```

Ciklična statička inicijalizacija

- Javlja se kada inicijalizacioni blok jedne klase poziva metodu druge klase a u drugoj klasi njen inicijalizacioni blok poziva metodu prve klase.
- Problem:
  - ako statički inicijalizator u klasi X poziva metod u klasi Y,
    a statički inicijalizator u klasi Y poziva neki metod u klasi X
  - ne može se otkriti u vreme prevođenja

- Ponašanje:
  - inicijalizatori X se izvršavaju do tačke poziva metoda klase Y
  - pre nego što se izvrši metod klase Y, izvršavaju se Y inicijalizatori
  - kada inicijalizator Y pozove metod klase X ovaj se izvrši (iako nije završena inicijalizacija)
  - završavaju se inicijalizatori klase Y
  - izvršava se pozvani metod klase Y
  - konačno, inicijalizatori klase X se završavaju

Ciklična statička inicijalizacija - primer

```
class A{
  static {
    System.out.println("Izvršenje statičkog bloka A počelo");
    B.metod();
    System.out.println("Izvršenje statičkog bloka A završava");
  }
  static void metod(){ System.out.println("A.metod"); }
  }
class B{
  static {
    System.out.println("Izvršenje statičkog bloka B počelo");
    A.metod();
    System.out.println("Izvršenje statičkog bloka B završava");
  }
  static void metod(){ System.out.println("B.metod");}
}
```

```
class T{ public static void main(String[] args){A a = new A(); } }
```

Izlaz:

```
Izvršenje statičkog bloka A počelo
 Izvršenje statičkog bloka B počelo
 A.metod
 Izvršenje statičkog bloka B završava
B.metod
Izvršenje statičkog bloka A završava
```

Metod finalize

- Ranije je rečeno o metodi finalize gde je naglašeno da garbage collector ne mora osloboditi prostor koji je objekat koji više nije referenciran zauzimao.
- Ako se koristi metoda finalize onda je preporuka kao što sledi:

```
protected void finalize() throws Throwable {
  super.finalize(); // dobra praksa
  //...
}
```

```
public class ObradaFajla{
  private MyStream fajl;
  public ObradaFajla(String ime){ fajl=new MyStream(ime);}
  //...
  public void zatvori(){
    if(fajl!=null){ fajl.close(); fajl=null;    }
  } //ne oslanjati se na finalize
  protected void finalize() throws Throwable {
    super.finalize();
    zatvori(); //posto je implementiran finalize dodati i ovo
               //mada se u praksi ovde dodaje logovanje
  }
```

Datoteke

- Primer rada sa datotekama:
  program kopira sadržaj datoteke ulazna.txt u datoteku izlazna.txt

```
import java.io.*;
public class Copy {
   public static void main(String[] args) throws IOException {
     File inputFile = new File("ulazna.txt");
     File outputFile = new File("izlazna.txt");

      FileReader in = new FileReader(inputFile);
      FileWriter out = new FileWriter(outputFile);
      int c;

      while ((c = in.read()) != -1){
          out.write(c);
      }

      in.close(); out.close();
     }
}
```

Datoteke

- Za korišćenje slučajnog pristupa datoteci može se koristiti klasa RandomAccessFile. U sledećem primeru kreira se datoteka u koju se upisuju i čitaju različiti tipovi podataka

```
import java.io.File;
import java.io.RandomAccessFile;
import java.io.IOException;
public class DemoRandomAccessFile {
  private static void probaRAFa() {
    try {
      File file = new File("C:/primer/RAFdemo.out");
      RandomAccessFile raf =new RandomAccessFile(file,"rw");
      raf.writeByte(65);
      raf.writeBytes("Elvis"); // zero string
      raf.write(0x0A);
      raf.seek(0);
      // cita se bajt
      byte ch = raf.readByte();
      System.out.println("Prvi karakter u fajlu je : " + (char)ch);
      // sada se cita preostali deo linije
      // od trenutne pozicije kursora
      // sve do znaka za kraj linije
      System.out.println("Linija teksta je : " + raf.readLine());
```

```
      // pomeranje na kraj fajla
      long mojapozicija = file.length();
      raf.seek(mojapozicija);
      // dodavanje na kraj fajla
      raf.write(0x0A);
      raf.writeBytes("Zavrsni tekst.");
      // pozicioniranje na pocetak dodatog dela
      raf.seek(mojapozicija);
      System.out.println("Dodato je:"+ raf.readByte() + raf.readLine());
      raf.close();
    }
    catch (IOException e) {
        System.out.println("IOException:");
        e.printStackTrace();
    }
  }
  public static void main(String[] args) { probaRAFa(); }
}
Izlaz:
   Prvi karakter u fajlu je : A
   Linija teksta je : Elvis
   Dodato je:10Zavrsni tekst.
```

Ulaz sa konzole

```
import java.io.*;
public class IOHelp {
  private static InputStreamReader isr = new
   InputStreamReader(System.in);
  private static BufferedReader br = new BufferedReader(isr);

  public static String readLine(String p) {
      String retVal = "";
      System.out.print(p+"> ");
      try {          retVal = br.readLine();        }
      catch (Exception e){
        System.out.println("IOHelp: " + e.getMessage());
      }
      return retVal;
  }
  public static int readInt(String prompt) {
    try {        return Integer.parseInt(readLine(prompt));
    }
    catch(Exception e) {
      System.out.println("Error reading int"); return 0; //npr
    }
  }
  public static double readDouble(String prompt) {
    try {       return Double.parseDouble(readLine(prompt));     }
    catch(Exception e) {
      System.out.println("Error reading double"); return 0;//npr
    }
  }
}
```

Korišćenje stream-ova za datoteke

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class WriteBinary2 {
public static void main(String[] argv) throws IOException {
    int i=42;
    double d = Math.PI;
    String strFileName ="binbin.bin";
    DataOutputStream dos =
                 new DataOutputStream(
                     new FileOutputStream(strFileName));
    dos.writeInt(i);
    dos.writeDouble(d);
    dos.close();
    DataInputStream dis =
                new DataInputStream(
                    new FileInputStream(strFileName));
    System.out.println("procitan : "+ dis.readInt() + "\nprocitan : " +
                               dis.readDouble());
    dis.close();
  }}
```

```
Izlaz:
procitan : 42
procitan : 3.141592653589793
```

Primer ulaza sa konzole korišćenjem stream-ova

- Posmatra se trivijalan primer:

```
import java.io.*;
class Test {
  public static void main(String[] args) {
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    System.out.println("unesite prirodan broj ");
    String str = "";
    try {
      str = br.readLine();
      int i = Integer.parseInt(str);
      if (i > 0)
      switch (i % 2) {
        case 0: System.out.println(i + " je paran broj"); break;
        default:
          System.out.println(i + " je neparan broj");
      }
    } catch (Exception e) { System.err.println("IOHelp: " +
                                                e.getMessage());}
  }
}
```

Primer Hanoj

```
import java.io.*;
public class Hanoj{
  private void prebaci(int n, int sa, int na, int pom){
        if(n>0){
          prebaci(n-1,sa,pom,na);
          System.out.println(sa+"-->"+na);
          prebaci(n-1,pom,na,sa);
        }
  }
  public static void main(String args[]){
        Hanoj han = new Hanoj();
        InputStreamReader isr=  new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String str=""; int brojkolutova;
        do{     System.out.println("unesite broj kolutova?(1-5)");
          try{  str = br.readLine();
                brojkolutova = Integer.parseInt(str);
          }
          catch( Exception ex) {  System.out.println( "Izuzetak "+ex);
                                  brojkolutova = -1;
          }
        }while((brojkolutova<1)||(brojkolutova>5));
        han.prebaci(brojkolutova,1,3,2);  }}
```

Scanner

- Za unos sa tastature koristi se i klasa Scanner sa parametrom `System.in`:

```
import java.util.*;
public class Skener {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        scan.useDelimiter("\\n"); // dobro za String pomocu next()
        System.out.println("unesite integer ? ");
        int i = scan.nextInt();   // unos integera
        System.out.println("integer je " + i);
        System.out.println("unesite long ? ");
        long ll = scan.nextLong(); // unos longa
        System.out.println("long je " + ll);
        System.out.println("unesite string ? ");
        String stri = scan.next(); // unos Stringa
        System.out.println("integer je " + stri);
    } // napisite bolji kod
}
```

- Za preuzimanje unosa sa tastature i tretiranja kao double koristiti nextDouble(), a za String može i String:nextLine() (ali paziti na zaostali \n).

```
Scanner sc = new Scanner(new File("myNumbers")); //iz File-a
while (sc.hasNextLong()) {
  long aLong = sc.nextLong();      }
```

Vector

- Klasa Vector je kontejnerska klasa kolekcije objekata kojima se pristupa indeksno.
- Obezbeđeno je dodavanje i uklanjanje elemenata na proizvoljoj indeksnoj poziciji.
- Klasa koristi polimorfizam tako da smešta reference na objekte tipa Object, čime je postignuta univerzalnost u tipu elemenata koje klasa Vector može sadržavati.

```
import java.util.Vector;
class PrimerVector {
  public static void main(String args[]) {
    Vector vector = new Vector();
    vector.addElement("Nulti element je String");
    vector.addElement(new Integer(5));
    vector.addElement("Drugi element ");
    for (int i=0; i<vector.size(); i++)
       System.out.println(vector.elementAt(i) + " ");
  }
}
```

Hashtable

- Klasa Hashtable vrši mapiranje ključeva u vrednosti. Ključevi i vrednosti su objekti. U primeru koji sledi prikazuje se izvlačenja kuglica sa brojevima iz bubnja sa ponavljanjem (izvučena kuglica se ponovo vraća u bubanj).
- Klasa Brojac služi za pamćenje koliko je puta izvučena kuglica sa brojem od 1 do 39. Konstruktor ove klase inicijalno dodeljuje podatku članu ***i*** vrednost 1 koja označava je je kuglica kojoj je pridružen objekat ove klase upravo izvučena.

```
class Brojac{
  private int i;
  Brojac(){ i=1; }
  void Inkrementiraj(){
    i++;
  }
  public String toString() {
    return Integer.toString(i) + "\n";
  }
}
```

- Ideja je da se u klasi PrimerHashtable u main metodi kreira referenca na objekat klase Hashtable koja će služiti za mapiranje brojeva od 1 do 39 kao ključeva i broja pojavljivanja datog broja kao pripadne vrednosti. U petlji od 10000 iteracija kreiraju se i ključevi i pripadne vrednosti pojavljivanja.

Hashtable

```
import java.util.*;
class PrimerHashtable{
  public static void main(String args[]) {
    Hashtable ht = new Hashtable();
    for (int i = 0; i < 10000; i++) {
      Integer r = new Integer((int)(1+(Math.random()*39)));
      if(ht.containsKey(r))
         ((Brojac)ht.get(r)).Inkrementiraj();
      else        ht.put(r, new Brojac());
    }
    System.out.println(ht);
  }
} //Random klasa, metoda nextInt(), broj = min + r.nextInt((max-min)+1);
```

- Metodom containsKey proverava se da li hash-tabela ima ključ koji odgovara tom broju:
  - ako ne postoji takav ulaz on se kreira metodom put koja ima formalne argumente ključ i vrednost, a to su ovde Integer r i novi neimenovani Brojac. Ovde će konstruktor klase Brojac postaviti vrednost brojanja na 1.
  - ako postoji ulaz za dati ključ (broj je već bio izvučen) onda se uzima pripadni brojač (kastovanjem na objekat koji se uzima) metodom get i inkrementira njegova vrednost brojanja.