# Vježbe 2 - varijable, grananja, petlje, scope, korištenje komandnog prozora i matematičke biblioteke

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

*matmih@math.hr*

18. listopada, 2022.

# Varijable

Lokalne varijable **deklarirane unutar metoda** ili **scope-a** se **ne inicijaliziraju automatski** unutar programskog jezika *Java*. Varijable deklarirane kao **članice klase** se **inicijaliziraju automatski** na početnu vrijednost **ovisno o tipu varijable**.

```
1 int a=2, b,c,d;
2
3     b = c+d;//greska pri prevodenju, varijable c i d nisu
    inicijalizirane!
4
5     b = c = d = a; //sada su sve varijable inicijalizirane
6     b = c+d; //OK!
```

Deklariranje lokalnih varijabli

# Konstantne varijable

*Java* također podržava konstantne varijable, deklariraju se dodavanjem ključne riječi `final`. Konstantnim varijablama **smijemo pridijeliti vrijednost samo jednom** i nakon toga je više **ne možemo mijenjati**.

```
1 final int e = a;//e je konstantna varijabla
2 final int f; //f je neinicijalizirana konstantna varijabla
3     f = b; //OK - inicijalizacija od f
4     f = e; //Greska pri prevodenju! f je konstantna
    varijabla
```

Deklariranje konstantnih varijabli

**Statičke varijable** se u *Javi* mogu deklarirati samo kao **članice klasa**! Funkcionalnost slična kao i u C++-u (varijablu dijele sve instance klase). Više o tome kod klasa i sučelja.

# Lokalne varijable nedefiniranog tipa *var*

Tip varijabli deklariranih s ključnom riječi `var` **zaključuje prevodioc** ili prema **tipu objekta koji se kreira** ili **prilikom pridjeljivanja vrijednosti** pomoću nekog izraza. *Java* **mora znati** tip svih varijabli za **vrijeme prevođenja**, stoga kada se jednom odredi tip varijable deklarirane ključnom riječi `var`, on se **više ne može mijenjati**.

```
1 var id=0;// Varijabla id postaje tipa int
2 id="34"; // Greska pri prevodenju (id je tipa int)
3 //java.lang.String se ne moze prevesti u tip int
4 var count=null;// Greska pri prevodenju, ne moze se odrediti
      tip reference
5 var x; //Greska pri prevodenju, varijablu treba
     inicijalizirati
6 var kont = new ArrayList<HashSet<Integer>>(); //korisno kod
     ovakvih deklaracija
```

Deklariranje varijabli ključnom riječi var

Varijable deklarirane ključnom riječi `var` se **ne mogu koristiti** kao **ulazni parametri funkcija**.

# Doseg varijabli

Varijable definirane unutar metode se smiju koristiti **bilo gdje unutar tijela metode nakon deklaracije varijable**.

```
1  public class Doseg {
2    public static void main(String[] args) {
3
4
5      // naredbe iznad ne smiju koristiti broj
6      double broj = 100.0;
7     //naredbe ispod smiju koristiti broj
8
9      System.out.println(broj);
10   }
11 }
```

Doseg varijabli

**Moguće je definirati blokove** koristeći separatore { i }. Varijable definirane unutar tih blokova **imaju doseg samo od deklaracije do kraja bloka**. U *Javi* **nije dozvoljeno maskiranje unutar tijela metode**. Jedino se **može maskirati varijabla članica klase**, **lokalnom varijablom** unutar metode.

# Doseg varijabli

```
1  public class Doseg {
2    public static void main(String[] args) {
3      // naredbe ne mogu koristiti broj
4       double broj1 = 22.55;
5      { // Blok
6        // naredbe ne mogu koristiti broj
7
8        double broj = 100.0;
9        double broj1 = 52.0;//nije dopusteno maskiranje unutar
       metode, greska!
10       //naredbe mogu koristiti broj
11       System.out.println(broj);
12
13     } // Kraj bloka
14
15   // Naredbe ne mogu koristiti broj
16   double broj = 23.5; // OK, dozvoljeno!
17   }
18 }
```

Doseg varijabli

# Doseg varijabli

```
1  public class Doseg {
2  int broj1; //OK! Inicijalizira se na 0!
3
4    public static void main(String[] args) {
5      // naredbe ne mogu koristiti broj
6       double broj1 = 22.55;//maskiranje dopusteno
7      { // Blok
8        // naredbe ne mogu koristiti broj
9        double broj = 100.0;
10       double broj1 = 52.0;//nije dopusteno maskiranje unutar
       metode, greska!
11       //naredbe mogu koristiti broj
12       System.out.println(broj);
13     } // Kraj bloka
14
15   // Naredbe ne mogu koristiti broj
16   double broj = 23.5; // OK, dozvoljeno!
17   }
18 }
```

Doseg varijabli

# Grananja

Kao i u C/C++, imamo *if* i *switch*.

```
if (uvjet1) {
  // blok koda za izvrsiti ukoliko je uvjet1 istinit
} else if (uvjet2) {
  // blok koda za izvrsiti ukoliko je uvjet2 istinit
} else {
  // blok koda koji treba izvrsiti ukoliko su uvjet1 i
    uvjet2 false
}

```

Grananje - if

Jednostavno grananje se može postići i uporabom uvjetnog operator ? (ternarni).

```
  variabla = (uvjet) ? izrazIstina :  izrazLaz;

```

Uvjetni operator

# Grananja

```
switch(izraz) {
  case a:
    // blok ukoliko je a istina
    break; //ukoliko nema break nastavlja dalje
  case b:
    // blok ukoliko je b istina
    break;
  default: //svi preostali slucajevi
    // blok koda
}

```

Grananje - switch

Ukoliko na kraju bloka koda povezanog s nekim case-om, ne stavimo naredbu `break`, **nastavit će se izvršavati blok** koda od sljedećeg case-a.

# Grananja

Od Java verzije 17, u `switch` se uvode pravila koja imaju funkcionalnost kao case uz upotrebu `break`:

```
1 switch(izraz) {
2   case a -> //naredba ukoliko je a istina
3   case b -> //naredba ukoliko je b istina
4   default -> // naredba za sve preostale slucajeve
5 }
6
```

Grananje - switch

Moguće je koristiti i `case null` što prije **nije** bilo moguće. Također je dozvoljeno korištenje kompleksnijih uvjeta tipa `case String st && (st.length > 1) -> naredba`. Kod ovog načina korištenja `switch` naredbe, nakon -> smije doći **jedna naredba** ili **niz naredbi ukoliko se nalaze unutar eksplicitno definiranog bloka** (npr. `case c -> {System.out.println(c); c++;}`).

# Petlje

Petlje `for`, `while` i `do-while` se ponašaju identično kao u C/C++-u. Jedina razlika je mogućnost iteriranja po kontejnerima unutar for petlje.

```
1  double brojevi[] = {1.4,2.5,4.8,6.9};
2    //ocuvanje poretka ovisi o tipu spremnika
3   for(double broj:brojevi){ //kod polja ocuvan poredak
4      System.out.println(broj);
5      }
6
7   for(double broj:brojevi){
8     if(broj>20.0)
9        break; //ako broj >20 prekinemo izvodenje for petlje
10       }
11
12  for(double broj:brojevi){
13    if(broj>20.0)
14       continue; //ako broj >20 prijedemo na sljedecu
     iteraciju for petlje
15       }
```

Primjer iteriranja po polju

# Korištenje funkcija paketa *java.lang.Math*

Funkcije paketa *java.lang.Math* se koriste **navođenjem** *Math.konstanta* ili *Math.metoda* unutar koda (sve metode i elementi su `static`). Pošto je paket `java.lang` **standardno dostupan** unutar *Java* programa, ne treba uključivati pakete koje sadrži (inače se to radi naredbom `import punoime_paketa`).

## Zadatak 1.

Isprobajte razne funkcije paketa *java.lang.Math*. Ispišite konstante e i $\pi$.

# Rad s komandnom linijom

Učitavanje s komandne linije korištenjem klase `Buffered Reader`.

```
1  import java.io.BufferedReader; //ucinkovito citanje inputa
2  import java.io.IOException;  //citanje/pisanje moze izazvati
       iznimku koju treba obraditi
3  import java.io.InputStreamReader; //ova klasa vrsi citanje
4  public class Test {
5      public static void main(String[] args)
6          throws IOException
7  {
8    BufferedReader citac = new BufferedReader(
9          new InputStreamReader(System.in)); //ucitamo podatke
10
11         String ime = citac.readLine(); //sada mozemo ucitane
       podatke jednostavno spremiti u string
12
13         System.out.println(ime); //ispis ucitane linije
14     }
15 }
```

Učitavanje i ispis korištenjem komandnog prozora

# Rad s komandnom linijom

Učitavanje s komandne linije korištenjem klase `Scanner`.

```
1  import java.util.Scanner; //klasa koja vrsi citanje
2
3  class Test1 {
4    public static void main(String args[])
5     {
6     Scanner ulaz = new Scanner(System.in); //dohvacanje ulaza
        s komandne linije
7     String s = ulaz.nextLine(); //spremanje u string
8     System.out.println("Uneseni string: " + s); //ispis u
       komandni prozor
9     int a = ulaz.nextInt(); //ucitavanje broja tipa int
10    System.out.println("Uneseni int: " + a); //ispis
11    float b = ulaz.nextFloat(); //ucitavanje broja tipa float
12    System.out.println("Uneseni float: " + b); //ispis
13 //scanner baca iznimku ukoliko je ucitan podatak krivog tipa
14     }
15 }
```

Učitavanje i ispis korištenjem komandnog prozora

# Rad s komandnom linijom

Učitavanje s komandne linije korištenjem klase `Consol`.

```
1 public class Test3 {
2     public static void main(String[] args)
3     {
4         String ime = System.console().readLine(); //
    ucitavanje linije i spremanje u string
5 
6         System.out.println("Unesli ste string: " + ime);
7     }
8 }
```

Učitavanje i ispis korištenjem komandnog prozora

# Rad s komandnom linijom

Učitavanje s komandne linije korištenjem varijabilnih argumenata.

```
1  class Test4 {
2      public static void main(String[] args) //args sadrzi
       argumente komandne linije
3      {
4
5        if (args.length > 0) { //provjeri postoje li argumenti
6              System.out.println("Argumenti:");
7
8              for (String val : args) //ispisujemo argumente
9                  System.out.println(val);
10         }
11         else
12             System.out.println("Nema argumenata "
13                                + "komandne linije.");
14     }
15 }
```

Učitavanje i ispis korištenjem komandnog prozora

# Izvlačenje informacija iz stringa

## Zadatak 2.

Učitajte dva broj u komandnu liniju, jedan tipa *int* i jedan tipa *double*, odvojene razmakom (bjelinom). Učitavanje linije u *Java* program vršite koristeći klasu `Buffered Reader` ili *Consol*. Izdvojite učitane brojeve te ih spremite u varijable odgovarajućeg tipa. Ispišite njihove vrijednosti.

**Napomene:** string možemo dijeliti koristeći metodu `String[] split(String regex)` definiranu unutar klase `String` (primijetite da funkcija nije *static*, stoga je potrebna instanca klase za poziv). String reprezentaciju broja tipa *int* i *double* možemo prebaciti u taj tip koristeći naredbe `double Double.parseDouble(String str)` i `int Integer.parseInt(String str)` (funkcije `public static double parseDouble(String str)` i `public static int parseInt(String str)` su **static metode** unutar klasa `Double` i `Integer` stoga nisu potrebne instance klase za poziv.