6. SWITCH IZRAZ I PETLJE

SADRŽAJ

---

1. `switch` izraz (kontrolna struktura)

Osim grananja sa if-else naredbom drugi način grananja je upotreba `switch` kontrolne strukture. Switch izraz se koristi rjeđe od if-else izraza, ali postoje određeni programerski zadaci gdje je korisna njegova upotreba . Switch izraz omogućava nam testiranje vrijednosti određenog izraza i ovisno o vrijednosti izraza, skok na određenu lokaciju unutar switch izraza. Vrijednost ispitivanog izraza treba biti cjelobrojna ili karakter. Ne može biti tipa String ili broj u pokretnom zarezu. Pozicije na koje je moguće skočiti imaju formu "case konstanta:". To je mjesto gdje program se nastaviti izvršavati kada je vrijednost izraza jednaka konstanti. Kao zadnji slučaj u switch izrazu možete opcionalno koristiti oznaku "default:", koja predstavlja mjesto gdje će program nastaviti s izvršavanjem ako nije odabrana nijedan "case konstanta:".

Forma `switch` izraza je:

```
switch (izraz) {
    case konstanta-1:
      izrazi-1
      break;
    case konstanta-2:
      izrazi -2
      break;
      .
      .   // (više case naredbi)
      .
    case konstanta-N:
      izrazi -N
      break;
    default:  // opcionalni default slučaj
      izrazi -(N+1)
} // kraj switch naredbe
```

Naredba `break` je tehnički opcionalna. Njen je efekt skok na kraj switch izraza. Ako bismo je izostavili Java bi nastavila s izvršavanjem slijedeće naredbi vezanih uz slijedeći case izraz. To je rijetko ono što želite, ali je sasvim legalno. Ponekad se može upotrijebiti i return na mjestu gdje bi inače upotrijebili break.

Slijedi primjer switch izraza. Primijetite da konstante u case oznakama ne moraju biti poredane , osim što moraju biti različite.

```
        switch (N) { // pretpostavimo da je N cjelobrojna varijabla
           case 1:
              System.out.println("Broj je 1.");
              break;
           case 2:
           case 4:
           case 8:
              System.out.println("Broj je 2, 4, ili 8.");
              System.out.println("(Broj je potencija broja 2)");
              break;
           case 3:
           case 6:
           case 9:
              System.out.println("Broj je 3, 6, ili 9.");
              System.out.println("(Broj je multiplikant broja 3)");
              break;
           case 5:
              System.out.println("Broj je 5.");
              break;
           default:
              System.out.println("Broj je 7,");
              System.out.println("   ili je izvan područja 1 do
9.");
         }
```

2. `while` petlja

Slijedi primjer koda kojeg bismo lakše realizirali pomoću neke petlje.
Npr. Treba unijeti četiri broja: (Pretpostavimo da je `in` objekt tipa ConsoleReader)

```
System.out.println(″Unesi vrijednosti:″);
double suma = 0;
suma  = suma + in.readDouble();
suma = suma + in.readDouble();
suma = suma + in.readDouble();
suma = suma + in.readDouble();
System.out.println(″Suma je ″ + suma);
```

Za četiri broja mogli bismo i zadržati prethodni kod, ali za slučaj da unosimo više brojeva ili da npr. iz neke datoteke čitamo na tisuće brojeva potrebno je upotrijebiti neku od petlji.

Pretpostavimo da varijabla `n` sadrži broj ulaznih vrijednosti.

```
System.out.println(”Unesi vrijednosti:”);
double suma = 0;

ponovi n puta  (pseudokod)
   suma = suma + in.readDouble();

System.out.println(”Suma je ” + suma);
```

Dio koda

```
Ponovi n puta
   suma = suma + in.readDouble();
```

potrebno je konvertirati u Java kod. Upotrijebiti ćemo **petlje** (**loop statements).**

Prva petlja koju ćemo upotrijebiti je **while** petlja čiji je opći oblik:

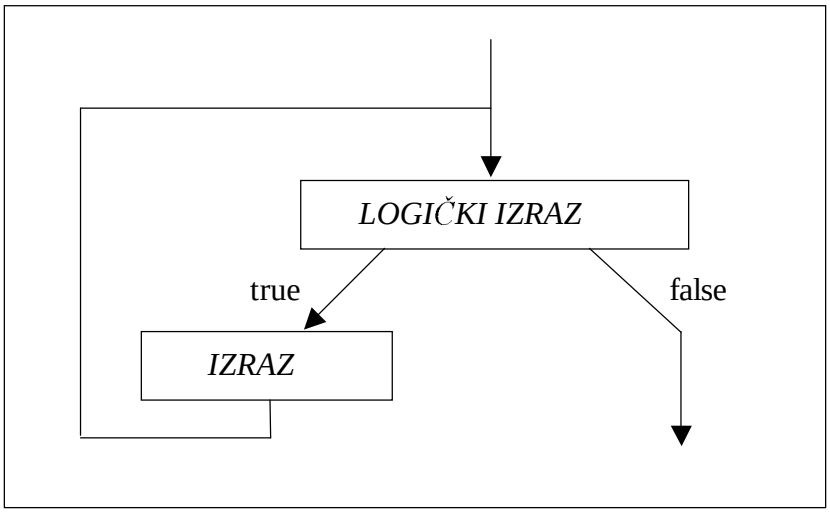
while petlja

```
while (LOGIČKI IZRAZ)
    IZRAZ
```

Što znači:

Izvršavaj u petlji *IZRAZ*.
Svaki put prije izvršavanja *IZRAZ* provjeri
da li je *LOGIČKI IZRAZ* ima vrijednost true.
Ako nema završi odmah.

Slijedeći dijagram toka pokazuje što se događa tijekom izvršavanja `while` petlje.



Izraz koji se ponavlja je **tijelo** petlje. Tijelo petlje može biti i blok koji se sastoji od niza izraza.

Petlja `while` može se koristiti za obavljanje neke operacije `n` puta.

```
int i = 0;
while (i < n)
{  Operacija
   i++;
}
```

Primijetite da kada *Operacija* bude izvršena `n` puta, `i` će biti jednako `n`, a uvjetni logički izraz `i < n` će biti false te će program izaći iz petlje.

Slijedi kod za određivanje sume `n` ulaznih vrijednosti.

```
System.out.println(”Unesi vrijednosti:”);

double suma = 0;
int i = 0;
while (i < n)
{  suma = suma + in.readDouble();
   i++;
}

System.out.println("Suma je " + suma);
```

Slijedi kompletan program koji zbraja brojeve koje unese korisnik. Program na početku pita koliki je broj ulaznih vrijednosti.

PRIMJER 1

```
public class Zbrajaj1

{  public static void main(String[] args)
   {  ConsoleReader in =
         new ConsoleReader(System.in);

      System.out.print
         ("Koliki je broj ulaznih vrijednosti: ");
      int n = in.readInt();

      System.out.println(”Unesi vrijednosti:”);
      double suma = 0;
      int i = 0;

      while (i < n)
      {  suma = suma + in.readDouble();
         i++;
      }

      System.out.println
         ("Suma je " + suma);
   }
}
```

2. `do-while` petlja

Java ima još jednu petlju koja je vrlo slična `while` petlji. Obično se naziva **`do-while` petlja**. (ili samo `do` petlja.) Jedina razlika između te dvije petlje je da `while` petlja ispituje logički uvjet izlaska iz petlje na početku, prije izvršavanja bilo koje naredbe unutar tijela petlje. Petlja `do-while` izvrši tijelo petlje bar jedanput jer logički uvjet izlaska iz petlje ispituje tak na kraju petlje

Oblik do-while petlje je slijedeći:

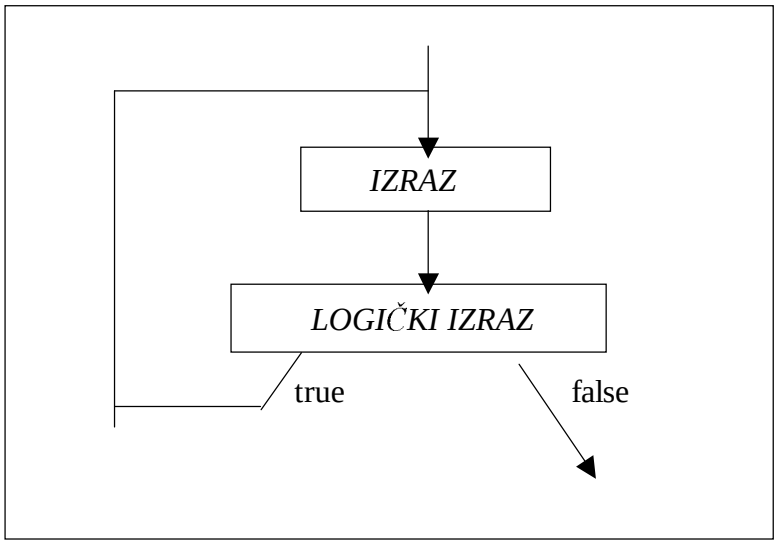
do-while petlja

```
do
    IZRAZ
while (LOGIČKI IZRAZ);
```

Što znači:

Izvršavaj u petlji *IZRAZ*.
Svaki put **nakon** što je *IZRAZ* izvršen,
Ako *LOGIČKI IZRAZ* ima vrijednost true nastavi s petljom.
Ako je false, izađi iz petlje.

Blok dijagram do-while petlje je slijedeći:



Jedina razlika je što `while` petlja ima jedan test više odnosno ispitivanje na početku petlje. Zbog toga `do-while` petlju koristimo jedino ako želimo da se tijelo petlje izvrši bar jednom.

Slijedi primjer `do-while` petlje. Program testira korisnika pitajući ga za rezultat operacije množenja dva broja.

Test da li će se test ponoviti ide na kraju i u ovom programu ćemo koristiti `do-while` petlju.

Da bismo generirali brojeve za koje ćemo postaviti upit koristit ćemo generator slučajnih brojeva. Java posjeduje klasu Random koju možemo koristiti za generiranje slučajnih brojeva. Prvo je potrebno generirati objekt tipa random:

```
Random rand = new Random();
```

Ovaj objekt posjeduje metodu `nextInt` koja vraća slučajni cijeli broj. Da bismo dobili slučajni broj u opsegu 0 do n–1, koristimo slijedeći izraz:

```
rand.nextInt(n)
```

Ako koristite `rand.nextInt(n)` više puta , svaki put ćete dobiti različite brojeve. Sekvenca brojeva koju dobijete na ovaj način je naizgled slučajna. Međutim istina je da se za dobivanje slučajnih brojeva koristi relativno jednostavna matematička formula pa po tome brojevi nisu uopće slučajni. Zbog svega ovoga brojevi proizvedeni na ovaj način nazivaju se *pseudo-slučajni* brojevi.

Slijedi primjer generiranja dva slučajna broja u opsegu 1 to 12:

```
int a = rand.nextInt(12) + 1;
int b = rand.nextInt(12) + 1;
```

Nakon ovoga slijedi kompletan program. Klasa Random nalazi se u `java.util` paketu u Java biblioteci. (`util` je skraćenica za 'utility'.)

PRIMJER 2

```
import java.util.*;

public class Multipliciranje

{  /* testiranje tablice množenja. */

   public static void main(String[] args)
   {  ConsoleReader in =
         new ConsoleReader(System.in);
      Random rand = new Random();

      System.out.println
         ("Malo mozganja za korisnika.");
      String reply;
      do
      {  /* generiraj dva pseudo-slučajna broja između 1 i 12*/
         int a = rand.nextInt(12) + 1;
         int b = rand.nextInt(12) + 1;

        /* Pitaj korisnika koliko je a*b i učitaj odgovor*/

         System.out.println
```

```
            ("Koliko je " + a + " puta " + b + "?");
         int odgovor = in.readInt();
         if (odgovor == a*b)
            System.out.println("Točno!");
         else
            System.out.println
               ("Netočno. Odgovor je " + a*b);

         /* Pitaj korisnika da li želi još pitanja*/
         System.out.println
           ("Želite li još pitanja da/ne)?");
         reply = in.readLine();
      }
      while (reply.equals("da"));

      System.out.println("Kraj");
   }
}
```

Jedno upozorenje oko `do-while` petlje. Razmotrimo slijedeću `do-while` petlju

```
do S while B;
```

Ako je S blok naredbi tada bilo koja varijabla koja je deklarirana unutar bloka S nije dostupna u logičkom izrazu B. Ako želite koristi iste varijable unutar S i B potrebno ih je deklarirati prije `do-while` petlje. (poput varijable `reply` u primjeru 2).

---

4. `for`-petlja

Java poput svih "C" jezika posjeduje **`for` petlju**. Njen opći oblik je:

`for` (*Uzmi prvu vrijednost*`;` *DokJeTočanIzraz*`;` *OdrediSlijedećuVrijednost*`)`
Operacije

For petlja u javi može koristiti sve cjelobrojne brojčane vrijednosti te brojeve u pokretnom zarezu.

Primjer petlje:

```
for (double x = 0; x <= 90; x = x+5)
   System.out.println
      ("\t" + x + "\t" + Math.cos(x));
```

Slijedi jedan od prethodnih primjera napisan korištenjem for petlje:
PRIMJER 5

```
public class Zbrajaj2

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

```
   {  ConsoleReader in =
         new ConsoleReader(System.in);

      System.out.print
         ("Koliko vrijednosti želite unijeti: ");
      int n = in.readInt();

      System.out.println("Unesi vrijednost");
      double suma = 0;
      for (int i = 0; i < n; i++)
         suma = suma + in.readDouble();
      System.out.println
         ("Suma je " + suma);
   }
}
```

5. Kontrola izvršavanja petlje - break i continue naredbe

break naredba

Za prisilni izlazak iz petlje koristimo **`break` naredbu**, koju nismo dosada susreli. Naredba `break` kaže Javi: zaustavi trenutačno izvršavanje petlje.

Npr. pišemo program za računanje kvadratnog korijena koji u beskonačnoj petlji učitava brojeve i ispisuje njihov kvadratni korijen. Kada korisnik upiše negativan broj program prestaje s radom.

```
while (true)
{  Čitaj slijedeću ulaznu vrijednost
   i pohrani je u x.
   if (x<0) break;
   Ispiši iznos kvadratnog korijena od x.
}
```

`break` naredba može se koristiti za prekid bilo koje petlje. Također se koristi u formiranju switch izraza.

Slijedi kompletan kod programa:

PRIMJER 3

```
public class KvadratniKorijen
{  /* Čitaj brojeve u pokretnom zarezu i ispisuj njihove
      kvadratne korijene.
      Završi kad korisnik upiše vrijednost <0.
   */
   public static void main(String[] args)
   {  ConsoleReader in =
         new ConsoleReader(System.in);
```

```
        System.out.println("Program za račun kv. korijena.\n" +
          "(Unesi vrijednost<0 za kraj.)");

        while (true)
        {  System.out.print("Unesi broj: ");
           double x = in.readDouble();
           if (x<0) break;
           System.out.println
             ("Kv. korijen = " + Math.sqrt(x));
        }

        System.out.println("Kraj");
     }
  }
```

continue naredba

Da bismo izbjegli trenutnu iteraciju petlje i nastavili sa slijedećom upotrebljavamo `continue` naredbu. Primjer continue naredbe je:

```
for (i=0;  i<=5;  ++i) {
   if (i % 2 == 0)
       continue;
   System.out.println("Ovo je " + i + ". iteracija");
}
```

ili

```
i = 0;
while (i <= 5)  {
      ++i;
      if (i % 2) == 0)
         continue;
      System.out.println("Ovo je neparna iteracija - " + i);
}
```

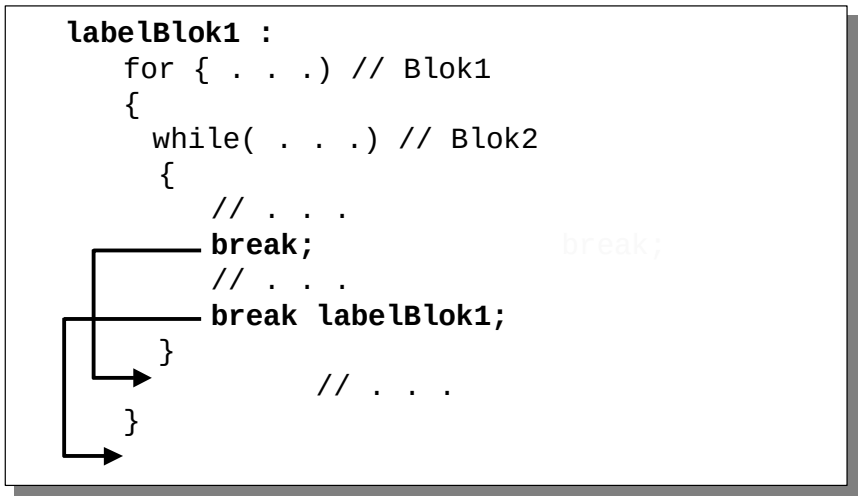
---

6. Ugniježđene petlje

Petlje možemo po volji ugnijezditi jednu u drugu. Ponekad je teško prisilno izaći iz takvih petlji. Java stavlja na raspolaganje varijantu naredbe break :

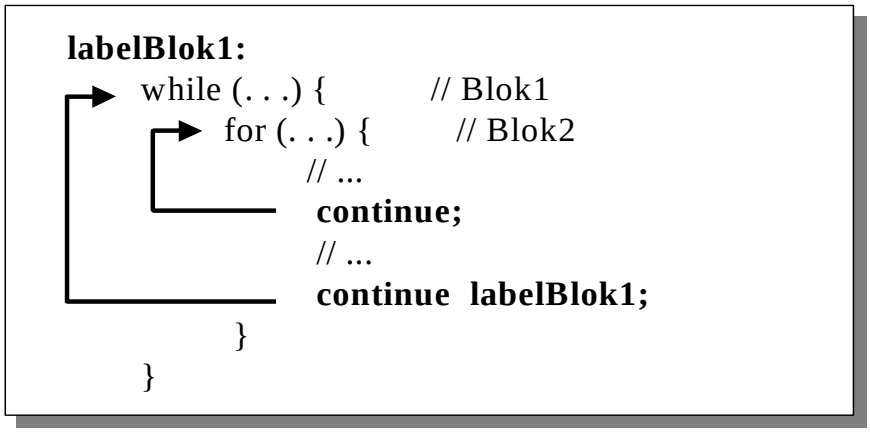
```
break labela;
```

Ova break naredbe izlazi iz petlje koja ispred sebe ima definiranu labelu naziva istog kao naziv labele naveden poslije break naredbe.



`continue` naredba ima također varijantu s labelom:

```
continue labela;
```



8. Zadaci za poglavlje 6

1. Napišite program koji će čitati cjelobrojnu vrijednost n koju unese korisnik i onda izračunati faktorijel od n te ga prikazati na ekranu. Koristite bilo koju od petlji. (Faktorijel.java)

2. Napišite program koji će izračunati i ispisati faktorijel za vrijednosti od 1 do 20. Za ovo će vam biti potrebna petlja unutar petlje. (Faktorijel2.java)

3. Napišite program koji će "zamisliti broj" od 1 do 1000. Zatim će korisnika pitati da pogodi koji je broj zamišljen. Kad korisnik unese broj program mu mora odgovoriti da li je broj veći, manji ili jednak zamišljenom broju. Ako je jednak prekida se izvršavanje uz odgovarajuću poruku (čestitka, broj pokušaja). Ako je manji ili veći onda se poslije odgovarajuće poruke korisniku nudi ponovno pogađanje.