PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

KI103 - JAVA 1: OSNOVE PROGRAMIRANJA U JAVI

Petlje

Lekcija 04

PRIRUČNIK ZA STUDENTE

KI103 - JAVA 1: OSNOVE PROGRAMIRANJA U JAVI

Lekcija 04

PETLJE

- Petlje
- Poglavlje 1: Petlja while
- Poglavlje 2: Petlja do while
- Poglavlje 3: Petlja for
- Poglavlje 4: Ugnežđavanje petlji
- Poglavlje 5: Domaći zadaci
- Zakljucak

UVOD

Petlja je iterativna kontrolna struktura programa pomoću koje se određeni skup naredbi ponavlja više puta, zavisno od uslova za izvršenje petlje.

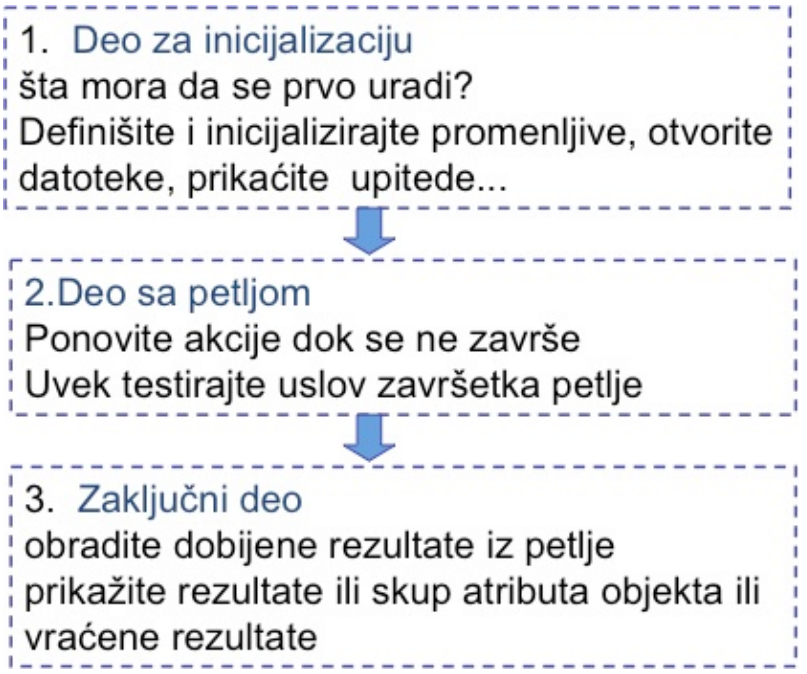
Mnogo savremenih mašina radi tako što beskonačno ponavlja isto kretanje. Motor vašeg automobila ponavlja cikluse kretanja, kojima pretvara energiju sagorevanja benzina u mehaničku. Isto važi i za električne motore. I jedna i druga vrsta mašina su korisne zato što stalno rade isto i to ponavljaju sve dok nam je potrebno.

I računarski programi u svom radu imaju cikluse. Kada program u sebi ima petlju, neki iskazi se ponavljaju sve dok ne obave posao. Većina programa, svaki put kada se izvrši, upotrebi više miliona iskaza. Obično se isti iskazi izvršavaju više puta.

Petlje se u programskim jezicima realizuju pomoću posebnih struktura. U Javi, a i većini drugih programskih jezika, postoje tri vrste petlji. To su petlje: **while, do while** i **for.**

Treba da razmišljate o sledeće tri aktivnosti kada pišete petlje u vašem programu (slika U. 1):

1. Deo za inicijalizaciju
2. Deo sa petljom
3. Zaključni deo.



Slika-1 Tipična kontrolna struktura (petlja)

Poglavlje 1

Petlja while

OPŠTI OBLIK WHILE NAREDBE

Naredbe ponavljanja while obezbeđuju izvršavanje neke (blok) naredbe više puta, sve dok je logički uslov zadovoljen, tj. true.

Naredbe ponavljanja ***while*** obezbeđuju izvršavanje neke (blok) naredbe više puta.

Broj ponavljanja te naredbe kod naredbi ponavljanja se kontroliše vrednošću jednog **logičkog izraza.**

To znači da se naredba ponavlja sve dok je vrednost tog **logičkog izraza** jednaka ***true*****,** a da se ponavljanje prekida u trenutku kada vrednost tog logičkog izraza postane **false.** Ponovljeno izvršavanje iste naredbe se može, u principu, izvoditi beskonačno, ali takva beskonačna petlja obično predstavlja grešku u programu**.**

Opšti oblik **while** naredbe je:

```
while (logičkiIzraz)
        naredba;
```

, a kada se umesto jedne, u telu patlje nalazi više naredbi, opšti oblik naredbe izgleda ovako:

```
while (logičkiIzraz){
        naredba1;
        ...
        naredbaN;
    }
```

Izvršavanj***ewhile*** naredbe se izvodi tako što se najpre izračunava vrednost logičkog izraza u zagradi. U jednom slučaju, ako je ta vrednost ***false***, odmah se prekida izvršavanje while naredbe. To znači da se preskače ostatak ***while*** naredbe i program se normalno nastavlja od naredbe koja sledi iza while naredbe.

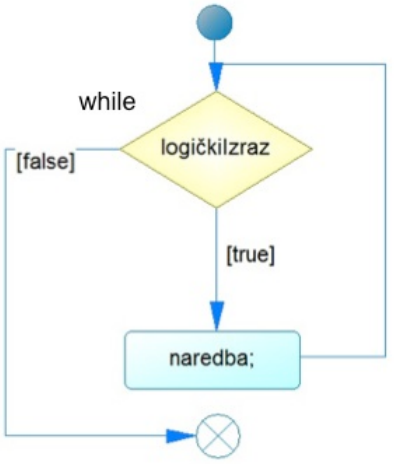
U drugom slučaju ako izračunavanje logičkog izraza daje true, najpre se izvršava naredba ili niz naredbi u bloku, a zatim se ponovo izračunava logički izraz u zagradi i ponavlja isti postupak. To jest, ako logički izraz daje ***false***, prekida se izvršavanje***while*** naredbe; ako logički izraz daje true, izvršava se naredba ili niz naredbi u bloku, opet se izračunava logički izraz u zagradi i zavisno od njegove vrednosti se ili ponavlja ovaj postupak ili se prekida izvršavanje while naredbe.

ALGORITAM PETLJE WHILE

Petlja se ponavlja sve dok je logički izraz tačan.

Efekat ***while***naredbe je ponavljanje jedne naredbe ili niza naredbi u bloku sve dok je logički izraz tačan. U trenutku kada on postane netačan, while naredba se završava i nastavlja se normalno izvršavanje programa od naredbe koja sledi iza while naredbe.

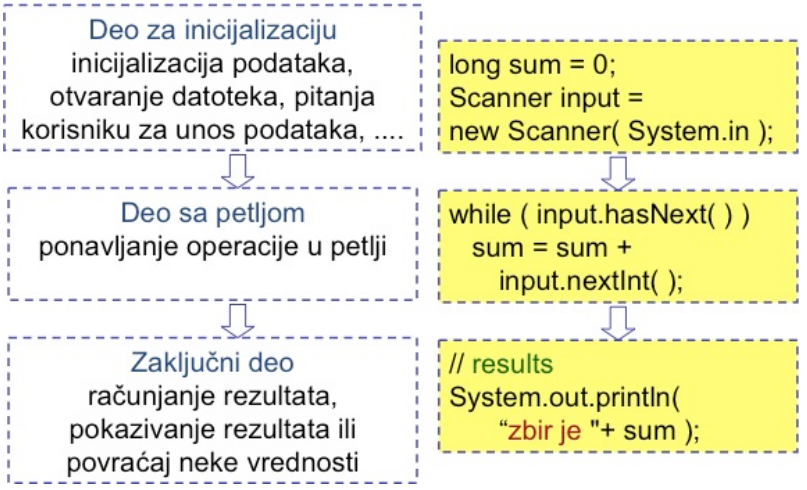
Slikoviti prikaz izvršavanja while naredbe je prikazan blok-dijagramom na slici 1 :



Slika 1.1 Algoritam petlje while

Kako se vidi sa slike, na desnoj strani blok-dijagrarna toka izvršavanja while naredbe se obrazuje petlja, pa se često govori o while petlji. Isto tako, naredba ili niz naredbi bloka unutar while petlje se naziva **telo petlje**, a jedno izvršavanje tela petlje se naziva jedna **iteracija while petlje**. Najzad, logički izraz koji se izračunava na početku svake iteracije se naziva **uslov prekida (ili nastavka) petlje**.

Obično, ponavljanja petljom while se vrše na sledeći način:



Slika 1.2 Opšti oblik ponavljanja naredbi u petlji while

DOPUNSKA RAZMATRANJA O WHILE PETLJI

Postoji više faktora koji utiču na ispravan rad while petlje.

Na kraju, razjasnimo još neke detalje u vezi sa while naredbom.

Prvo, šta se događa ako je logički izraz (tj. uslov nastavka) ***while*** petlje netačan pri prvom njegovom izračunavanju, kada se telo petlje još nijedanput nije izvršilo ? U tom slučaju, izvršavanje while petlje se odmah završava i zato se telo petlje nikad ne izvršava. To znači da broj iteracija while petlje može biti proizvoljan, uključujući nulu.

Drugo, šta se događa ako logički izraz (uslov nastavka) while petlje postane netačan negde u sredini izvršavanja tela petlje ? Da li se while petlja odmah tada završava? Odgovor je ne, nego se telo petlje izvršava do kraja. Tek kada se onda ponovo izračuna logički izraz i dobije njegova netačna vrednost, dolazi do završetka izvršavanja while petlje.

Treće, u telu while petlje se mogu nalaziti proizvoljne naredbe, pa tako i druge petlje. Kada se jedna petlja nalazi unutar tela druge petlje, tada se govori o ugnježđenim petljama.

Kada u svojim programima budete koristili petlje, treba da obratite pažnju na sledeće:

- Inicijalne vrednosti moraju biti podešene na pravi način;
- Uslov petlje mora biti tačan;
- U telu petlje se moraju menjati vrednosti promenljivih na pravi način. Ovo se odnosi na promenljive koje učestvuju u uslovu petlje. U telu petlje se mora desiti neka akcija koja menja promenljive uslova petlje, tako da taj uslov u nekom trenutku dobije vrednost netačno. Ako se to ne bi desilo, dobili bismo beskonačnu petlju.

Petlja koristi tzv. Kontrolnu promenljivu (npr. brojac) najčešće tipa int, i to i u uslovu i u telu petlje.. Ipak, ne moraju sve petlje da imaju kontrolnu promenljivu. Ovakve petlje predstavljaju poseban tip petlji, tzv. **petlje sa brojanjem**.

Svaki prolaz kroz telo petlje se naziva **iteracijom**. Kod petlji sa brojanjem u svakoj iteraciji se menja celobrojna kontrolna promenljiva petlje.

PETLJA WHILE (VIDEO)

Video preko primera objašnjava primeni while petlje

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 1: PRIKAZIVANJE CELIH BROJEVA OD 0 DO 9 U JEDNOM REDU NA MONITORU

Primer pirikazuje brojeve od 0 do 9 u jednoj liniji na monitoru.

U sledećem prostom primeru **while** naredbe se na ekranu prikazuju brojevi 0, 1, 2, ..., 9 u jednom redu**:**

```
int broj = 0;
    while (broj < 10) {
        System.out.print(broj + " ");
        broj = broj + 1;
    }
    System.out.println();
```

Ovde se na početku celobrojna promenljiva broj inicijalizuje prvom vrednošću 0.

Zatim dolazi while naredba, kod koje se najpre određuje vrednost logičkog izraza broj < 0. Vrednost ovog izraza je true, jer je trenutna vrednost promenljive broj jednaka 0 i, naravno, broj 0 je manji od broja 10. Zato se izvršavaju dve naredbe bloka u telu petlje, odnosno na ekranu se prikazuje vrednost 0 promenljive broj i ta vrednost se uvećava za 1. Nova vrednost promenljive broj je dakle 1.

Sledeći korak kod izvršavanja while naredbe je ponovno izračunavanje logičkog izraza broj < 10. Vrednost ovog izraza je opet true, jer je trenutna vrednost promenljive broj jednaka 1 i, naravno, broj 1 je manji od broja 10. Zato se opet izvršavaju dve naredbe u telu petlje, odnosno na ekranu se prikazuje vrednost 1 promenljive broj i ta vrednost se uvećava za 1. Nova vrednost promenljive broj je dakle 2.

Pošto izvršavanje***while*** naredbe još nije završeno, ponovo se izračunava logički izraz broj < 10. Opet se dobija njegova vrednost true, jer je trenutna vrednost promenljive broj jednaka 2 i, naravno, broj 2 je manji od broja 10. Zato se opet izvršava telo petlje, izračunava se uslov prekida petlje.

U svakoj iteraciji petlje se prikazuje trenutna vrednost promenljive broj i ta vrednost se uvećava za 1, pa je jasno je da će se to ponavljati sve dok promenljiva broj ne dobije vrednost 10. U tom trenutku će logički izraz broj < 10 imati vrednost false, jer 10 nije manje od 10, a to predstavlja uslov prekida izvršavanja while naredbe. Nakon toga se izvršavanje programa nastavlja od naredbe koja se nalazi iza while naredbe, što u našem primeru dovodi do pomeranja kursora na ekranu u novi red.

PRIMER 2: PETLJA SA BROJAČEM

Kontrolna promenljiva povećava vrednost za 1 posle svakog prolaza petlje, tj. posle svake iteracije.

Da objasnimo primer na slici 3. Da objasnimo sada kako ovo funkcioniše:

1. Promenljivoj brojac se dodeljuje vrednost Izračunava se izraz (brojac <= 3) i njegova vrednost je true.
2. Pošto je izraz tačan izvršava se blok iza petlje:
3. Izračunava se izraz (brojac <= 3) i njegova vrednost je ponovo tačno.
4. Ponovo se izvršava blok iza iskaza while:.
5. Ponovo se izvršava blok iza iskaza while:

6. Izračunava se izraz (brojac <= 3) i njegova vrednost je sada netačno.
7. Blok iza iskaza while se više ne izvršava.
8. Izvršava se iskaz iza petlje:

Primer:

```
class Petlje{
   public static void main (String[] args )   {
        int brojac = 1;  // brojanje počinje od 1
        while (brojac <= 3 ) {
            System.out.println( "Brojac je:" + brojac);
            brojac = brojac + 1; // brojac se povecava za 1
        }
        System.out.println( "Iza petlje" );
    }
}
```

ZADATAK 1

Prikazivanje imena n puta na ekranu pomoću while petlje.

Kreirajte program koji primenom while petlje ispisuje vaše ime n puta na ekranu. Ime se ispisuje u istom redu sa jednostrukim razmakom između dva zapisa. Broj n se unosi sa tastature.

PRIMER 3: POGAĐANJE BROJA OD 0 DO 100

Igra pogađanja celog broja od 0 do 100.

U sledećem primeru je prikazan program kojim se simulira igra pogađanja broja sa korisnikom - program "zamišlja" slučajni ceo broj između 1 i 100, a korisnik treba da ga pogodi.

Ako korisnik nije pogodio zamišljeni broj u jednom pokušaju, onda program treba da pomogne korisniku za sledeći pokušaj prikazujući poruku da je zamišljeni broj manji ili veći od pokušanog, kao i da omogući korisniku ponovno pogađanje zamišljenog broja, pri čemu ovaj dijalog treba da se ponavlja sve dok korisnik ne pogodi zamišljeni broj.

Logička promenljiva pogodak je potrebna da bi se prekinulo izvršavanje while petlje kad korisnik pogodi zamišljeni broj. S obzirom da se uslov prekida petlje proverava na početku svake iteracije, neophodno je inicijalizovati promenljivu pogodak na vrednost false na početku, kako bi se prva iteracija sigurno izvršila.

```
import java.util.Scanner;

    public class PogadjanjeBroja {
        public static void main(String[] args) {
            int zamisljeniBroj;
            int pokusanBroj;
```

```
            boolean pogodak;
            Scanner ulaz = new Scanner(System.in);
            zamisljeniBroj = 1 + (int) (Math.random() * 100);
            pogodak = false;
            while (!pogodak) {
                System.out.println("Pogodite broj izmedju 1 i 100: \n");
                pokusanBroj = ulaz.nextInt();
                if (pokusanBroj < zamisljeniBroj) {
                    System.out.println("Zamisljeni broj je veci.");
                } else if (pokusanBroj > zamisljeniBroj) {
                    System.out.println("Zamisljeni broj je manji.");
                } else {
                    System.out.println("Pogodili ste broj!.");
                    pogodak = true;
                }
            }
        }
    }
```

PRIMER 4

While petlja sa više namena

Napisati program koji pokazuje tabelu konverzije milja u kilometre od 1 do 100 i tabelu konverzije kilometara u milje od 20 do 515 tako da se broj kilometara svaki put poveća za 5. Treba koristiti jednu while petlju. Program treba da bude deo NetBeans projekta pod nazivom KI103-PR4 i deo paketa primer4. Naziv klase treba da bude Primer4.

Miles	Kilometers	Kilometers	Miles
1	1.609	20	12.430
2	3.218	25	15.538
...			
100	160.900	515	320.075

Slika 1.3 rezultujuće tabele

Objašnjenje:

Kod ovog zadatka, ideja je da formiramo tabelu sa slike, tako da prva kolona označava milje od 1 do 100, druga njihovu odgovarajuću vrednost u kilometrima. Treća kolona predstavlja brojeve od 20 do 515 gde je svaki naredni broj uvećan za 5. Četvrta kolona prikazuje odgovarajuću vrednost prethodne kolone, izraženo u miljama. U ovom slučaju koristimo while petlju koja ima samo jedan uslov kao parametar i definiše da se blok koda koji sledi izvršava sve dok je uslov ispunjen.

Kako je broj iteracija u oba slučaja isti, uslov koji postavljamo je da je brojač došao do vrednosti 100, pri čemu je pre while petlje brojač miles postavljen na 1 i u svakom prolasku se vrednost povećava za 1. Ovaj brojač će ujedno poslužiti i za predstavljanje prve kolone. Za potrebe prikaza treće kolone i konverzije iste i prikaz u četvrtoj koloni, definišemo i brojač

kilometers čija je početna vrednost pre while petlje jednaka 5 i u svakoj iteraciji uvećava za 5. Na osnovu ova dva brojača, formiramo zadatu tabelu.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package primer4;

/**
 *
 * @author razvoj
 */
public class Primer4 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer4();
    }

    public Primer4() {
        /**
         * %7s i %12s označavaju prikaz stringova, gde se za prikaz izdvajaju 7,
         * odnosno 12 karaktera
         */
        System.out.printf("%7s %12s | %12s %7s  \n", "Miles", "Kilometers",
                "Kilometers", "Miles");
        System.out.println("-----------------------------------------------");

        int miles = 1;
        int kilometers = 20;
        while (miles <= 100) {
            System.out.printf("%7d %12.3f | %7d %12.3f \n", miles, miles * 1.609,
                    kilometers, kilometers / 1.609);
            miles++;
            kilometers += 5;
        }
    }

}
```

PRIMER 5

Korišćenje petlje za pronalazak najboljeg rezultata uz JOptionPane

Napisati program koji zahteva od korisnika da unese broj studenata, a potom da unese i informacije o tim studentima: njihovo ime kao i prosečnu ocenu. Nakon toga, program treba

da ispiše koji od studenata ima najveću prosečnu ocenu. Za razliku od prethodnog zadatka u ovom zadatku treba koristiti JOptionPane i while petlju. Program treba da bude deo NetBeans projekta pod nazivom KI103-PR5 i deo paketa zadatak4. Naziv klase treba da bude Primer5.

Objašnjenje:

U ovom primeru koristimo komponentu JOptionPane za unos imena studenta i while petlju umesto brojačke for petlje. Cilj je da se blok while petlje izvrši onoliko puta koliko ima studenata. Zato se brojač brojStudenata u svakom prolasku smanjuje, a uslov je da je brojStudenata bude veći od 0.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package primer5;

import javax.swing.JOptionPane;

/**
 *
 * @author razvoj
 */
public class Primer5 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer5();
    }

    public Primer5() {
        int brojStudenata = Integer.parseInt(JOptionPane.showInputDialog("Unesite
broj studenata: "));

        if (brojStudenata > 0) {
            double najvecaOcena = 0;
            String najboljiStudent = "";
            while (brojStudenata > 0) {
                String ime = JOptionPane.showInputDialog("Unesite ime studenta: ");
                double ocena =
Double.parseDouble(JOptionPane.showInputDialog("Unesite prosečnu ocenu: "));

                if (ocena > najvecaOcena) {
                    najboljiStudent = ime;
                    najvecaOcena = ocena;
                }
                brojStudenata--;
            }
            JOptionPane.showMessageDialog(null, "Najbolji student je " +
```

```
najboljiStudent + " sa prosečnom ocenom  " + najvecaOcena);
        } else {
            JOptionPane.showMessageDialog(null, "Broj studenata mora biti broj veći
od nule!");
        }
    }

}
```

ZADATAK 2

Broj parnih brojena od a do b pomoću while petlje.

Zadatak 1- 1:

Kreirajte program koji koristi while petlju za prebrojavanje parnih brojeva na intervalu (a,b). Brojevi a i b se zadaju sa tastature. Prevideti i slučaj kada je a>b. Tada petlja broji unazad, sve ostalo je isto.

Poglavlje 2

Petlja do while

OPŠTI OBLIK DO-WHILE PETLJE

Kod do-while petlje uslov za ponavljanje petlje se proverava na kraju petlje, za razliku od while pelje.

Kod ***while*** petlje uslov prekida petlje se proverava na početku svake iteracije. Međutim, ponekad je prirodnije proveravati taj uslov na kraju svakog izvršavanja tela petlje. U takvim slučajevima se može koristiti do-while naredba koja je vrlo slična while naredbi, osim što su reč while i uslov prekida pomereni na kraj, a na početku se nalazi reč do.

Opšti oblik **do-while petlje je:**

```
do
        naredba;
    while (logičkiIzraz);
```

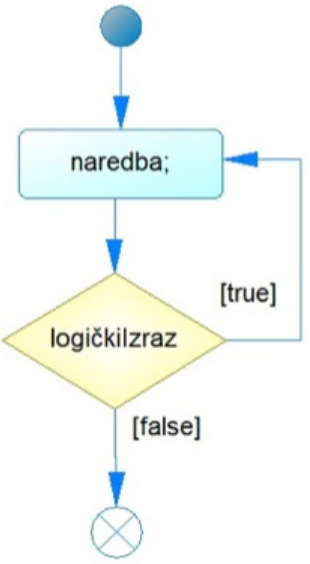
Ako telo petlje sadrži više naredbi, onda opšti oblik ove petlje ima sledeći oblik:

```
do{
        naredba1;
        ...
        naredbaN;
    }
    while (logičkiIzraz);
```

Znak za tačku-zarez do-while je deo do-while naredbe i ne može se izostaviti.

Kod izvršavanja do-while naredbe se najpre izvršava telo petlje, odnosno naredba ili niz naredbi u bloku. Zatim se izračunava

vrednost logičkog izraza u zagradi. U jednom slučaju, ako je ta vrednost false prekida se izvršavanje do-while naredbe i nastavlja se normalno izvršavanje programa od naredbe koja sledi iza reči while. U drugom slučaju, ako izračunavanje logičkog izraza daje true, izvršavanje se vraća na početak tela petlje i ponavlja isti postupak od početka. Ovo je predstavljeno blok-dijagrarnom na slici:



Slika 2.1 Algoritam do-while petlje

SPECIFIČNOSTI DO-WHILE PETLJE

Pošto se logički izraz kod do-while petlje izvršava na kraju petlje, to se telo petlje izvršava bar jedanput.

Primetimo da, pošto se logički izraz izračunava na kraju iteracije, telo petlje se kod **do-while** naredbe izvršava bar jedanput. To se razlikuje od while naredbe gde se telo petlje ne mora uopšte izvršiti. Napomenimo i da se do-while naredba može uvek simulirati while naredbom (a i obratno), jer je efekat do-while naredbe ekvivalentan sledećem programskom fragmentu:

```
naredba;
    while (logičkiIzraz)
        naredba;
```

Da bismo se uverili da je ponekad prirodnije koristiti do-while petlju umesto while petlje, razmotrimo primer programa za igru pogađanja broja. U tom primeru smo morali da veštački uvodimo promenljivu pogodak na osnovu koje smo prekidali igru ili nastavljali sa pogađanjem zamišljenog broja. Ako koristimo do-while petlju, ova veštačka promenljiva nam više nije potrebna, jer svakako moramo proći bar kroz jednu iteraciju i na njenom kraju znamo da li je zamišljeni broj pogođen ili ne.

:

Na primer, relevantni deo nove verzije ovog programa je

```
do {
        System.out.println("Pogodite broj izmedju 1 i 100: \n");
        pokusanBroj = ulaz.nextInt();
        if (pokusanBroj < zamisljeniBroj) {
            System.out.println("Zamisljeni broj je veci.");
```

```
        } else if (pokusanBroj > zamisljeniBroj) {
            System.out.println("Zamisljeni broj je manji.");
        } else {
            System.out.println("Pogodili ste broj!.");
            pogodak = true;
        }
    } while (pokusanBroj != zamisljenBroj);
```

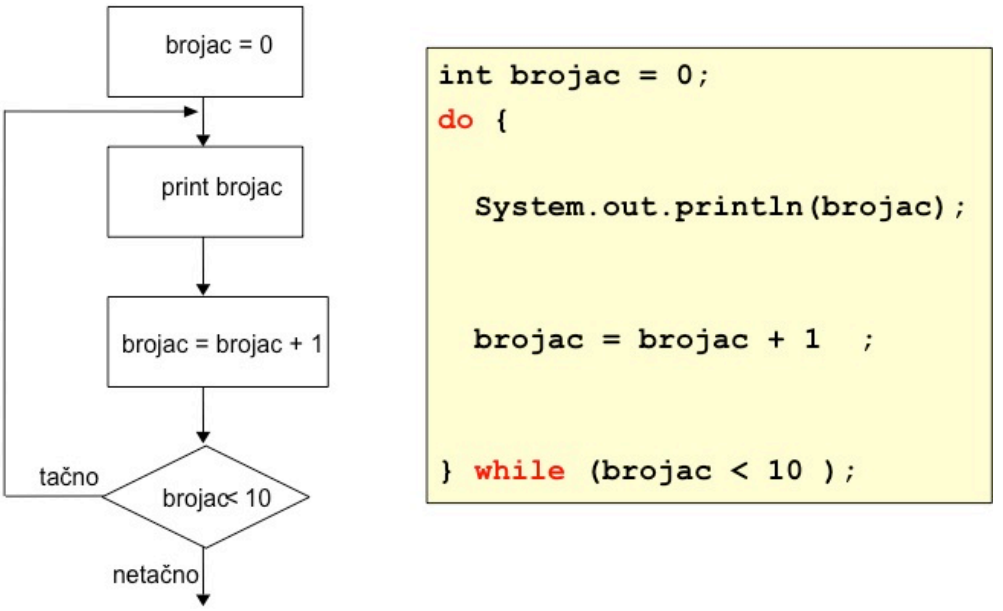
PRIMER 6: ŠTAMPANJE BROJEVA OD 0 DO 9 POMOĆU DO WHILE PETLJE

Telo petlje se izvršava makar jednom.

Primer prikazuje petlju koja štampa cele brojeve od 1 do 9

Najbitnije što treba imati na umu kod *do while petlji jeste da se telo petlje izvršava makar jednom.* Tek nakon prvog izvršavanja se proverava uslov petlje.

```
int brojac = 0;  // brojac se incijalizuje na 0
do {
  System.out.println(brojac); // telo petlje: sadrži
i
       // kod za  promenu brojaca
  brojac = brojac + 1  ;
} while (brojac < 10 ); // proverava se da li telo petlje
        //treba ponovo da se izvrši
```



Slika 2.2 Petlja Do While

ZADATAK 3

Samostalno vežbanje petlje do while

Samostalno kreirajte nov Java program za pogađanje broja od 0 do 100 primenom petlje do while. Kreirajte nov brojač, na primer brojPokusaja, koji će kao rezultat vratiti iz koliko pokušaja je pogođen zamišljeni broj. Prilikom izrade zadatka možete se koristiti primerom PRIMER 3 obrađenim za petlju while.

ZADATAK 4

Ovi zadaci imaju namenu da sami sebe proverite. Ne donose Vam poene, ali pomažu da utvrdite svoje razumevanje novostečenog znanja.

3-1. Pretpostavio da se unose sledeći podaci u ovaj program: 2 3 4 5 0 . Koji rezultat daje sledeći program?

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int number, max;
    number = input.nextInt();
    max = number;
    do {
    number = input.nextInt();
    if (number > max)
        max = number;
    } while (number != 0);
    System.out.println("max is " + max);
    System.out.println("number " + number);
  }
}
```

3-2. Koje su razlike između petlje while i petlje do-while? Promenite petlju while u sledećem delu jednog programa sa petljom do-while.

```
Scanner input = new Scanner(System.in);
int sum = 0;
System.out.println("Enter an integer " +
  "(the input ends if it is 0)");
int number = input.nextInt();
while (number != 0) {
  sum += number;
System.out.println("Enter an integer " +
  "(the input ends if it is 0)");
```

```
number = input.nextInt();
}
```

Poglavlje 3

Petlja for

OPŠTI OBLIK PETLJE FOR

Petlja for se obično koristi kada je potrebno izvršiti telo petlje za sve vrednosti određene promenljive u nekom intervalu.

Treća vrsta petlji u Javi se obično koristi kada je potrebno izvršiti telo petlje za sve vrednosti određene promenljive u nekom intervalu. Za prikazivanje brojeva 0, 1, 2, ..., 9 u jednom redu na ekranu, na primer, koristili smo naredbe while i do-while. U tim naredbama se zapravo telo petlje sastoji od prikazivanja promenljive broj, a ovo telo petlje se izvršava za sve vrednosti promenljive broj u intervalu od 0 do 9. U ovom slučaju je mnogo prirodnije koristiti naredbu for, jer je odgovarajući ekvivalentni programski fragment mnogo kraći i izražajniji:

```
for (int broj = 0; broj < 10; broj = broj + 1)
        System.out.print(broj + " ");
    System.out.println(); // pomeranje kursora u novi red
```

Opšti oblik ***for*** naredbe je:

```
for (kontrolniDeo)
        naredba;
```

Ili,u slučaju da telo petlje ima više naredbi, opšti oblik izgleda ovako:

```
for (kontrolniDeo){
        naredba1;
        ...
        naredbaN;
    }
```

Kod for naredbe je kontrolnim delom na početku naredbe obuhvaćeno na jednom mestu sve što je bitno za upravljanje petljom, a to doprinosi čitljivosti i boljem razumevanju logike petlje.

Ovaj kontrolni deo je dalje podeljen u tri dela koji se međusobno razdvajaju tačka-zapetom:

```
inicijalizacija; logičkiIzraz; završnica
```

Prema tome, opšti oblik (proste) naredbe for je zapravo:

```
for (inicijalizacija; logičkiIzraz; završnica)
        naredba;
```

IZVRŠENJE FOR PETLJE

U kontrolnom delu for naredbe, u delu inicijalizacija obično se nekoj promenljivoj dodeljuje početna vrednost, a u delu završnica se vrednost te promenljive uvećava ili smanjuje za određen

Efekat izvršavanja ove for naredbe se može predstaviti sledećim programskim fragmentom u kome se koristi while naredba

```
inicijalizacija;
    while(logičkiIzraz){
        naredba;
        završnica;
    }
```

Kod izvršavanja for naredbe dakle, na samom početku se izvršava deo inicijalizacija kontrolnog dela, i to samo jedanput. Dalje se uslov nastavka for petlje predstavljen logičkim izrazom izračunava pre svakog izvršavanja tela for petlje, a izvršavanje petlje se prekida kada izračunata vrednost logičkog izraza jeste netačno. Deo završnica kontrolnog dela se izvršava na kraju svakog izvršavanja tela for petlje, neposredno posle izvršavanja tela petlje i pre ponovne provere uslova nastavka petlje.

Deo inicijalizacija može biti bilo koji izraz, mada je to obično naredba dodele.

Deo završnica može takođe biti bilo koji izraz, mada je to obično naredba inkrementiranja, dekrementiranja ili dodele. Interesantno je primetiti da svaki od tri dela u kontrolnom delu for naredbe može biti prazan (ali se tačka-zapete ne mogu izostaviti). Ako je logički izraz u kontrolnom delu izostavljen, podrazumeva se da umesto njega stoji true. Zato for naredba sa "praznim" logičkim izrazom obrazuje beskonačnu petlju.

U kontrolnom delu for naredbe, u delu inicijalizacija obično se nekoj promenljivi dodeljuje početna vrednost, a u delu završnica se vrednost te promenljive uvećava ili smanjuje za određen korak. Pri tome se vrednost te promenljive još proverava u logičkom izrazu radi nastavka ili prekida petlje.

Promenljiva koja se na ovaj način koristi u for naredbi se naziva kontrolna promenljiva ili kratko brojač petlje.

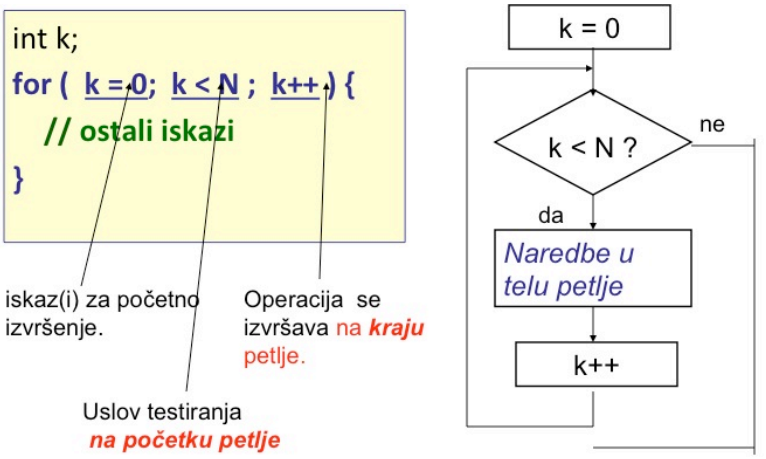
Kako inicijalizacija tako i završnica u kontrolnom delu for naredbe mogu se sastojati od nekoliko izraza razdvojenih zapetama. Tako se u sledećem primeru istovremeno prikazuju brojevi od 0 do 9 i od 9 do 0 u dve kolone:

```
for (int n = 0, int m = 9; n < 10; n++, m-- ) {
        System.out.printf("%5d", n); // kolona širine 5 mesta za n
        System.out.printf("%5d", m); // kolona širine 5 mesta za m
        System.out.println;         // prelazak kursora u novi red
    }
```

PRIMER IZVRŠENJA PETLJE FOR

Uslov petlje for se testira na početku petlje, slično kao i petlja while.

Slika 1 prikazuje način izvršenja petlje for:



Slika 3.1 Način definisanja i rada petlje for

Sledeći primer prikazuje jednostavnu for petlju:

```
for ( brojac = 0; brojac < 25;  ){
  System.out.println("Brojac je: " + brojac);
  brojac = brojac + 1;
}
```

Kod petlje for uslov petlje se testira pre prve iteracije petlje:

```
int k, N;
System.out.print(“Unestite N:");
N = scanner.nextInt( );
for (  k = 0;  k < N ;  k++ ) {
 System.out.println(“sada je k = "+k);
}
System.out.println(“završeno. k = "+k);
```

PRIMER 7

Pretvoriti milje u kilometre za vrednost milja od 1 do 100 primenom for petlje

Napisati program koji će za sve vrednosti redom od 1 do 100 milja prikazati odgovarajuću vrednost u kilometrima. Jedna milja iznosi 1.60934 kilometara. Program treba da bude deo NetBeans projekta pod nazivom KI103-PR7 i deo paketa primer7. Naziv klase treba da bude Primer7.

Objašnjenje:

Kako je potrebno odraditi preračun za svaki broj od 1 do 100, potrebno je koristiti for petlju. For petlja ima 3 bitna parametra. Prvi parameter je početni brojač (int i=0) što označava početnu vrednost brojača. Drugi parametar predstavlja uslov do kada će se ono što je u bloku petlje ponavljati, dok treći parametar predstavlja ono što će se desiti na kraju svakog ponavljanja (u ovom slučaju *i* se povećava za 1 svaki put kada se petlja ponovi i na taj način obezbeđuje da se petlja izvršava redom za vrednosti i=1, 2, 3 ..., 100).

Svaka iteracija for petlja izvršava blok programskog koda od { do }, konkretno ovde je to štampanje i preračun milja u kilometre.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package primer6;

/**
 *
 * @author razvoj
 */
public class Primer7 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer6();
    }

    public Primer7() {

        for (int i = 1; i <= 100; i++) {
            System.out.printf("%d miles = %.5f km \n", i, i * 1.60934);

        }
    }

}
```

UPOREĐENJE PETLJE FOR SA PETLJOM WHILE

Isti problem se može rešiti i petljom for i petljom while.

Na slici 2 prikazano je upoređenje opštih oblika izražavanja petlji for i while:

for petlja	<==>	while petlja

for (incijalizacija;
uslov_petlje;
izraz_petlje)
  teloPetlje ;
		incijalizacija;
		while(uslov_petlje){
		  teloPetlje;
		  izraz_petlje
		}

Slika 3.2 Upoređenje opštih oblika petlji for i while

Na slici 3 prikzani je isti primer urađen pomoću for petlje, i while petlje

Ova dva programa su ekvivalentna. Na primeru while petlje se može bolje uočiti kako funkcioniše for petlja. Brojač (count) se podešava na početku rada petlje. Inicijalizacija se odvija samo jednom. Uslov petlje se stalno ocenjuje, da bi se videlo da li telo petlje treba da se ponovo izvrši. Izraz petlje se izračunava uvek na kraju tela petlje.

:

for petlja	while petlja

int count, sum;
 	
sum = 0;
for(count=0;count<=5;count
=count+1){
  sum = sum + count ;
  System.out.print(count
+ " ");
}
System.out.println("suma
je: " + sum);
	int count, sum;
	sum = 0;
	count = 0;
	while (count <= 5){
	  sum = sum + count ;
	  System.out.print(count
	+ " ");
	  count=count+1;
	}
	System.out.println("suma
	je: " + sum);

Slika 3.3 Isti primer rešen primenom petlje for i petlje while

DOMEN KONTROLNE PROMENLJIVE PETLJE

Domen kontrolne promenljive su granice petlje u kojoj je promenljiva deklarisana.

Već smo pomenuli da se for petlje koriste najviše kod petlji sa brojanjem kod kojih postoji kontrolna promenljiva, na osnovu koje se određuje da li petlja treba da se ponovo izvršava. Domen ove promenljive su granice petlje u kojoj je promenljiva deklarisana. Ovo naravno važi ako je ta promenljiva deklarisana u delu inicijalizacije petlje.

Pogledajmo sledeći primer:

```
int brojac;
. . . . . .
for (brojac = 0;  brojac < 10; brojac ++ )
  System.out.print(brojac + " " );
```

Kontrolna promenljiva petlje brojac je deklarisana negde u programu (možda i daleko ispred petlje). Ovim se narušava ideja o kontroli petlje u samo jednom iskazu. To se i može uraditi na sledeći način:

```
for ( int brojac = 0;  brojac < 10; brojac ++ )
  System.out.print(brojac + " " );
```

Ovaj drugi način je bolji i treba ga koristiti kad god je moguće (poželjno, skoro uvek).

Ako u delu inicijalizacije for petlje deklarišete promenljivu petlje, onda je sledeće neispravno:

```
for ( int brojac = 0;  brojac < 10; brojac ++ )
  System.out.print(brojac + " " );

// Nije deo tela petlje
System.out.println( "\nPosle petlje brojac je:" + brojac);
```

Pošto iskaz println() nije deo tela petlje, u njemu se ne može koristiti promenljiva brojac. Kompajler će javiti grešku tipa "cannot resolve symbol", što znači da brojac nije dostupan

PRIMER 8

Korišćenje for petlje za pronalazak najboljeg rezultata

Napisati programkoji zahteva od korisnika da unese broj studenata, a potom da unese i informacije o tim studentima: njihovo ime kao i prosečnu ocenu. Nakon toga, program treba daispiše koji od studenata ima najveću prosečnu ocenu. Zadatak treba uraditi korišćenjem Scanner klase. Program treba da bude deo NetBeans projekta pod nazivom KI103-PR8 i deo paketa primer8. Naziv klase treba da bude Primer8.

Objašnjenje:

U ovom slučaju je lakše koristiti for petlju kao brojač jer broj prolazaka zavisi od broja studenata.

Za svaki prolazak korisnik unosi novog studenta, odnosno njegovo ime i prosečnu ocenu, a program proverava uslovom da li je uneta ocena veća od najveće ocene do sada. Pre prvog prolaska, definišemo da je najveća ocena 0. Ukoliko jeste najveća ocena postaje uneta ocena.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package primer8;
```

```
import java.util.Scanner;

/**
 *
 * @author razvoj
 */
public class Primer8 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer8();
    }

    public Primer8() {
        Scanner input = new Scanner(System.in);
        System.out.print("Unesite broj studenata: ");
        int brojStudenata = input.nextInt();
        if (brojStudenata > 0) {
            double najvecaOcena = 0;
            String najboljiStudent = "";
            for (int i = 0; i < brojStudenata; i++) {
                System.out.print("Unesite ime studenta: ");
                String ime = input.next();
                System.out.print("Unesite prosečnu ocenu studenta: ");
                double ocena = input.nextDouble();
                if (ocena > najvecaOcena) {
                    najboljiStudent = ime;
                    najvecaOcena = ocena;
                }
            }
            System.out.printf("Najbolji student je %s i njegova prosečna ocena je
%.2f. \n", najboljiStudent, najvecaOcena);
        } else {
            System.out.println("Broj studenata mora biti broj veći od nule!");
        }
    }

}
```

ZADATAK 5

Samostalno vežbanje korišćenja for petlje

Modifikujte prethodni primer i omogućite mu funkcionalnost po kojoj program vraća, kao deo rezultata, broj studenata koji imaju prosek veći od 8.

DECIMALNI BROJEVI KAO BROJAČI PETLJE

Za kontrolnu promenljivu for petlje možete da koristite i decimalni broj (tipa real), koji biste u svakom prolazu povećavali opet za neki decimalni broj.

Kod petlji sa brojanjem je najbolje da se kao kontrolna promenljiva petlje koristi celobrojna promenljiva. Ipak, ako želite, možete da koristite i decimalni broj (tipa real), koji biste u svakom prolazu povećavali opet za neki decimalni broj. U sledećem primeru smo pokazali primer koji štampa x i ln(x) za vrednosti x od 0.1 do 2.0.

```
class Logaritam{
  public static void main ( String[] args )  {
    System.out.println( "x" + "\t ln(x)" );

    for ( double x = 0.1; x <= 2.0; x = x + 0.1 )
    System.out.println( x + "\t" + Math.log( x ) );
  }
}
.
```

Ovaj program će se iskompajlirati i radiće svoj posao, ali ne baš na najbolji način. Izlaz iz njega je dat na slici 4 .:

```
x                    ln(x)
0.1                  -2.3025850929940455
0.2                  -1.6094379124341003
0.30000000000000004  -1.203972804325936
0.4                  -0.916290731874155
0.5                  -0.6931471805599453
0.6                  -0.5108256237659907
0.7                  -0.35667494393873245
0.7999999999999999   -0.22314355131420985
0.8999999999999999   -0.1053605156578264
0.9999999999999999   -1.1102230246251565E-16
1.0999999999999999   0.09531017980432474
1.2                  0.1823215567939546
1.3                  0.26236426446749106
1.4000000000000001   0.336472236621213
1.5000000000000002   0.40546510810816455
1.6000000000000003   0.47000362924573574
1.7000000000000004   0.5306282510621706
1.8000000000000005   0.5877866649021193
1.9000000000000006   0.641853886172395
```

Slika 3.4 Prikaz rezultata izvršenja programa na slici 3

PRIMER 9 - UPOTREBA NAREDBE FOR ZA ODREĐIVANJE PROSTOG BROJA

U ovom primeru je prikazan program kojim se određuje da li je ulazni ceo broj prost.

Ceo broj veći od jedan je prost ako je deljiv bez ostatka samo sa samim sobom i jedinicom. Početak niza prostih brojeva je: 2, 3, 5, 7, 11, 13... U ovom primeru je prikazan program kojim se određuje da li je ulazni ceo broj prost.

Ako je n ulazni broj, postupak kojim se određuje da li je n prost broj je da se svi brojevi od 2 do n - 1 redom provere da li dele n bez ostatka - ako se pronađe bar jedan takav delilac, n nije prost, u suprotnom broj je prost. Postupak se može skratiti ako primetimo da ako broj n ima delioca d većeg od 1 i manjeg od n, onda ima i delioc n/d koji je manji ili jednak korenu od n. Dakle, dovoljno je proveravati delioce broja n iz intervala od 2 do korena iz n.

```
import java.util.*;

    public class ProstBroj {
        public static void main(String[] args) {
            int n;
            Scanner ulaz = new Scanner(System.in);
            System.out.println("Unesite ceo broj veci od 1: ");
            n = ulaz.nextInt();
            int maksimalniDelilac = (int) Math.sqrt(n);
            for (int i = 2; i <= maksimalniDelilac; i++) {
                if (n % i == 0) {
                    System.out.println("Broj nije prost.");
                    System.exit(0);
                }
            }
            System.out.println("Broj je prost.");
        }
    }
```

KONTROLNE PROMENLJIVE KAO ČLANOVI NIZA

Vrednosti kontrolne promenljive petlje for zadate su preko članova niza, pri čemu redni broj u nizu odgovara rednom broju iteracije petlje.

Uobičajeno je da kontrolna promenljiva for petlje, u svakoj interaciji, menja svoju vrednost za jedanu konstatnu vrednost, najčešće za 1. Međutim, moguće je koristiti i for petlje u kojoj je vrednost kontrolne promenljive u svakoj iteraciji različita i može da joj se zada bilo koja vrednost. Te vrednosti se definišu u jednodimenzonom nizu vrednosti, gde mesto u nizu definiše redni broj iteracije, a vrednost člana niza definiše vrednost kontrolne promenljive u određenoj iteraciji.

Opšti oblik for petlje u ovom slučaju je:

```
for (int i : niz)
```

Ovaj oblik for petlje dodeljuje brojaču petlje i u svakoj iteraciji inkrement broj koji predstavlja jedan element datog niza brojeva.

U svakoj iteraciji, vrednost inkrementa je jednaka vrednosti sledećeg elementa niza, tj. elementu koji ne mestu u nizu koji odgovara rednom broju iteracije.

PRIMER 10

Kontrolna promenljiva kao član niza

U sledećem primeru, vrednosti kontrolne promenljive petlje, su zadate u vidu članova niza.

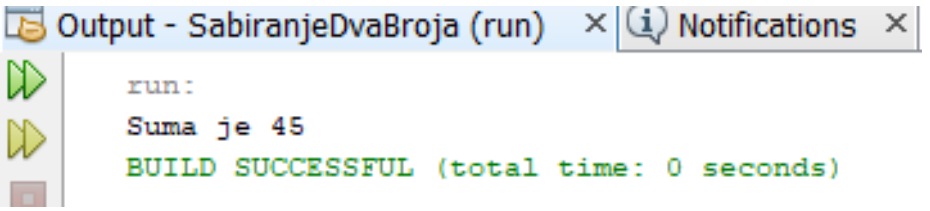
```
package l02;

/**
 * *
 * @author Vladimir Milićević
 */
public class KontrolnaPromenljiva {

    public static void main(String[] args) {
        int[] niz = {2, 4, 5, 7, 8, 9, 10};
        int suma = 0;
        for (int i : niz) {
            suma += i;
        }
        System.out.println("Suma je " + suma);

    }

}
```



Slika 3.5 Rezultat izvršavanja programa

ZADATAK 6

Samostalno vežbanje primene kontrolne promenljive kao člana niza

U prethodnom primeru kao rezultat vratiti sledeće:

- broj parnih brojeva u nizu;
- broj neparnih brojeva u nizu;
- zbir parnih brojeva u nizu;
- zbir neparnih brojeva u nizu.

PETLJA FOR (VIDEO)

Video objašnjava korišćenje petlje for.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 4

Ugneždavanje petlji

PETLJA U PETLJI

Unutar tela neke petlje može da postoji druga petlja, pa unutar te druge – treća itd

.. To znači da Petlje mogu da se postave i unutar neke već postojeće petlje, tj. unutar tela neke petlje može da postoji druga petlja, pa unutar te druge – treća itd. Kod ovakvih petlji pravilo je da se uvek prvo izvršava krajnja unutrašnja petlja, pa onda ona u kojoj se ova nalazi i tako redom.

Kod ovako umetnutih petlji nije bitno koji su tipovi petlji. To znači da možete proizvoljno kombinovati petlje while, for i do while.

S obzirom na to da se petlje mogu ugnježđavati jedna u drugu, ukoliko se break naredba nalazi u unutrašnjoj petlji, postavlja se pitanje koju petlju prekida break naredba - unutrašnju ili spoljašnju ? Pravilo je da break naredba uvek prekida izvršavanje samo prve obuhvatajuće petlje, ne i eventualnu spoljašnju petlju koja obuhvata petlju u kojoj se neposredno nalazi break naredba. Slično pravilo važi i za continue naredbu: ako se continue naredba nalazi u ugnježđenoj petlji, ona prekida aktuelnu iteraciju samo te petlje i nema nikakvog uticaja na eventualne obuhvatajuće petlje.

Ovo dejstvo naredbi break i continue u unutrašnjim petljama je prepreka da se prekine neki glavni postupak koji se sastoji od više ugnježđenih petlji, a logičko mesto prekida je duboko ugnježđena petlja. Takvi slučajevi se često javljaju pri otkrivanju

logičkih grešaka u programu kada više nema smisla nastaviti dalju obradu, nego treba potpuno prekinuti započeti postupak.

Da bi se izvršilo potpuno prekidanje spoljašnje petlje ili samo njene aktuelne iteracije, u Javi se mogu koristiti označene petlje. Oznaka petlje se navodi ispred bilo koje vrste petlje u formi imena sa dvotačkom. Na primer, početak označene while petlje sa oznakom spoljPetlja može biti

spoljPetlja:while (...)...

Unutar tela ove petlje u nekoj unutrašnjoj petlji može se zatim koristiti naredba:

break spoljPetlja;

ili:

continue spoljPetlja;

radi potpunog prekida ili prekida samo aktuelne iteracije petlje označene sa spoljPetlja.

POVEZANE PETLJE - PETLJA U PETLJI (VIDEO)

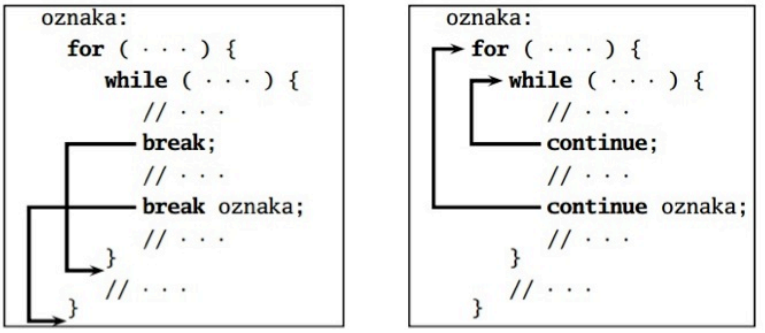
Video objašnjava primenu povezanih petlji

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMENA NAREDBI BREAK I CONTINUE U UGNJEŽDENIM PETLJAMA

Naredbe break i continue se mogu koristiti i u ugnježdenim petljama radi prebacivanja kontrole izvršenja na kraj (sa break), odn. na početak petlje (sa continue).

Efekat naredbi break i continue u ugnježđenim označenim i neoznačenim petljama ilustrovan je na slici:



Slika 4.1 Primena naredbi break i continue u unutrašnjim petljama

Pored toga što se naredbe break i continue mogu koristiti u svakoj vrsti petlje (while, do-while i for), naredba break se može koristiti i za prekid izvršavanja naredbe switch.

ZAKLJUČAK:

- Naredba ***break*** prekida dalje izvršenje petlje i šalje izvršenje prvu liniju izvan koda petlje;
- Naredba ***continue*** prekida dalje izvršenje petlje i šalje izvršenje na početak petlje, tj. na uslov petlje.

PRIMER 11 - NAREDBA BREAK U UGNEŽDENOJ PETLJI

Naredba break prekida dalje izvršenje petlje i šalje izvršenje prvu liniju izvan koda petlje.

Primer:

Prost broj je onaj koji nije deljiv bez ostatka ni sa jednim drugim brojem (osim sa 1). Treba utvrditi sve proste brojeve od broja 0 do broja 50.
Linijama 8 i 11 je prikazano ugnježđavanje petlji for. Linija 16 sadrži naredbu ***break .*** Ukoliko se izvrši ova naredba, kontrola odmah izlazi iz unutrašnje petlje i nastavlja sa sledećom naredbom iza te petlje. U ovom slučaju je to nova iteracija spoljašnje petlje.

```
public class ProstiBrojevi{
  public static void main (String{} args){
    int nVrednost = 50;     //maksimalna vrednost
                            //koja se proverava
    boolean prostBroj = true;   // tacno ako je
                            //prost broj
  // provera svih vrednosti od 2 do nVrednost:
    for(int i = 2; i <= nVrednost; i++){
     prostBroj = true;  // pretpostavka da je
                    //trenutna vrednost prost broj
    for(int j = 2; j < i; j++){
        if(i % j == 0){ //tacno ako deli bez ostatka
        prostBroj = false; // postoji deljenje bez
                  //ostatka, sto znaci da
                  //nije prost broj
        break;
        }
    }
    // ispisivanje prostih brojeva:
    if(prostBroj) {         // da li je to prost broj
        System.out.println(i); // u pitanju je prost
                        // broj pa se ispisuje
    }
  }
}
```

PRIMER 12 - PRIMENA NAREDBE CONTINUE

Naredba continue prekida dalje izvršenje petlje i šalje izvršenje na početak petlje, tj. na uslov petlje.

Pogledajmo sledeći primer:

```
for(int i = 1; i <= limit; i++){
 if(i % 3 == 0)
  continue; // preskače se ostatak iteracije
 sum += i;
}
```

Iskaz break se može upotrebiti za trenutni prekid petlje. Program sa iskazom break nastavlja kodom koji sledi iza petlje. U sledećim primerima biće demonstrirana primena naredbe break u petljama.

PRIMER 13

Traženje najvećeg zajedničkog delioca.

Napisati program koji od korisnika zahteva da unese dva broja. Na osnovu uneta dva broja naći najveći zajednički delioc. Program treba da bude deo NetBeans projekta pod nazivom KI103-PR13 i deo paketa primer13. Naziv klase treba da bude Primer13.

Objašnjenje:

Delioc je broj koji može da podeli oba broja bez ostatka.

Ono što treba prvo da proverimo jeste koji je od unetih brojeva manji. Manji broj čuvamo u pomoćnoj promenljivoj *nzd* koji označava početni delioc. Nakon toga, svaka iteracija kroz petlju umanjuje promenljivu *nzd* za 1 i ima zadatak da proveri da li taj umanjeni broj može bez ostatka da podeli prvi i drugi broj. Ukoliko može da podeli bez ostatka oba broja, taj broj postaje nzd, čime je program uspešno okončan i najveći zajednički delioc jeste broj koji je sačuvan u promenljivoj nzd. Ukoliko nijedan od brojeva manjih od početnog delioca ne može brojeve da podeli bez ostatka, petlja se ponavlja sve dok brojač nzd ne dođe do 1, čime definišemo da je najveći zajednički delioc broj 1.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package primer13;

import java.util.Scanner;
import javax.swing.JOptionPane;

/**
 *
 * @author razvoj
 */
public class Primer13 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer13();
    }

    public Primer13() {
        Scanner input = new Scanner(System.in);
        System.out.print("Unesite prvi broj: ");
        int prviBroj = input.nextInt();
        System.out.print("Unesite drugi broj: ");
        int drugiBroj = input.nextInt();
```

```
        //isto sto je i if(broj1<broj2) { d=broj1; } else { d= broj2 };
        int nzd = (prviBroj < drugiBroj) ? prviBroj : drugiBroj;
        for (; nzd >= 1; nzd--) {
            if ((prviBroj % nzd == 0) &amp;amp;&amp;amp; (drugiBroj % nzd == 0)) {
                break;
            }
        }
        System.out.printf("Najveći zajednicki delilac broja %d i broja %d je %d.
\n", +prviBroj, drugiBroj, nzd);

    }

}
```

PRIMER 14

Provera da li je broj prost ili ne

Napisati program koji od korisnika traži da upiše broj, a potom mu ispisuje da li je broj prost ili ne.Program treba da bude deo NetBeans projekta pod nazivom KI103-PR14 i *deo paketa primer14. Naziv klase treba da bude Primer14*.

Objašnjenje:

Broj je prost ako je deljiv samo sa sobom i sa jedinicom. Kod ovog zadatka prvo definišemo promenljivu *prost* koja je true. Ova promenljiva nam označava samo pretpostavku da je broj prost. U petlji, počev od broja koji je unet -1, pa sve do 1 proveravamo da li postoji broj koji ga deli bez ostatka. Ukoliko postoji, promenljiva prost postaje false i odmah zaključujemo da uneti broj nije prost. Ukoliko ne postoji broj koji ga deli bez ostatka, odnosno ukoliko petlja dođe do 1 i promenljiva *prost* bude true, možemo zaključiti da je uneti broj prost.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package primer14;

import java.util.Scanner;
import javax.swing.JOptionPane;

/**
 *
 * @author razvoj
 */
public class Primer14 {

    /**
     * @param args the command line arguments
```

```
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer14();
    }

    public Primer14() {
        Scanner ulaz = new Scanner(System.in);
        //Broj je prost ako je deljiv samo sa sobom i sa jedinicom
        System.out.println("Unesite broj za proveru da li je broj prost: ");
        int broj = ulaz.nextInt();
        /**
         * Pretpostavka da je broj prost
         */
        boolean prost = true;
        /**
         * Ukoliko je broj deljiv sa bilo kojim deliocom koji je manji od samog
         * broja, a veci od 1 on nije prost broj
         */

        int delioc = broj - 1;
        for (; delioc > 1; delioc--) {
            if (broj % delioc == 0) {
                prost = false;
                break;
            }
        }
        if (prost) {
            System.out.println("Broj " + broj + " je prost broj");
        } else {
            System.out.println("Broj " + broj + " nije prost broj");
        }
    }
}
```

ZADATAK 8

Samostalno vežbanje ugneždavanja petlji.

- Date su dve petlje sa brojačima i i j;
- Petlja sa brojačem j je ugnježdena u petlju sa brojačem i;
- Deklarisati dve int promenljive zbir1 i zbir2 i inicijalizovati ih;
- Proveriti da li je tekuće i manje od tekućeg j;
- Ako jeste dodati njihov zbir na tekuću vrednost promenljive zbir1;
- Ako nije dodati njihov proizvod na tekuću vrednost promenljive zbir2;
- Pokrenuti program i prokazati rezultate.

ISKAZI BREAK I CONTINUE U PETLJAMA (VIDEO)

Video objašnjava primenu iskaya beak i continue u petljama.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 5

Domaći zadaci

ZADACI 1 - 4

Sledeći zadaci su za samostalan rad bez rešenja

Zadatak 1

Napraviti program koji pretvara kilograme u grame od 1 do 200. Koristiti for petlju.

Zadatak 2

Napraviti program koji računa površinu kvadrata za sve stranice a od 20 do 1000 koristeći while petlju

Zadatak 3
Napraviti program koji traži od korisnika da unese broj artikala, a potom imena i cenu svih artikala. Na kraju program treba da ispiše sumu svih unetih artikala preko JOptionPane-a. Koristiti petlju za unos više artikala.

ZADACI 4 I 5

Zadaci za samostalni rad studenata

Zadatak 4

Napisati program koji prikazuje sve cele brojeve u intervalu čije granice unosi korisnik koji su deljivi brojevima 5 ili 6.

Zadatak 5
Napisati program koji će pomoću while petlje odrediti najmanji broj n čiji je stepen n2 veći od 12000.

Zakljucak

REZIME NAUČENOG

Petlje omogućavaju iterativno ponavljanje jedne ili više programskih naredbi.

1. Postoje tri tipa iterativnih kontrolnih struktura koje omogućavaju ponavljanje programskih naredbi, i to **while**, **do-while** i **for.**
2. Deo petlje koji sadrži naredbe koje se iterativno ponavljaju naziva se **telom petlje**.
3. **Iteracija** petlje je jedno izvršenje tela petlje.
4. **Beskonačna petlja** je ona čiji se naredbe beskonačno izvršavaju.
5. Pri definisanju petlje, morate uzeti u razmatranje i kontrolnu strukturu petlje, i telo petlje.
6. Kod **while** petlje prvo se proverava uslov petlje. Ako je uslov zadovoljen, tj. daje vrednost true, ide se u izvršenje tela petlje. Ako je uslov nezadovoljen, tj. false, onda se završava izvršenje petlje.
7. Petlja **do-while** je slična petlji while, s tom razlikom što se kod petlje do-while, prvo izvršava telo petlje, pa se tek onda proverava uslov petlje.
8. Petlje **while** i **do-whil**e se najčešće koriste u slučajevima kada nije unapred definisan broj prolaza kroz telo petlje.
9. Kontrolna vrednost je specijalna vrednost koja označava kraj petlje.
10. Petlja **for** se generalno koristi kada se zna koliko puta se telo petlje treba da izvrši.
11. Kontrolni deo petlje for ima tri dela. Prvi deo je akcija inicijalizacije kontrolne promenljive. Drugi deo je uslov petlje koji određuje da li će se telo petlje izvršiti. Treći deo izvršava se posle završetka svake iteracije petlje i najčešće se koristi za podešavanje nove vrednosti kontrolne promenljive. Najčešće se kontrolne promenljive inicijalizuju i menjaju u kontrolnoj strukturi petlje.
12. Kod petlji **while** i **for** provera uslova petlje se vrši pre izvršenja petlje, a kod petlje do-while uslove se proverava posle izvršenja tela petlje.
13. Naredba **break** okončava unutrašnju petlju, a naredba **continue** samo prekida trenutnu iteraciju.