6 Funkcije

Kao i do sada, počnimo ovo poglavlje sa zadatkom:

Zadatak 6.1 (Nizovi)**.**

Korisnik unosi pozitivan ceo broj n, a zatim i $2n$ cela broja. Prvi set n brojeva smestiti u prvi niz, a drugi set n brojeva smestiti u drugi niz. Ispisati poruku:

- "Suma prvog niza je veca od sume drugog niza"; ako je suma prvog niza veća od sume drugog niza.
- "Suma drugog niza je veca od sume prvog niza"; inače

Ne preostaje nam ništa drugo nego da krenemo redom:

Zadatak6.1

```
1  package nekiPaket;
2
3  import java.util.Scanner;
4
5  public class Zadatak1 {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9
10         int n = sc.nextInt();
11         int[] arr1 = new int[n];
12         int[] arr2 = new int[n];
13
14         for (int i = 0; i < arr1.length; i++) {
```

```
15              arr1[i] = sc.nextInt();
16          }
17
18          for (int i = 0; i < arr2.length; i++) {
19              arr2[i] = sc.nextInt();
20          }
21
22          int suma1 = 0;
23          for (int i = 0; i < arr1.length; i++) {
24              suma1 += arr1[i];
25          }
26
27          int suma2 = 0;
28          for (int i = 0; i < arr2.length; i++) {
29              suma2 += arr2[i];
30          }
31
32          if (suma1 >= suma2) {
33              System.out.println("Suma prvog niza je veca
    od sume drugog niza");
34          }
35          else {
36              System.out.println("Suma drugog niza je veca
    od sume prvog niza");
37          }
38      }
39  }
```

što je i validno rešenje zadatka, no pogledajmo malo bolje priloženi kod. Instrukcije kreacije prvog niza `arr1`, tj. linije koda `14-16` su identične do na ime drugog niza `arr2`, tj. linije koda `18-20`. Takodje, računanje sume niza je isto za oba niza. Za dva niza i nije preveliki problem, ali zamislite da radimo sa 100 različitih nizova. Bilo bi jako korisno kada bi smo mogli Javi da jednom 'pokažemo' kako se niz kreira, a zatim da joj damo instrukciju da taj postupak

ponovi odredjen broj puta. Upravo ova kreacija uopštenog algoritma koga kasnije pozivamo na konkretnim primerima nazivamo pisanjem **funkcija**.

Funkcije uopšteno

Detaljan matematički opis funkcija možemo videti u pogavlju 0. Uvod u sekciji Matematičke osnove - Skupovi, relacije i funkcije, te ovde nećemo iznositi formalnu definiciju funkcija, već ćemo uprošćenim jezikom izneti šta su to funkcije.

Na funkcije možemo da posmatramo kao na nešto sto prikuplja odredjene ulazne podatke, zatim ih obradi na unapred odredjen način i na kraju vrati neke podatke. Tako na primer:

- Bankomat kao ulazan podatak primi svotu novca koja treba da se podigne, zatim u pozadini prikupi novac i kao izlazni podatak izbaci tu svotu korisniku bankomata
- Funkcija 'crop picture' kao ulazan podatak primi sliku i markiran deo koji treba da odseče, zatim u pozadini odseče nemarkiran deo i kao izlazni podatak vrati izmenjenu sliku
- Funkcija $f(x) = 2 \cdot x$ kao ulazni podatak prima neki broj a kao izlazni vraća dupliranu vrednost primljenog broja
- Funkcija 'izbriši otpad' ne prima ništa kao ulazni podatak, zatim briše sve iz otpada zauvek i ne vraća ništa kao izlazni podatak
- itd.

Kao što možemo da vidimo u svim ovim primerima, kada smo opisivali rad ovih funkcija, nigde nismo naveli tačne, odnosno konkretne primere, već smo osvemu pričali uopšteno; "neka svota novca", "neka funkcija", "neki broj x" itd. Prilikom dizajniranja funkcija, mi opisujemo *uopšteni* postupak kako ta funkcija treba da

radi, a zatim je *pozivamo* na *konkretnim primerima*. Tako na primer, funkciju $f(x) = 2 \cdot x$, možemo da pozovemo na konkretnom primeru $x = 7$ i kao povratnu vrednost da dobijemo broj 14.

Šablon pisanja funkcija u Javi izgleda:

```
[<modifikatori>] <tip_povratne_vrednosti> <ime_funkcije>
(<argumenti_funkcije>) {
    //Telo funkcije
}
```

Kao i kod grananja/petlji, razlikujemo *potpis*, odnosno glavu funkcije i *telo* funkcije.

- `[<modifikatori>]` - niz modifikatora; može biti prazan
- `<tip_povratne_vrednost>` - tip koji funkcija mora da vrati. Za naše primere iznad, to bi bili: svota novca, slika, broj i ništa redom.
- `<ime_funkcije>` - jedinstveni identifikator funkcije
- `<argumenti_funkcije>` - argumenti, odnosno ulazni podaci funkcije. Može ih biti nula, jedan ili više. Različiti argumenti funkcije se navode po šablonu: `<tip_argument>` `<identifikator_argument>` odvojeni zarezima. Za naše primere iznad funkcije bi redom kao argumente uzimale: broj, sliku, broj i ništa.

Za sada sve naše funkcije će počinjati ključnim rečima `public`, `static` i ostale modifikatore neće imati. Ulogu ovih modifikatora ćemo detaljno objasniti u drugom delu kursa.

Java je programski jezik u kome funkcije **nisu** 'gradjani I reda', odnosno nije moguće dodeliti čitave funkcije promenjivima[1], ali od Java verzije 8, moguće je emulirati *zatvorenja*.[2] S toga, na

funkcije posmatramo kao na potpuno drugačije konstrukcije od promenjivih.
Najbitnije je napomenuti da **nemožemo kreirati funkciju unutar tela druge funkcije!**

Pošto funkcije kreiraju svoj blok koda i nemogu biti smeštene unutar drugih funkcija, to znači da mora biti deklarisana direktno unutar klase, te joj na raspolaganju stoje *svi* identifikatori!
Jednom kreirana, funkcija može biti *pozvana* iz bilo koje druge funkcije[3] tako što joj se proslede konkretne vrednosti, odnosno promenjive odgovarajućeg tipa kao argumenti onim redom koji je naznačen u potpisu funkcije!
Bitno je naglasiti da je redosled slanja argumenata bitan. Npr. za funkciju $f(x, y) = x/y$, funkciju koja kao rezultat daje količnik prvog i drugog argumenta, redosled argumenata je važan. Tako, $f(4, 2) = 2 \neq \frac{1}{2} = f(2, 4)$!

Mi smo do sada već pisali funkcije, i to svaki put kada smo kreirali naše programe. Naime, funkcija `main` je jedna specijalna funkcija u Javi, koja govori programu da je *pokretljiv* i označena je kao *glavna* funkcija, ona koja se prva pokreće pri pokretanju programa. Njen argument je niz niski pod nazivom `args`, što je skraćeno od engleske reči 'arguments', koje mi eksplicitno nismo navodili šta su ti argumenti i nismo ih koristili unutar naše `main` funkcije (našeg programa) prilikom pokretanja programa. Kako je `main` jedna funkcija, sve funkcije koje želimo da kreiramo idu izan nje ali unutar klase. Redosled pisanja funkcija nije bitan, jer se prvo pokreće `main` funkcija, koja dalje može da poziva ostale funkcije onim redom koji joj zadamo, no po nepisanom pravilu, sve ostale funkcije se pišu izmedju početka klase i `main` funkcije, tj. `main` funkcija se piše kao poslednja funkcija programa.

Pri kraju ovog poglavlja, razumećemo detaljanije kako funkcioniše ova specijalna funkcija.

Kao i do sada sa promenjivima, funkcije treba nazivati po camelCase-u, tako da što bolje opisuju posao koji obavljaju.

Funkcije se *pozivaju*, odnosno iskorišćavaju tako što se u kodu navede ime funkcije zajedno sa konkretnim argumentima nad kojima ona treba da radi. Tada program 'uskače' u datu funkciju i izvršava naredbe napisane u njenom telu na standardan način, liniju-po-liniju odozgo nadole. U momentu kada naidje na kraj funkcije, Java povratne vrednosti prikuplja i 'vraća' ih na mesto odakle je funkcija bila pozvana i završava izvodjenje te naredbe. Nakon toga, program nastavlja sa radom na standardni način.

Izlazak iz funkcije, odnosno naglašavanje povratne vrednosti se piše ključnom rečju **return** i ta naredba mora biti **poslednja** naredba funkcije, odnosno date grane funkcije. Nakon njenog izvršavanja, program se vraća na lokaciju odakle je funkcija bila pozvana i nastavlja svoj tok izvršavanja na standardan način.

Kada se priča o funkcijama kao sastavni deo neke klase, tj. pri objektno-orijentisanom programiranju, umesto `funkcija` koristi se izraz `metoda`.

U Javi razlikujemo dve vrste funkcija:

- Funkcije sa povratnom vrednošću
- Funkcije bez povratne vrednosti

Funkcije sa povratnom vrednošću

Definicija 6.1 (Funkcije sa povratnom vrednošću)**.**

Funkcije sa povratnom vrednošću su funkcije koje vraćaju onaj tip podatka koji se poklapa sa tipom funkcije, odnosno sa onim tipom koji mora da vrati. Ukoliko funkcija sadrži grananje, mora da zagarantuje da svaka grana funkcije vraća zadati tip podatka.

Provežbajmo pisanje funkcija sa povratnom vrednošću u Javi kroz par praktičnih primera:

Zadatak 6.2 (Funkcije sa povratnom vrednošću)**.**

Napisati funkciju koja:

1. Prima dva cela broja i vraća njihov zbir
2. Prima dva realna broja i vraca njihov količnik
3. Prima ceo broj i vraća poruku "Pozitivan", "Neutralan" ili "Negativan" u zavisnosti od prosledjene vrednosti
4. Prima dva cela broja i vraća odgovor na pitanje da li je prvi broj veći od drugog
5. Prima tri cela broja i vraća najveći od njih
6. Ne prima ništa i vraća poruku "Zdravo svete!"

Zadatak6.2

```
1  package nekiPaket;
2
3  public class Zadatak21 {
4      //Ovde navodimo naše funkcije
5
6      public static int zbirDvaBroja(int x, int y) {
7          int zbir = x + y;
8          return zbir;
```

```
9       }
10      /* U prethodnoj funkciji kao argument primamo dva
11      cela broja naznačenim kao x i y.
12      U telu funkcije u novu promenjivu 'zbir' smeštamo
    rezultat
13      zbira ova dva prosledjena broja.
14      Na kraju, kao povratnu vrednost, vracamo celobrojni
15      literal smesten u promenjivoj 'zbir'*/
16
17      public static void main(String[] args) {
18          int x = 5;
19          int y = 7;
20          int zbir = zbirDvaBroja(x, y);
21          System.out.println("Zbir je " + zbir);
22      }
23  }
```

Prodjimo detaljno kako se izvršava gornji kod:

1. Rekli smo da program prvo ulazi u `main` funkciju i izvršava njen kod liniju po liniju.
2. U linijama 18 i 19 deklarišemo dve promenjive `x` i `y`.
3. U liniji 20 pozivamo napisanu funkciju `zbirDvaBroja` i kao argumente joj prosledjujemo `literale` smeštene u promenjivama `x` i `y`, tj pozivamo `zbirDvaBroja(5, 7)`
4. Java ulazi u zadatu funkciju i kreće da je izvršava liniju-po-liniju odozgo na dole, zamenjujući svako pojavljivanje promenjive `x` sa literalom `5` i svako pojavljivanje promenjive `y` sa literalom `7`.
5. Java računa zbir ova dva literala i pamti ga u novu promenjivu
6. Na kraju, Java vraća `literal` smešten u promenjivoj `zbir` na mesto odakle je funkcija pozvana, tj. liniju 20. U tom momentu linija 20 dobija oblik: `int zbir = 12;` i ta komanda se izvršava.

7. Java sada nastavlja izvršavanje funkcije `main` prelazeći u liniju 21 i ispisuje poruku `"Zbir je 12"`

Linije koda 6-9 definišu funkciju `zbirDvaBroja` i opisuju njeno ponašanje *uopšteno*, sa *opštim* vrednostima, dok linija koda 20 poiva funkciju `zbirDvaBroja`, tj. realizuje je, sa *konkretnim* vrednostima.

Vrlo je važno primetiti da funkcije `zbirDvaBroja` i `main` definišu `paralelene` blokove koda, te sve promenjive deklarisane u jednoj funkciji **nemaju nikakve veze sa promenjivima deiklarisanim u drugoj!** Što znači da promenjiva `x` definisana u telu funkcije `main` nema nikakve veze sa promenjivom `x` definisanom kao argumentom funkcije `zbirDvaBroja`, drugim rečima, identično rešenje bi moglo da izgleda i ovako:

Zadatak6.2-Ponovo

```
1  package nekiPaket;
2
3  public class Zadatak21 {
4
5      public static int zbirDvaBroja(int x, int y) {
6          int zbir = x + y;
7          return zbir;
8      }
9
10     public static void main(String[] args) {
11         int a = 5;
12         int b = 7;
13         int c = zbirDvaBroja(a, b);
14         System.out.println("Zbir je " + c);
15     }
16 }
```

Java će izvršiti ovaj kod na isti način. Čim naidje na liniju 13, poziva funkciju `zbirDvaBroja` i sada svako pojavljivanje promenjive `x` zamenjuje sa prosledjenim `literalom` na toj poziciji prilikom pozivanja funkcije, što je `literal` promenjive `a` koji iznosi `5`. Analogno i za proemnjivu `y`. Rešenje će opet biti 12 i opet će se ispisati identična poruka!

Primetimo da u ovom konkretnom primeru koristimo reč `literal`, dok u odeljku kada smo o funckijama pričali samo uopšteno, izbegavali smo da koristimo ovu reč. To je iz razloga što argumenti u Javi mogu da se pošalju po dva principa:

1. Slanje argumenata po vrednosti
2. Slanje argumenata po referenci

Slanje argumenata po vrednosti podrazumeva slanje `literala` zadate promenjive i ovaj način slanja argumenata se uvek koristi kada su argumenti prostog tipa.
Slanje argumenata po referenci podrazumeva slanje pokazivača zadate promenjive i ovaj način slanja argumenata se uvek koristi kada su argumenti složenog, odnosno referencnog tipa. O ovom principu ćemo više pričati u odeljku `Funkcije nad nizovima`.

Kao što možemo videti, umesto promenjivih možemo slati literale i isto to važi za vraćanje vrednosti iz funkcija; sve dok je taj literal istog tipa kao tip funkcije, te rešenje gornjeg zadatka može i da se skrati na:

Zadatak6.2-Krace

```
1  package nekiPaket;
2
3  public class Zadatak21 {
4
```

```
5       public static int zbirDvaBroja(int x, int y) {
6           return x + y;
7       }
8
9       public static void main(String[] args) {
10          int c = zbirDvaBroja(5, 7);
11          System.out.println("Zbir je " + c);
12          /* Odnosno jos krace sa:
13          System.out.println("Zbir je " + zbirDvaBroja(x,
    y));
14          */
15      }
16  }
```

Primetimo da tekst zadatka kaže samo da kreiramo funkciju `zbirDvaBroja`, te pisanje `main` funkcije uopšte nije neophodno za rešenje zadatka. Mi ćemo nastaviti da pišemo `main` funkciju jedino iz razloga provere valjanosti našeg koda. Na poslednjem delu kursa, nećete uopšte pisati `main` funkciju već ćete pisati samo uopštene funkcije koje će obavljati neophodan posao.

Možemo primetiti da u rešenjima ovog prvog zadatka, unutar našeg fajla, imamo 2 funkcije: `zbirDvaBroja` i `main` funkciju. To nam govori da u jednom Java fajlu možemo imati i proizvoljan broj funkcija, pa ćemo rešenje ostalih zadataka navesti u jednom fajlu:

Zadatak6.2-Ceo

```
1
2   public class Zadatak2Ceo {
3
4       //1. Prima dva cela broja i vraća njihov zbir
5       public static int zbirDvaBroja(x, y) {
6           return x + y;
7       }
```

```
8
9       //2. Prima dva realna broja i vraca njihov količnik
10      public static double kolicnikDvaBroja(x, y) {
11          return x / y;
12      }
13
14      //3. Pozitivan, neutralan, negativan
15      public static String pozitivanNeutralanNegativan(int
    x) {
16          if (x > 0) {
17              return "Pozitivan";
18          }
19          else if (x == 0) {
20              return "Neutralan";
21          }
22          else {
23              return "Nagativan";
24          }
25      }
26
27      //4. Prima dva cela broja i vraća odgovor na
28      //pitanje da li je prvi broj veći od drugog
29      public static boolean daLiJeVeci(int prvi, int drugi)
    {
30          return prvi >= drugi;
31      }
32
33      //5. Prima tri cela broja i vraća najveći od njih
34      public static int najveciOdtri(int x, int y, int z) {
35          if (x >= y && x >= z) {
36              return x;
37          }
38          else if (y >= x && y >= z) {
39              return y;
40          }
41          return z;
```

```
42      }
43
44
45      //6. Ne prima ništa i
46      //vraća poruku "Zdravo svete!"
47      public static String zdravoSvete() {
48          return "Zdravo svete!";
49      }
50  }
```

2. Drugi zadatak je potpuno analogan prvom.
3. U trećem zadatku imamo 3 grane i za sve 3 moramo da zagarantujemo da imamo `return` naredbu, pošto naša funkcija mora da vrati `literal` tipa `String`.
4. Četvrti zadatak izvršava logičko poredjenje čija vrednost će biti neki logički literal. Tada se taj logički literal vraća pozivalacu te funkcije.
5. Kao i kod trećeg zadatka, imamo 3 grane koda. Ukoliko prilikom poziva funkcije, ni prvi ni drugi uslov nisu ispunjeni, možemo vratiti treći broj, pošto je on zasigurno veći od prva dva. Kada imamo grananje u funkciji sa povratnom vrednošću, moramo da zagarantujemo da svaka grana koda vraća neku vrednostu i da zagarantujemo da glavna grana funkcije vraća neku vrednost. Ovde je to izloženo linijom 41. Umesto toga, mogli smo da iskoristimo i `else` granu `if-else-if` lanca kao i u trećem zadatku ili pak, mogli smo treći zadatak da uradimo po uzoru na peti - oba rešenja su podjednako validna.
6. Funkcija iz šestog zadatka ne treba da primi ništa, što u Javi zapisujemo sa praznim zagradama - tj. kreiramo jednu *konstantnu* funkciju.

Već smo rekli da jedna funkcija može da poziva drugu i to možemo da primetimo u zadatim primerima kada `main` funkcija poziva

funkciju `zbirDvaBroja`. Naravno, nije `main` funkcija jedina koja može da poziva ostale funkcije. Dokažimo ovo tako što ćemo ponovo rešiti zadatak 6.2 pod 5, gde ćemo ovaj put iskoristiti sledeću taktiku:

- Napisaćemo pomoćnu funkciju `veciOdDvaBroja` (suštinski našu `Math.max()` funkciju) koja prima dva broja i vraća veći od njih
- Funkcija `najveciOdTri` će prvo pozvati funkciju `veciOdDva`, proslediti joj prva dva broja kao argument i zapamtiti rezultat.
- Tada će rezultat biti veći od prva dva broja pa će naše rešenje biti rešenje funkcije `veciOdDva` kada joj prosledimo taj rezultat i treći broj.

NajveciOdTri-Ponovo

```
1  package nekiPaket;
2
3  public class NajveciOdTri {
4
5      public static int veciOdDva(int x, int y) {
6          if (x >= y)
7              return x;
8          return y;
9      }
10
11     public static int najveciOdTri(int x, int y, int z) {
12         int veciOdPrvaDva = veciOdDva(x, y);
13         return veciOdDva(veciOdPrvaDva, z);
14         /* Ili u jednom redu:
15         return veciOdDva(veciOdDva(x, y), z);
16         */
17     }
18 }
```

U 'ubrzanom' primeru u komentaru, izraz u liniji 15 se izvršava na standardni način: prvo se sve unutar zagrada razrešava, što je funkcija `veciOdDva(x, y)` a zatim Java ponovo poziva funkciju `veciOdDva` sada sa dobijenom vrednosti iz unutrašnjeg poziva `veciOdDva` funkcije i brojem `z`.

Prateći formule (F1) i (F2) iz oblasti 0.Uvod, poglavlje Matematičke osnove - Skupovi, relacije i funkcije, znamo da funkcija može imati samo jednu *sliku*, što se kod nas izražava činjenicom da funkcije mogu imati samo jednu povratnu vrednost![4]

Kao i u primeru sa brisanjem podataka iz kante za otopatke, neke funkcije jednostavno treba da odrade odredjen posao i ne treba nikakvu povratnu vrednost odnosno informaciju da vrate. Takve funkcije nazivamo **procedurama**.

Funkcije bez povratne vrednosti - procedure

Definicija 6.2 (Funkcije bez povratne vrednosti)**.**

Funkcije bez povratne vrednosti, još poznate kao procedure, su funkcije tipa `void` i one ne vraćaju nikakvu povratnu vrednost.

Na funkcije bez povratne vrednosti možemo i da posmatramo kao na specijalan oblik funkcija sa povratnom vrednošću - one koje kao povratnu vrednost vraćaju `ništa`. Svakako, obe interpretacije su valjane. Da funkcija nema povratnu vrednost naznačavamo tako što umesto tipa povratne vrednosti pišemo ključnu reč `void` (eng. ništa) i kao takve, one ne zahtevaju `return` naredbu, već svoj rad prekidaju čim dodju do kraja svog bloka koda, odnosno tela.

Primeri ovakvih funkcija bi bili:

- Promeni pozadinu ekerana
- Usisaj kuću
- Ispiši poruku
- itd.

Zadatak 6.3 (Funkcije bez povratne vrednosti)**.**

Napisati funkciju koja:

1. Prima ceo broj i ispisuju poruku "Paran", ukoliko je zadat broj paran, odnosno poruku "Neparan" inače.
2. Ispisuje poruku "Zdravo svete!" 100 puta.
3. Od korisnika zahteva da unese pozitivan ceo broj koji označava godine korisnika i ispisuje poruku "Punoletni ste", ukoliko je korisnik punoletan, a poruku "Niste punoletni" inače.
4. Iscrtava iks-oks praznu tablu.

Zadatak6.3

```
1  package nekiPaket;
2
3  import java.util.Scanner;
4
5  public class Zadatak3 {
6
7      //1. Paran-Neparan
8      public static void paranNeparan(int x) {
9          if (x % 2 == 0)
10             System.out.println("Paran");
11         else
12             System.out.println("Neparan");
```

```java
    }

    //2. Zdravo svete 100 puta
    public static void zdravoSveteStoPuta() {
    /* Ova funkcija ne treba ni da primi išta
    *  ni da vrati išta
    */
        for (int i = 0; i < 100; i++)
            System.out.println("Zdravo svete!");
    }

    //3. Punoletstvo
    public static void punoletstvo() {
        /* Ako smo mogli u main funkciji
        *  da deklarisemo i koristimo
        *  skener, onda to mozemo i u
        *  ostalim funkcijama
        */
        Scanner sc = new Scanner(System.in);
        int brojGodina = sc.nextInt();
        if (brojGodina >= 18)
            System.out.println("Punoletni ste");
        else
            System.out.println("Niste punoletni");
    }

    //4. Iks-oks tabla
    public static void iscrtajTablu() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (j != 2)
                    System.out.print(" |");
                else
                    System.out.print(" ");
            }
            System.out.println();
```

```
            if (i != 2)
                System.out.println("------");
        }
        return;
        /* Iako nema potrebe
        *  mozemo eksplicitno naglasiti
        *  return naredbu, bez ikakve
        *  povratne vrednosti.
        */
    }

    //Pisemo main funkciju samo da istestiramo napisane stvari
    public static void main(String[] args) {
        /* Sada nema potrebe pamtiti povratne
        *  vrednosti funkcija, jer ih ni nema
        *  kao ni navoditi ih u System.out.print
        *  naredbama jer ce one same da ispisuju podatke
        */
        paranNeparan(5); //Ispisace se "Neparan"
        zdravoSveteStoPuta(); //Ispisace se odgovarajuca poruka 100 puta
        punoletstvo();
        /* Prethodna funkcija ce kreirati novi skener
        *  zatim na isti nacin kao i do sada
        *  od korisnika zahtevati da unese ceo broj
        *  i onda ispisati odgovarajucu poruku
        */
        iscrtajTablu(); //Iscrtace tablu
    }
}
```

Zadatak 6.3 pod 3 smo mogli da uradimo i na sledeci način:

Punoletstvo-Skener

```
package nekiPaket;
```

```
2
3  import java.util.Scanner;
4
5  public class Punoletstvo{
6
7      public static void punoletstvo(Scanner sc) {
8          int brojGodina = sc.nextInt();
9          if (brojGodina >= 18)
10             System.out.println("Punoletni ste");
11         else
12             System.out.println("Niste punoletni");
13     }
14
15     public static void main(String[] args) {
16         Scanner sc = new Scanner(System.in);
17         punoletstvo(sc);
18     }
19 }
```

Bitna razlika izmedju ova dva rešenja leži u tome što u prvom rešenju mi kreiramo potpuno novi skener, koji će biti drugačiji od onog iz `main` funkcije i tokom izvršavanja funkcije `punoletstvo` taj novi skener je prazan i čeka korisnikov input. S druge strane, u drugom rešenju mi prosledjujemo naš skener iz `main` funkcije kao `arugment`, a pošto je skener jedan objektni, odnosno referencni tip, onda se on prosledjuje kao `referenca` i sve što je bilo sadržano u skeneru (svi neiskorišćeni tokeni) će takodje biti prisutni i tokom izvršavanja funkcije `punoletstvo` pošto se radi o jednom istom skeneru! Više o ovom načinu prosledjivanja parametara ćemo pričatu u narednom odeljku.

Funkcije nad nizovima

Funkcije mogu da primaju i nizove kao svoje argumente, pa čak i da vraćaju nizove kao povratne vrednosti. Kako smo rekli da su identifikatori nizova pokazivači, oni se onda prosledjuju po referenci. Ilustrujmo ovo ponašanje jednim primerom:

ReferencaVSvrednost

```
1  package nekiPaket;
2
3  import java.util.Arrays;
4
5  public class ReferencaVSVrednost {
6
7      public static void promeniBroj(int x) {
8          x = 123;
9          System.out.println(x);
10     }
11
12     public static void promeniNiz(int[] arr) {
13         arr[0] = 123;
14         System.out.println(Arrays.toString(arr));
15     }
16
17     public static void main(String[] args) {
18         int x = 5;
19         promeniBroj(x);
20         System.out.println(x);
21
22         System.out.println("===");
23
24         int[] arr = {1, 2, 3, 4, 5};
25         promeniNiz(arr);
26         System.out.println(Arrays.toString(arr));
27     }
28 }
```

Prethodni kod će se izvršavati na sledeći način:

1. Kreiraće se primitivna promenjiva tipa `int` sa vrednošću 5.
2. Funkcija `promeniBroj` kao argument uzima vrednost prosledjene promenjive, što je `literal` 5. Funkcija odmah menja tu vrednost u 123 i ispisuje se baš taj broj `"123"`.
3. Nakon povratka iz funkcije `promeniBroj` ispisuje se vrednost promenjive `x`, što je sada `literal` 5, dakle poruka `"5"`. Ovo se dešava pošto funkcije primaju `vrednosti` primitivnih promenjivih a ne njihovu memorijsku lokaciju!
4. Ispisuje se deliminator `"==="`
5. Kreira se niz od pet brojeva i poziva se funkcija `promeniNiz` koja prima jedan niz `int[]` kao arugment. Kako su identifikatori niza reference, to se prosledjuje `referenca` na niz `arr` iz `main` funkcije, funkciji `promeniNiz`. Ona menja prvi element niza u 123 i ispisuje poruku `"[123, 2, 3, 4, 5]"`.
6. Nakon povratka u `main` funkciju, ispisuje se porukua `"[123, 2, 3, 4, 5]"`, zato što je funkcija `promeniNiz` dobila referencu niza `arr` i u liniji 13 pristupila prvom elementu baš tog konkretnog niza!

Na sličan način se ponašao i skener u poslednjem primeru iz prethodnog odeljka.
Više o tome kako funkcije tretiraju svoje argumente se može naći u odeljku stek okviri.

Svaki put kada se iskoristi ključna reč `new`, kreira se **nova** referenca na zadati objekat/niz itd. Dakle, ako želimo da naše funkcije kreiraju nove nizove, ili da duboko kopiraju nizove čija je referenca prosledjena kao argument, to moramo raditi sa ključnom rečju `new`.

Pogledajmo par zadataka koji se tiču pisanja funkcija za rad sa nizovima.

Zadatak 6.4 (Funkcije nad nizovima)**.**

Napisati funkcije za lakši rad sa nizovima (celih brojeva). Preko funkcija, definisati samo jednom svaku od bitnijih procedura nizova poput:

1. Kreiranje niza zadate dužine i zadatih vrednosti
2. Ispisivanje elemenata niza (bez korišćenja `Arrays.toString()`)
3. Računanje sume elemenata niza
4. Pronalaženje najvećeg i najmanjeg elementa niza
5. itd.

FunkcijeNadNizovima

```java
package nekiPaket;

import java.util.Scanner;

public class FunkcijeNadNizovima {

    //Kreacija niza duzine n
    //Zelimo da vratimo kreiran niz!
    public static int[] kreirajNiz(int n) {
        int[] arr = new int[n];
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();

        return arr;
    }

```

```java
    //Ispisivanje niza
    public static void ispisiNiz(int[] arr) {
        System.out.print("[");
        for (int i = 0; i < arr.length - 1; i++)
            System.out.print(arr[i] + ", ");
        System.out.println(arr[arr.length - 1] + "]");
    }

    //Suma niza
    public static int sumaNiza(int[] arr) {
        int suma = 0;
        for (int a : arr)
            suma += a;
        return suma;
    }

    //Najveci element niza
    public static int maxNiza(int[] arr) {
        int max = arr[0];
        for (int a : arr)
            if (a > max)
                max = a;
        return max;
    }

    //Najmanji element niza
    public static int minNiza(int[] arr) {
        int min = arr[0];
        for (int a : arr)
            if (a < min)
                min = a;
        return min;
    }

    //0 - bira najmanji,
    //1 ili ostalo - bira najveci
```

```
54      public static int minMaxNiza(int[] arr, int odabir) {
55          if (odabir == 0)
56              return minNiza(arr);
57          return maxNiza(arr);
58      }
59
60      //itd.
61      //Provera:
62      public static void main(String[] args) {
63          Scanner sc = new Scanner(System.in);
64          int duzinaNiza = sc.nextInt();
65          int[] mojNiz = kreirajNiz(duzinaNiza);
66          //npr mojNiz = [4, 5, 3, 6, 2];
67          ispisiNiz(mojNiz); //Ispisuje [4, 5, 3, 6, 2]
68          int suma = sumaNiza(mojNiz); //suma = 20
69          int max = maxNiza(mojNiz); //max = 6
70          int min = minNiza(mojNiz); //min = 2
71          /* Umesto da prvo kreiramo niz
72          *  pa da ga prosledjujemo, to mozemo
73          *  da uradimo direkno sa navodjenjem
74          *  niza; kao i za primitivne promenjive
75          */
76          int minMaxNovogNiza = minMaxNiza(new int[]{101,
    102, 103}, 0); // = 101
77      }
```

Konačno, rešimo prvi zadatak koristeći funkcije

Zadatak6.1-Funkcije

```
1   package nekiPaket;
2
3   import java.util.Scanner;
4
5   public class Zadatak1Funkcije {
6
```

```java
    public static int[] kreirajNiz(int n) {
        int[] arr = new int[n];
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();

        return arr;
    }

    public static int sumaNiza(int[] arr) {
        int suma = 0;
        for (int a : arr)
            suma += a;
        return suma;
    }

    public static void prviVeciOdDrugog(int[] a, int[] b)
{
        int suma1 = sumaNiza(a);
        int suma2 = sumaNiza(b);
        if (suma1 >= suma2)
            System.out.println("Suma prvog niza je veca
od sume drugog niza.");
        else
            System.out.println("Suma drugog niza je veca
od sume prvog niza.");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] prviNiz = kreirajNiz(n);
        int[] drugiNiz = kreirajNiz(n);
        prviVeciOdDrugog(prviNiz, drugiNiz);
        /* Ili krace
        * prviVeciodDrugog(kreirajNiz(n), kreirajNiz(n));
```

```
40          */
41      }
42
```

Proizvoljan broj argumenata funkcije

U Javi, moguće je navesti i proizvoljan broj argumenata funkcije uz par restrikcija:

- Takvi argumenti moraju biti istog tipa
- Takvi argumenti se moraju biti poslednji argument funkcije

Proizvoljan broj argumenata funkcije se navodi kao `<tip_tih_promenjivih> ... <identifikator>`. To nam omogućava da prosledimo proizvoljan broj argumenata odgovarajućeg tipa, odvojenih zarezom, prilikom poziva date funkcije. Interno, Java te argumente smešta u niz koji označava prosledjenim identifikatorom i tim argumentima pristupamo na standardan način kada radimo sa nizovima. Pogledajmo jedan praktičan primer:

ProizvoljanBrojArgumenata

```
1  package nekiPaket;
2
3  public class ProizvoljanBrojArgumenata {
4
5      public static int sumaSvih(int ... brojevi) {
6          int suma = 0;
7          for (int elem : brojevi)
8              suma += elem;
9          return suma;
10     }
11
12     public static void main(String[] args) {
```

```
13          int suma1 = sumaSvih(1, 2, 3, 4, 5); // = 15
14          int suma2 = sumaSvih(100, 200); // = 300
15      }
16  }
```

Bitna razlika izmedju:

```
1  public static int f(int[] a) {
2      //...
3  }
```

i

```
1  public staic int f(int ... a) {
2      //...
3  }
```

je u tome što u prvom primeru moramo da prosledimo neki niz brojeva, dok u drugom je dovoljno da navodimo proizvoljne promenjive, odnosno literale tipa `int`.

Kao što smo napomenuli, pored argumenta sa proizvoljnim brojem članova, mogu se naći i ostali argumenti, npr:

```
1  public static void f(String s, double d, char[] c, double
   ... a) {
2      //...
3  }
```

ali argument sa proizvoljnim brojem članova uvek mora biti **poslednji** argument jer u suprotnom Java ne bi znala koje prosledjene vrednosti pripadaju kojim argumentima (u slučaju npr. `double a, double ... b, double c, double ... d`). Kao posledicu

ovoga, nije moguće kreirati funkciju koja ima dva argumenta sa proizvoljnim brojem članova.

Stek okviri

Definicija 6.3 (Stek)**.**

Stek (eng. 'stack') je struktura podataka koja organizuje svoje podatke tako što ih 'redja jedan na drugi'. Jedine operacije koje možemo da vršimo nad stekom su dodavanje stvari na vrh steka, skidanje stvari sa vrha steka i uz to imamo informaciju o veličini našeg steka.[5]

Prilikom pozivanja odredjene funkcije, kreira se njen takozvani **stek okvir** na koji se redom postavljaju zadati primitivni argumenti te funkcije.

Definicija 6.4 (Stek okvir)**.**

Stek okvir funkcije je njena privatna stek memorija na koju se redom postavljaju zadati primitivni argumenti te funkcije. Pored toga, funkcija ima dve dodatne (skrivene) promenjive, jednu koja označava gde u memoriji počinje zadati stek, a drugu koja označava gde se nalazi vrh zadatog steka.

Svaka nova promenjiva (primitivnog tipa) koja se deklariše u telu funkcije će pripadati tom konkretnom steku koji se kreira kada se funkcija poziva. Nakon izvršavanja cele funkcije (odnosno pri `return` naredbi) ceo stek okvir se briše i jedino što se vraća pozivaocu funkcije je povratna vrednost.[5] Upravo zbog ovakvog funkcionisanja funkcija, primitivne promenjive zadate kao argumenti u potpisu funkcije nemaju nikakve veze sa primitivnim

promenjivima koje prosledimo kao argumente prilikom poziva funkcija.

Složeni tipovi podataka ne žive u steku funkcije već se nalaze u zajedničkoj memoriji koju nazivamo **hipom** (eng. 'heap'), te prilikom poziva funkcija sa složenim tipovima kao argumentima mi šaljemo `referencu` kao `pokazivač` na zadatu memorijsku lokaciju u hip memoriji. Ovo za posledicu ima to da svaka promena nad složenim promenjivima unutar funkcije ima isti uticaj na promenjivu koja je prosledjena kao argument iz funkcije-pozivaoca. Ovakvu pojavu nazivamo **sporednim efektom**.

Definicija 6.5 (Sporedni efekat)**.**

Sporedni efekat funkcije je svaki efekat koja funkcija ima na stvari van njenog tela.

Ovakvo ponašanje odstupa od ponašanja funkcija u matematici. Naime, za neku matematičku funkciju $f(x)$ i za neku konkretnu vrednost *originala*, npr vrednost a, funkcija **uvek vraća istu povratnu vrednost**! Npr:

$$\begin{aligned} f(x) &= x^2 \\ x &= 5 \\ f(5) &= 25 \end{aligned}$$

Kvadratna funkcija x^2 će za jednu zadatu vrednost uvek vraćati isti rezultat - taj broj kvadriran. Naravno ona može za različite vrednosti da vrati isti rezultat (zato što nije "1-1", npr. $f(-5) = f(5) = 25$) ali ništa ne može da naruši ovu relaciju: $f(5)$ će uvek biti 25!

U Javi to nije slučaj. Pogledajmo naredni primer:

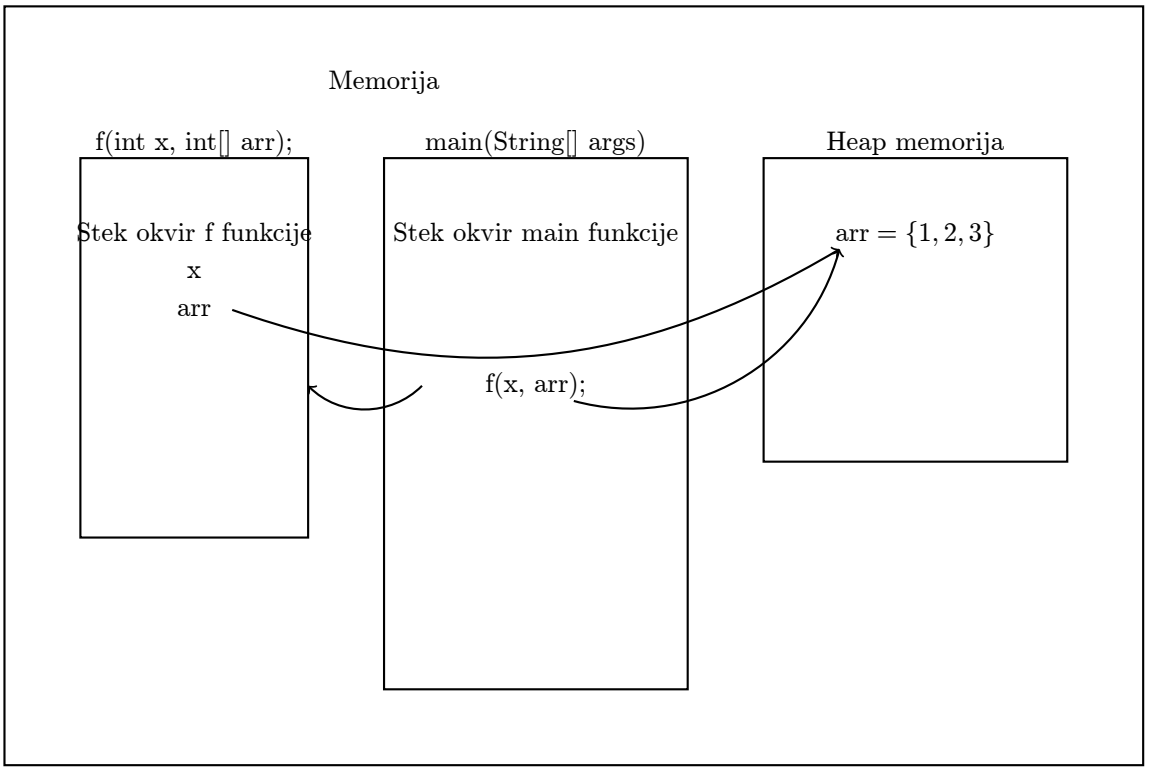
SporedniEfekat

```
1  package nekiPaket;
2
3  public class SporedniEfekat {
4
5      public static void f(int x, int[] arr) {
6          if (x == arr[0])
7              System.out.println("Isti su");
8          else
9              System.out.println("Nisu isti");
10         arr[0] = x == 0 ? 1 : 0;
11     }
12
13     public static void main(String[] args) {
14         int x = 5;
15         int[] arr = {5, 4, 3};
16         f(x, arr); //Prvi poziv
17         f(x, arr); //Drugi poziv
18     }
19 }
```

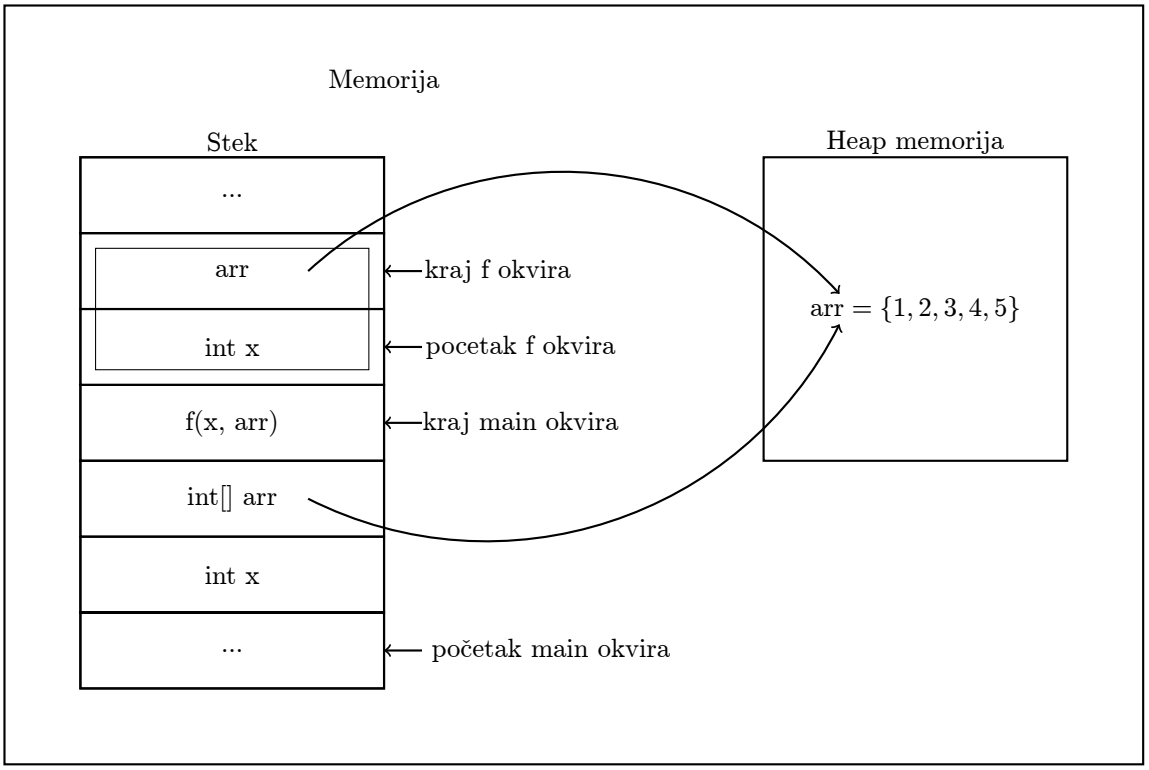
Linije 14 i 15 pozivaju istu funkciju sa potpuno istim argumentima, što bi matematički gledano, trebalo da vrati istu vrednost (u našem slučaju ispiše istu poruku), ali to nije ovde slučaj. Prvi poziv funkcije će ispisati poruku "Isti su", pošto je udatom momentu `x = 5` i `arr[0] = 5`. Nakon tog ispisa, komanda u liniji 8 menja vrednost prvog elementa niza u `0`. Zatim se poziva ista funkcija drugi put, ali ovaj put se ispisuje poruka "Nisu isti", jer su sada proseldjene vrednost `x = 5` i `arr = {0, 4, 3}`. Dakle, funkcija `f` je imala sporedni efekat na niz `arr` iz funkcije `main`. Na sporedne efekte treba posebno obratiti pažnju jer mali propust može dovesti do ozbiljnih lančanih posledica!

Prikažimo grafičke ilustracije funkcija i njihovih stek okvira:

- Interpretirana slika



- Malo realnija slika



Za par završnih reči ostalo je još da kažemo da za ovaj deo koji se tiče neobjektno-orijentisanog programiranja, pisanje, odnosno korišćenje funkcija formalno nije neophodno, ali nam one znatno

pomažu u organizaciji našeg koda u logičke celine, u abstraktizaciji rešenja naših problema i u smanjenju linija koda koje treba ispisati.

---

*Rekurzija

todo:

---

1. Programski jezici poput Python-a i JavaScript-a imaju ovu mogućnost. Oni tretiraju funkcije kao bilo koje druge promenjive. Java nema ovu funkcionalnost. Programski jezici koji su potpuno funkcionalni, poput Haskell-a, nemaju obične promenjive, jer kod njih je sve funkcija! ↩
2. Zatvorenja su još jedan koncept iz funkcionalne paradigme i u Javi se realizuju pomoću `java.util.Function.Functions` biblioteke. ↩
3. Pa čak i same sebe! Videti odeljak 'Rekurzija'. ↩
4. Ovo nije slučaj u programskim jezicima poput Python-a. Naime, Python ima ključnu reč `yield` koja vraća zadatu vrednost iz funkcije nazad pozivaocu te funkcije i nastavlja dalje sa svojim radom. ↩
5. Zapravo, ona se sada postavlja na vrh stek okvira funkcije-pozivaoca. ↩