5 Nizovi

Posmatrajmo naredni zadatak

Zadatak 5.1 (Ocene Djaka)**.**

Korisnik unosi pozitivan ceo broj n a zatim i n ocena nekog djaka. Izračunati prosek tog djaka.

Ovaj zadatak ima jednostavno rešenje:

OceneDjaka

```
1  package nekiPaket;
2
3  import java.util.Scanner;
4
5  public class OceneDjaka {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9          int n = sc.nextInt();
10         int sumaOcena = 0;
11         for (int i = 0; i < n; i++)
12             sumaOcena += sc.nextInt();
13
14         System.out.println("Prosek je " +
   ((double)sumaOcena/n));
15     }
16 }
17
18 Input:
```

```
5 5 2 5 5
Prosek: 4.4
```

Možemo videti da za ovaj konkretni primer, ova dvojika našeg djaka sprečava da ima prosek preko 4.5. Sa ovim sistemom, nikako ne možemo da dopustimo djaku da ispravi svoju ocenu i dostigne status odličnog djaka. Za tako nešto, morali bi smo ove ocene da smestimo u neke promenjive, ali pre korisničkovog unosa broja n, ne znamo koliko će nam promenjivih biti potrebno! Ovako nešto možemo da postignemo koristeći **nizove**.[1]

Nizovi kao tip podatka

Definicija 5.1 (Niz)**.**

Niz (eng. array) predstavlja 0-indeksiran skup literala istog tipa.
Pod 0-indeksiranim podrazumevamo da svaki element tog skupa ima svoj indeks, tj. da je taj skup uredjen, i da prvi element ima indeks 0.

Šablon deklarisanja sa inicijalizacijom niza od N članova u Javi je:

```
<tip_podatka>[] <identifikator> = {<element1>,
<element2>, ..., <elementN>};
```

ili, u takozvanom C stilu[2]:

```
<tip_podatka> <identifikator>[] = {<element2>,
<element2>, ..., <elementN>};
```

Naravno, prateći definicju niza, svaki navedeni element mora biti onog tipa koji je `<tip_podatka>`.
Deklarišimo i inicijalizujmo par praktičnih nizova:

```java
package nekiPaket;

import java.util.Scanner;

public class PrimerNizova {

    public static void main(String[] args) {
        int[] nizPrvihPetPozitivnihBrojeva = {1, 2, 3, 4, 5}; //Niz od 5 int brojeva
        char[] jaSamNizKaraktera = {'Z', 'd', 'r', 'a', 'v', 'o', ' ',
                                    's', 'v', 'e', 't', 'e', '!'};
        double[] nizDoubleBrojeva = {3.14}; //Niz od jednog double broja
        String[] nizNiski = {"Zdravo", "svete!", "Kako", "ste?"}; //Niz niski tj. niz čiji je svaki element niz karaktera.
        boolean[] nizBooleana = {true, false, true, true};
    }
}
```

Pored deklarizacije sa inicijalizacijom, možemo i da odložimo inicijalizaciju, i da samo deklarišemo nizove u Javi:

```
<tip_podatka>[] <identifikator> = new <tip_podatka>[<dužina_niza>];
```

Npr:

```java
package nekiPaket;

import java.util.Scanner;

```

```
5  public class PrimerNizova2 {
6
7      public static void main(String[] args) {
8          int[] nizCelihBrojeva = new int[5]; //Deklarisali
   smo niz od 5 celih brojeva
9      }
10 }
```

Ovakav zapis zapravo deklariše jedan niz od 5 elementa i postavlja te elemente na podrazumevane vrednosti. Podrazumevane vrednosti su:

- `int` -> 0
- `double` -> 0.0
- `char` -> '\u0000' tj `null` karakter
- `boolean` -> false
- `String` -> `null`
- Objekat -> `null`
  Dakle, naš gornji kod je kraći zapis za:

```
1  int[] nizCelihBrojeva = {0, 0, 0, 0, 0};
```

Takodje, samu inicijalizaciju dužine niza možemo da odložimo, te je dozvoljeno:

```
1  int[] arr;
2  //Neki ostali kod
3  arr = new int[10]; //U ovom momentu imamo niz od 10 nula
```

Suštinski, uglaste zagrade `[]` naznačavaju da radimo sa nizovima.

Važna činjenica nizova je da kada jednom odredimo dužinu niza, bilo eksplicitno sa `new int[<literal_celog_broja>]` ili implicitno

zadavanjem vrednosti izmedju vitičastih zagrada `{}`, tu dužinu ne možemo da promenimo! Dakle, nije nam dozvoljeno da dodajemo niti da oduzimamo elemente niza! Ono što nam je dozvoljeno, je da menjamo elemente niza, pošto time ne narušavamo dužinu niza.

Pristupanje elementima niza

Već smo rekli da nizovi pretstavljaju 0-indeksirane skupove i da svaki element niza ima svoj indeks u tom nizu. Da pristupimo elementu niza koristimo šablon:

```
<identifikator_niza>[<indeks_željenog_elementa>]
```

Kao rezultat ove komande, dobijamo element niza koji je smešten na zadati indeks u zadatom nizu. Tako na primer, ako želimo da pristupimo prvom elementu nizova iz prvog primera, to bi smo mogli da uradimo na sledeći način:

```
package nekiPaket;

import java.util.Scanner;

public class PristupanjeElementima {

    public static void main(String[] args) {
        //indeksi                                0  1  2  3  4
        int[] nizPrvihPetPozitivnihBrojeva = {1, 2, 3, 4, 5};
        char[] jaSamNizKaraktera = {'Z', 'd', 'r', 'a', 'v', 'o', ' ',
                                    's', 'v', 'e', 't', 'e', '!'};
        double[] nizDoubleBrojeva = {3.14};
```

```
        String[] nizNiski = {"Zdravo", "svete!", "Kako", "ste?"};
        //indeski                    0     1      2     3
        boolean[] nizBooleana = {true, false, true, true};

        int prviUIntNizu = nizPrvihPetPozitivnihBrojeva[0]; // = 1
        char prviUCharNizu = jaSamNizKaraktera[0]; // = 'Z'
        double prviUDoubleNizu = nizDoubleBrojeva[0]; // = 3.14
        String prviUStringNizu = nizNiski[0]; // = "Zdravo"
        boolean prviUBooleanNizu = nizBooleana[0]; // = true
    }
}
```

Ovim putem, u zadate promenjive u pisali smo prvi element iz odgovarajućeg niza. Koristili smo broj 0 da pristupimo prvom elemntu niza upravo zato što su nizovi 0-indeksirani u Javi. Drugim rečima, za niz dužine N, indeksi elemenata su brojevi `0 ... N-1`, gde `N-1` označava indeks *poslednjeg* elementa u nizu (recimo za niz karaktera to bi bio karakter `'!'`).
Naravno, ukoliko bi smo želeli da otštampamo prvi element iz niza niski, ne bi smo morali da ga pamtimo u promenjivu, već bi mogli da uradimo nešto poput:

```
System.out.println(nizNiski[0]);
```

Da promenimo element na zadatom indeksu nekog niza, što smo rekli da je dozvoljeno, koristimo narednu konstrukciju:

```
```

```
<identifikator_niza>[<indeks>] =
<novi_element_odgovarajućeg_tipa>;
```

Dakle, ako bi smo želeli da umesto `"svete!"` u našem nizu niski imamo reč `"narode!"`, taj element bi mogli da modifikujemo na sledeći način:

```
nizNiski[1] = "narode!"; //Pristupamo drugom elementu,
dakle on se nalazi na indeksu 1!
```

Mi smo do sada vrlo eksplicitno koristili niz. Da li možete da primetit gde?

Sa dosadašnjim znanjem, probajmo da odradimo sledeći zadatak

Zadatak 5.2 (Ispis zadatog niza)**.**

Kreirati niz od prvih pet pozitivnih celih brojeva. Ispisati taj niz.

Naivno rešenje bi bilo:

IspisZadatogNiza

```
package nekiPaket;

public class IspisZadatogNiza {

    public static void main(String[] args) {
        int arr = {1, 2, 3, 4, 5};
        System.out.println(arr);
    }
}
```

U koliko pokrenemo ovaj kod, ne dobijamo očekivani ispis, npr `[1, 2, 3, 4, 5]`, već dobijamo `[I@<heksadecimalan_broj>` ! Ovakvo

ponašanje je totalno valjano i Java stvarno ispisuje ono što smo joj zadali, a to je **pokazivač** ka našem nizu.

Pokazivači

Kako Java pripada klasi programskih jezika koji automatski alociraju i oslobadjaju memoriju, ne postoji potreba za `pokazivačima` kao posebnim tipovima podataka, ali to ne znači da se ne javljavju u složenijim konstrukcijama poput nizova i objekata. [3] Definišimo pojam pokazivača:

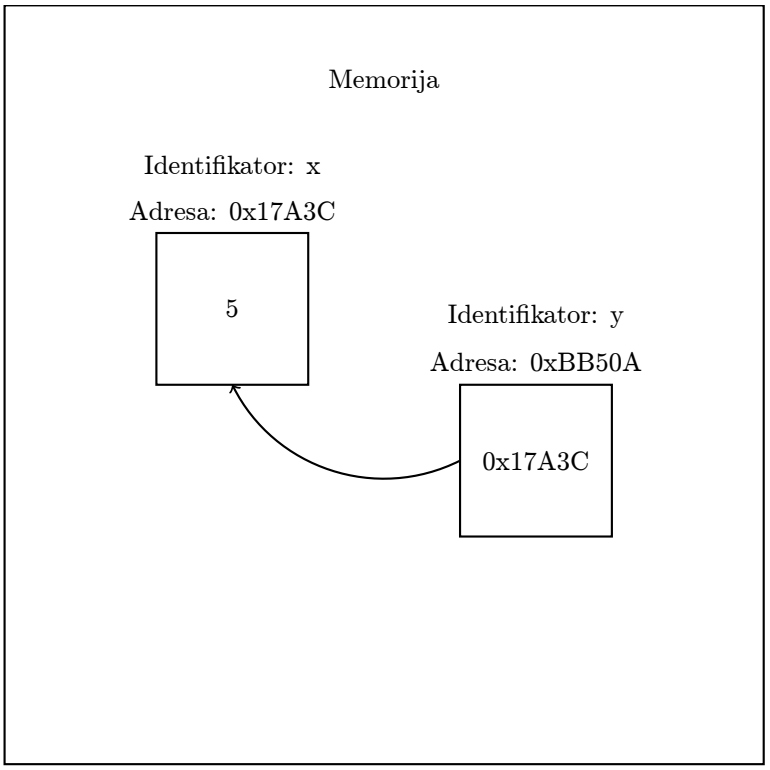
Definicija 5.2 (Pokazivač)**.**

Pokazivač odredjenog tipa predstavlja lokaciju u memoriji čija je vrednost neka druga lokacija u memoriji tog istog tipa.

Pošto pokazivači kao tipovi podataka ne postoje u Javi, radi ilustracije, koristićemo C-ovsku notaciju u ovom odeljku. Primer pokazivača tipa `int` bi bio:

```
1  int x = 5; //promanjiva tipa int
2  int* y = &x; //pokazivač na tip int
```

Simbol zvezdice, `*` naznačava da će promenjiva biti pokazivač, dok simbol ampersenda, `&`, predstavlja memorijsku lokaciju kojoj je pridružen zadati identifikator. U memoriji, sve ovo bi izgledalo:

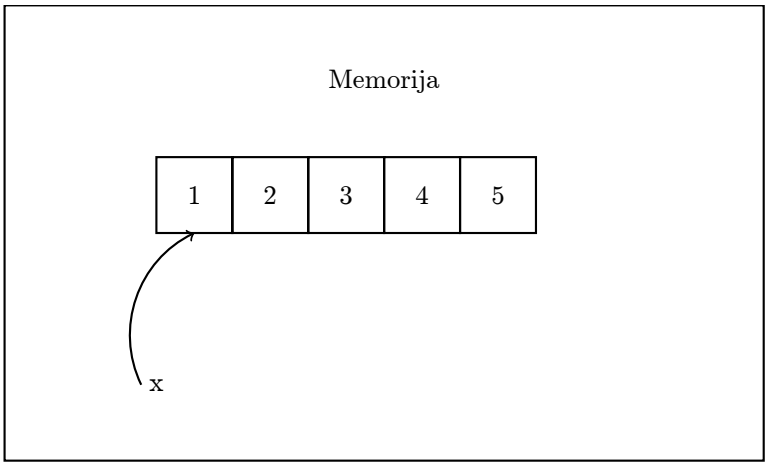


Dakle, vrednost promenjive (koja je tipa `pokazivač`) y je 0x17A3C, što je upravo memorijska lokacija koju smo mi označili sa `x` i njena vrednost je 5.

U Javi, kada jednom nizu dodelimo identifikator, taj identifikator je zapravo pokazivač na početak našeg niza. Već smo pričali o tome da u memoriji, Java promenjive alocira na neki svoj način, koji je nama nepoznat. Isto kao i u gornjem C primeru, kada bi smo kreira dve promenjive tipa `int`, jednu ispod druge, to ne znači da bi oni bili na susednim pozicijama u memoriji, već bi ih Java alocirala na bilo koja dva slobodna mesta. Kada su nizovi u pitanju, Java (a i ostali programski jezici) elemente niza postavljaju na memorijske lokacije koje su medjusobni susedi, tako da na prvu memorijsku lokaciju ide prvi element niza, na drugu drugi itd. Grafički prikaz Java koda:

```
1  int[] x = {1, 2, 3, 4, 5};
```

u memoriji bi bio:



Za identifikator `x` još kažemo da je **referenca** niza.

Jasno je da jedan `int` zauzima 4B memorije, što je ovde naznačeno kao jedan kvadratić, te stoga, zadati niz će zauzimati 5 * 4B = 20B memorije[4]. Sa `[<index>]` mi suštinski naznačavamo koliko umnožaka potrebnih bajtova treba da se pomerimo od početka našeg niza, pa tako sa:

- `x[0]` kažemo: "Idi na memorijsku lokaciju naznačenom pokazivačem `x` i pomeri se 0 * 4B = 0B udesno i zatim pročitaj naredni 4B-ni broj", s čime dobijamo broj 1.
- `x[1]` kažemo: "Idi na memorijsku lokaciju naznačenom pokazivačem `x` i pomeri se 1 * 4B = 4B udesno i zatim pročitaj naredni 4B-ni broj", s čime dobijamo broj 2.
- itd.

Zato, ukoliko iskoristimo komandu `System.out.println(x);` dobijamo `I[@<heksadecimalni_broj>`, gde:

- `I` - označava da radimo sa `int` tipovima
- `[` - označava da je u pitanju niz
- `@` - simbol 'at'
- `<heksadecimalni_broj>` - označava lokaciju prvog elementa niza

pa dakle `I[@<heksadecimalni_broj>` -> "niz int-ova na `<heksadecimalni_broj>` lokaciji".

Pored ovoga, svaki niz u Javi sa sobom vodi i skrivenu informaciju o svojoj dužini. Sa komandom **.length** dobijamo jedan `int` koji označava dužinu datog niza.

DuzinaNiza

```
1  package nekiPaket;
2
3  public class DuzinaNiza {
4
5      public static void main(String[] args) {
6          int[] arr = {-17, 123, 55};
7          int duzinaMogNiza = arr.length;
8          System.out.println("Duzina mog niza je " +
   duzinaMogNiza);
9      }
10 }
11
12 OUTPUT: Duzina mog niza je 3
```

Uobičajeni algoritmi kod rada sa nizovima

U ovom odeljku, kroz zadatke i primere, videćemo neke osnovne algoritme koji se često javljaju prilikom rada sa nizovima.

Primer1

```
1  package nekiPaket;
2
3  public class Primer1 {
4
```

```
5       public static void main(String[] args) {
6           int arr = {1, 2, 3, 4, 5};
7           System.out.println("Duzina niza je " +
    arr.length); //arr.length = 5
8           System.out.println("Treci element niza je " +
    arr[2]); //arr[2] = 3
9           arr[2] = -17; //Sada je nas niz oblika arr = [1,
    2, -17, 3, 4, 5];
10          System.out.println("Treci element niza je " +
    arr[2]); //arr[2] = -17
11          int sestiElement = arr[5]; //Greska, izasli smo
    van opsega niza
12          int minusDrugiElement = arr[-2]; //Greska,
    najmanji indeks niza je 0!
13      }
14  }
```

Zadatak 5.2 (Ispis niza)**.**

Za zadati niz ispisati sve element tog niza

Zadatak5.2

```
1   package nekiPaket;
2
3   public class Zadatak2 {
4
5       public static void main(String[] args) {
6           int arr = {1, 2, 3, 4, 5};
7           for (int i = 0; i < arr.length; i++)
8               System.out.print(arr[i] + " ");
9           System.out.println(); //Obican prelazak u novi
    red
10      }
11  }
```

```
12
13  OUTPUT:
14  1 2 3 4 5
15
```

Zadatak 5.3 (Upis u niz)**.**

Korisnik unosi ceo broj n a zatim i n celih brojeva. Smestiti te brojeve u niz.

Zadatak5.3

```java
package nekiPaket;

import java.util.Scanner;

public class Zadatak3 {

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

        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
    }
}
```

Zadatak 5.4 (Suma i Proizvod)**.**

Korisnik unosi ceo broj n a zatim i n celih brojeva. Smestiti te brojeve u niz. Izračunati sumu i proizvod tih elemenata.

Zadatak5.4

```java
package nekiPaket;

import java.util.Scanner;

public class Zadatak4 {

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

        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();

        int suma = 0;
        for (int i = 0; i < arr.length; i++)
            suma += arr[i];

        int proizvod = 1;
        for (int i = 0; i < arr.length; i++)
            proizvod *= arr[i];
    }
}
```

Konačno, možemo rešiti prvi zadatak sa početka poglavlja preko nizova

Zadatak5.1

```java
package nekiPaket;

import java.util.Scanner;

public class Zadatak1 {

```

```
6
7       public static void main(String[] args) {
8           Scanner sc = new Scanner(System.in);
9
10          int brojOcena = sc.nextInt();
11          int[] ocene = new int[brojOcena];
12          for (int i = 0; i < brojOcena; i++)
13              ocene[i] = sc.nextInt();
14
15          int suma = 0;
16          for (int i = 0; i < ocene.length; i++)
17              suma += ocene[i];
18
19
20          double prosek = (double)suma / ocene.length;
21      }
22  }
```

Zadatak 5.5 (Maksimalan element niza)**.**

Za zadati niz, naći maksimalan element tog niza

Zadatak5.5

```
1   package nekiPaket;
2
3   import java.util.Scanner;
4
5   public class Zadatak5 {
6
7       public static void main(String[] args) {
8           Scanner sc = new Scanner(System.in);
9
10          int n = sc.nextInt();
11          int[] arr = new int[n];
```

```
12          for (int i = 0; i < n; i++)
13              arr[i] = sc.nextInt();
14
15          int najveciElement = arr[0]; //Pamtimo prvi
    element kao tekuci najveci element
16          int brojac = 1; //postavljamo brojac na 1
17          for (; brojac < arr.length; brojac++) {
18              //Ako naidjemo na veci element od tekuceg,
    onda njega postavljamo kao najveci tekuci element
19              if (arr[brojac] > najveciElement)
20                  najveciElement = arr[brojac];
21          }
22
23          System.out.println(najveciElement);
24      }
25  }
```

Prodjimo korak po korak kroz iteraciju petlje gornjeg primera:

```
1  INPUT arr = [4, 5, 3, 2, 6, 3]
2
3  najveciElement = 4;
4  (brojac = 1) arr[1] = 5 > najveciElement = 4 ==>
   najveciElement = 5
5  (brojac = 2) arr[2] = 3 < najveciElement = 5
6  (brojac = 3) arr[3] = 2 < najveciELement = 5
7  (brojac = 4) arr[4] = 6 > najveciElement = 5 ==>
   najveciElement = 6
8  (brojac = 5) arr[6] = 3 < najveciElement = 6
9  OUTPUT: 6
```

Ovaj zadatak smo mogli da resimo i na još jedan način, gde izbegavamo alociranje prvog elementa niza kao tekućeg najvećeg, tako što ćemo reći da na samom početku, tekući najveći element je

negativna beskonačnost, kojoj možemo pristupiti kao konstanti **matematičke biblioteke** sa `Math.NEGATIVE_INFINITY`;

Zadatak5.5-Math

```
1  package nekiPaket;
2
3  import java.util.Scanner;
4
5  public class Zadatak5 {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9
10         int n = sc.nextInt();
11         int[] arr = new int[n];
12         for (int i = 0; i < n; i++)
13             arr[i] = sc.nextInt();
14
15         int najveciElement = Math.NEGATIVE_INFINITY;
   //Postavljamo na negativnu bekonacnost jer sve sto dodje
   posle nje je strogo vece
16         for (int i = 0; i < arr.length; i++) {
17             if (arr[i] > najveciElement)
18                 najveciElement = arr[i];
19         }
20
21         System.out.println(najveciElement);
22     }
23 }
```

Umesto da svaki put koristimo (`for`) petlju da štampamo elemente niza, umesto toga možemo da iskoristimo funkciju `Arrays.toString(<identifikator_niza>)`, koja to radi za nas automatski, uz potreban `import`:

StampanjeNiza

```
package nekiPaket;

import java.util.Scanner;
import java.util.Arrays;

public class StampanjeNiza {

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

        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();

        System.out.println(Arrays.toString(arr));
    }
}

INPUT:
n = 5
arr = [1, 2, 3, 4, 5]
OUTPUT:
[1, 2, 3, 4, 5]
```

Kopiranje nizova

Kreirajmo jednu promenjivu tipa `int` i zatim kreirajmo njenu kopiju, u smislu nove promenjive sa istom vrednošću.

```
package nekiPaket;

public class KopijaPrimitivnih {

```

```
5       public static void main(String[] args) {
6           int x = 5;
7           int y = x; //y je kopija od x.
8           System.out.println(x); //stampa 5
9           System.out.println(y); //stampa 5
10          //Promenimo vrednost originalne promenjive x na
    -123
11          x = -123;
12          System.out.println(x); //stampa -123
13          System.out.println(y); //stampa 5
14          //Promenimo vrednost kopije, promenjive y na 300
15          y = 300;
16          System.out.println(x); //stampa -123
17          System.out.println(y); //stampa 300
18      }
19  }
```

U gornjem kodu jasno možemo da vidimo da menjanje vrednosti originalne promenjive ne utiče na vrednost kopije i obratno. Pokušajmo da kreiramo analogni kod za nizove:

```
1   package nekiPaket;
2
3   import java.util.Arrays/
4
5   public class KopijaNizova {
6
7       public static void main(String[] args) {
8           int[] x = {1, 2, 3};
9           int[] y = x;
10          System.out.println(Arrays.toString(x)); //stampa
    [1, 2, 3]
11          System.out.println(Arrays.toString(y)); //stampa
    [1, 2, 3]
12          //Promenimo vrednost prvog clana originalnog niza
```

```
13          x[0] = -123;
14          System.out.println(Arrays.toString(x));
15          //stampa [-123, 2, 3]
16          System.out.println(Arrays.toString(y));
17          //stampa [-123, 2, 3]
18          //Promenimo vrednost poslednjeg clana kopije
19          y[2] = 300;
20          System.out.println(Arrays.toString(x));
21          //stampa [-123, 2, 300]
22          System.out.println(Arrays.toString(y));
23          //stampa [-123, 2, 300]
24      }
25  }
```

U ovom primeru vidimo da bilo koja izmena na originalnom nizu, utičen na isti način i na kopiju i obratno, bilo koja izmena kopije utiče na isti način i na originalni niz. Da bi smo bolje objasnili ovo ponašanje, moramo da definišemo dva nova pojma:

Definicija 5.2 (Plitka kopija)**.**

Pod plitkom, odnosno referencnom, kopijom podrazumevamo kopiranje pokazivača na zadati niz.

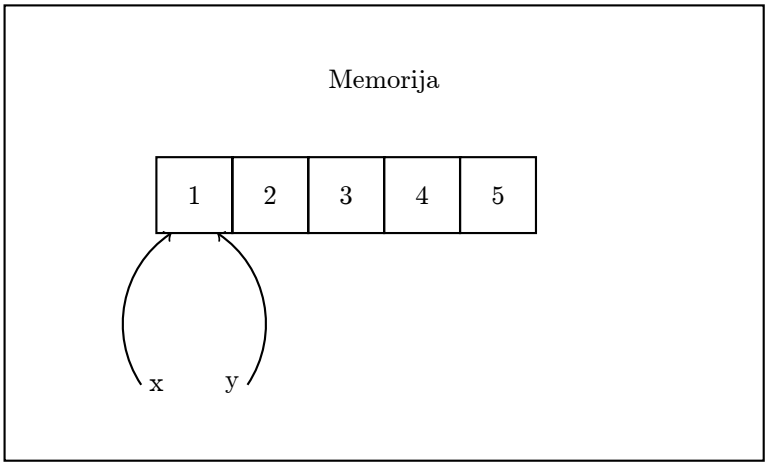
Definicija 5.3 (Duboka kopija)**.**

Pod dubokom kopijom podrazumevamo kreiranje novog niza sa identičnim elementima kao polaznim.

Da bi smo bolje razumeli razliku izmedju ovih kopija, pogledajmo kako se gornji kod realizuje u memoriji. Naime, prilikom izvršavanja linije:

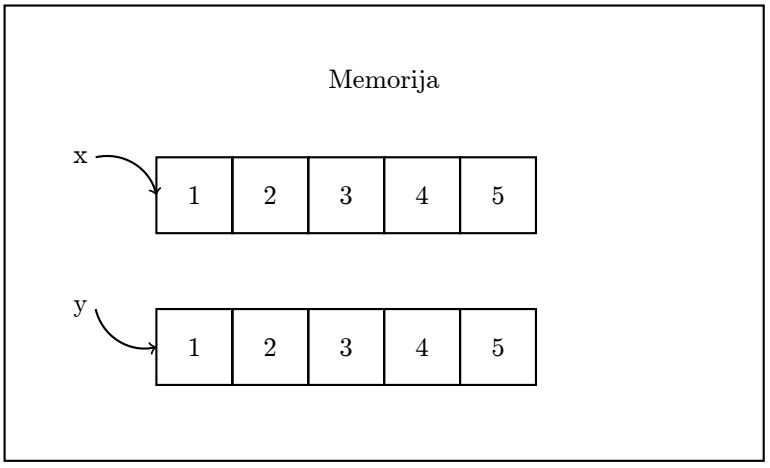
```
int[] y = x;
```

jedino što mi radimo je kreacija novog pokazivača na isti niz. Grafički prikazano, to bi izgledalo:



Sada se jasno vidi, da npr. drugi elment "niza y" je ista stvar kai i drugi element "niza x", tj. da `x[1]` i `y[1]` pokazuju na istu memorijsku lokaciju!

Da bi smo stvarno kreirali pravu, duboku kopiju nekog niza, tj. da dobijemo memorijsku sliku:



koristimo sledeći algoritam:

DubokoKopiranje

```
package nekiPaket;

```

```
3  import java.util.Arrays;
4
5  public class DubokoKopiranje {
6
7      public static void main(String[] args) {
8          int x = {1, 2, 3};
9          int y = new int[x.length]; //Kreiramo prazan niz
   iste duzine
10         /* Svaki element zadatog niza
11         *  ponaosob postavljamo na istu poziciju
12         *  u novokreiranom nizu
13         */
14         for (int i = 0; i < y.length; i++)
15             y[i] = x[i];
16
17         x[0] = -123;
18         y[2] = 300;
19         System.out.println(Arrays.toString(x));
   //Ispisuje [-123, 2, 3]
20         System.out.println(Arrays.toString(y));
   //Ispisuje [1, 2, 300]
21     }
22 }
```

Sada, pošto u svakoj iteraciji radimo baš sa promenjivima tipa `int`, više nemamo kopiranje po referenci, vec imamo kopiranje po vrednosti, te memorija izgleda kao na prethodnoj slici i menjanje vrednosti nego elementa jednog niza, ne utiče nikako na elemente drugog niza.

Foreach petlja

Sada kada smo upoznati sa nizovima, možemo se pozabaviti poslednjim oblikom petlji koje smo naveli u prethodnom poglavlju - **foreach** petljom.

Definicija 5.3 (Foreach petlja)**.**

Foreach petlja, odnosno enhanced for petlja, je petlja koja se preimenjuje na dobro uredjene kolekcije podataka, i koja iterira kroz kolekciju od početnog ka krajnjem elementu.
Pod drobro uredjenom kolekcijom podrazumevamo nizove ili klase koje implementiraju Iterator interfjes.

Šta se tačno podrazumeva pod `Iterator interfejsom` ostavljamo za kasnije, kada se budemo detaljno upoznali sa svim pojmovima klasa, a za sada izlažemo kako se piše `foreach` petlja uopšteno:

```
for (<tip_podatka_kolekcije> <identifikator> :
<kolekcija>) {
    //Telo petlje
}
```

U početnoj iteraciji petlje, zadati `identifikator` će uzeti vrednost prvog elementa kolekcije, u narednoj vrednost drugog elementa kolekcije i tako dalje sve dok ne predje kroz sve elemente kolekcije. Ovakva petlja nam omogućava da iteriramo kroz kolekciju bez brige o načinu pristupanja njenim elementima, ali zato dolazi sa ogromnom manom, a to je da nikako ne možemo da predjemo u proizvoljnu iteraciju ili da olako pristupimo nekom drugom elementu kolekcije.

Primer ispisa svih elemenata niza koristeći se foreach petljom bi izgledao:

```
package nekiPaket;

public class ForEachPetlja {
```

```
4
5       public static void main(String[] args) {
6           int x = {1, 2, 3, 4, 5};
7           for (int elem : x) {
8               System.out.print(elem + ", ");
9           }
10          System.out.println();
11      }
12  }
13  OUTPUT:
14  1, 2, 3, 4, 5,
15
```

Nizovi nizova - Matrice

Pored `literala`, kao elementi nizova mogu i da se jave `pokazivači` zadatog tipa podataka i u tom slučaju, svaki element zadatog niza bi bio pokazivač na zaseban niz tog tipa podatka. Time bi smo kreirali **niz nizova**, odnosno jednu **matricu** čiji matrični elementi bi bili zadatog tipa podatka. U daljem tekstu ovog zaglavlja, koristićemo izraze `matrica` i `niz nizova` naizmenično, a na kraju ćemo izneti 'razliku' izmedju ova dva pojma.

Kreacija matrica je potpuno analogna kreacija nizova i zahteva dodavanjem jednog para uglastih zagrada `[]`:

```
1  <tip_podatka>[][] <identifikator> = {<prvi_niz>,
   <drugi_niz>, ...};
```

odnosno praktično:

```
1  int[][] matricaIntova1 = {{1, 2, 3}, {-123, 17, 300},
   {55, 6}, {100}};
```

```
int[][] matricaIntova2 = new int[2][3]; //Kreira se matrica 2x3
```

Primetimo da je u prvom primeru, gde eksplicitno zadajemo elemente matrice, mi suštinski kreiramo jedan niz, čiji je svaki element niz `int` literala, dok u drugom primeru suštinski kreiramo:

```
int[][] matricaIntov2 = {{0, 0, 0}, {0, 0, 0}};
```

U Javi, elementi matrica, što su obični nizovi, ne moraju svi biti istog broja elemenata, kao što smo mogli da vidimo u prvom primeru. Ovo odstupa od ponašanja matrica u matematici[5] i ovakva konstrukcija se retko vidja u praksi.
Već smo rekli da je svaki element matrice zapravo jedan pokazivač na niz, te za zadate komande dobijamo zadate ispise:

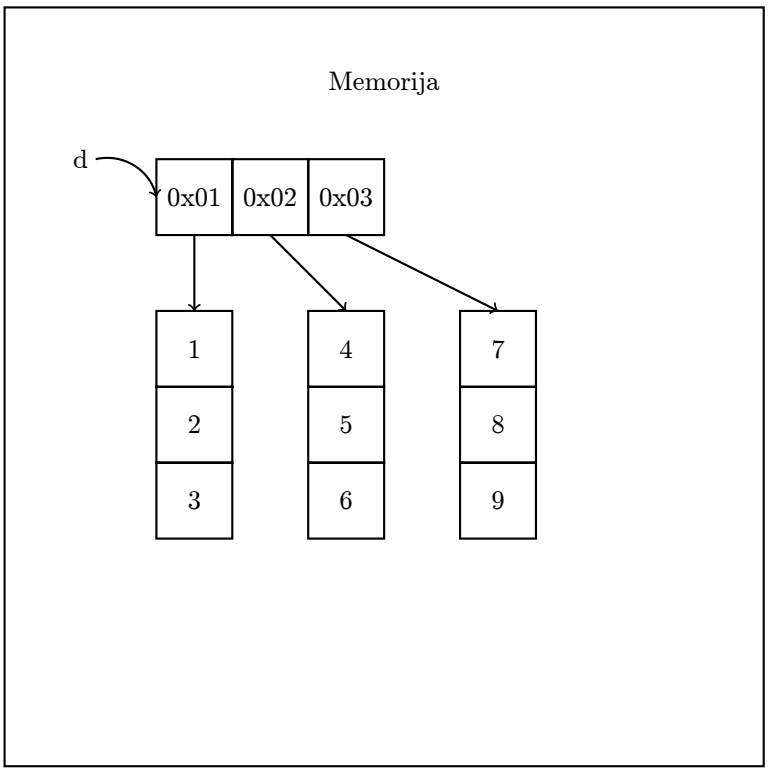
```
package nekiPaket;

import java.util.Arrays;

public class IspisMatrica {

    public static void main(String[] args) {
        double d = new double[3][3];
        System.out.println(d); //Output: [[D@<heksadecimalan_broj1>
        System.out.println(d[0]); //Output: [D@<heksadecimalan_broj2>
        System.out.println(Arrays.toString(d));
        /*Output:
        [[D@<heksadecimalanbroj2, [D@<heksadecimalanbroj2, [D@<heksadecimalanbroj3
        [D@<heksadecimalanbroj4, [D@<heksadecimalanbroj5]
    }
```

```
16  }
```

Memorijska slika jedne 3x3 matrice:

```
1  double[][] d = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

bi izgledala:



Dakle svaki element `double[] d` niza, je jedan pokazivač na `double[] d` niz, što je naznačeno kao `double[][] d`.

Svi algoritmi koji su se odnosi na `nizove`, potpuno analogno se odnose i na `matrice`, pazeći na to da nam je sada svaki element niza jedan niz, te moramo da koristimo ugnježdene petlje.

UpisIspisMatrice

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

```
6
7       public static void main(String[] args) {
8           Scanner sc = new Scanner(System.in);
9
10          //Unos elemenata u matricu
11          int n = sc.nextInt();
12          int m = sc.nextInt();
13          int[][] mat = new int[n][m];
14          for (int i = 0; i < mat.length; i++) {
15              for (int j = 0; j < mat[i].length; j++) {
16                  //mat.length = n, mat[0].length = m
17                  mat[i][j] = sc.nextInt();
18              }
19          }
20
21          //Ispis elemenata u matrici
22          for (int i = 0; i < mat.length; i++) {
23              for (int j = 0; j < mat[i].length; j++) {
24                  System.out.print(mat[i][j] + ", ");
25              }
26              System.out.println();
27          }
28          /*
29          * Odnosno preko foreach petlje:
30          * for (int[] row : mat) {
31              * for (int col : mat) {
32                  * System.out.print(col);
33              * }
34              * System.out.println();
35          * }
36          */
37      }
38  }
39  Input:
40  n = 3
41  m = 3
```

```
1 2 3 4 5 6 7 8 9
OUTPUT:
1 2 3
4 5 6
```

Do sada smo koristili pojmove `matrice` i `niza nizova` proizvoljno. Ovo možemo da radimo pošto oba ova pojma predstavljaju jednu valjanu interpretaciju onoga što se dešava u memoriji, tako u zavisnosti od potrebe odnosno u zavisnosti od načina korišćenja, možemo pričati o matrici, nizu vektora, nizu nizova itd.

Ovaj postupak 'dodvanja' uglastih zagrada možemo da nastavimo u nedogled i tako da pravimo niz čiji su svi elementi nizovi čiji su svi elementi nizovi, odnosno '3D matrice', tj. tenzore itd.

```
int[][][] x = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}, {{9, 10}, {11, 12}}};
```

---

1. U matematici, pod nizovima podrazumevamo funkcije oblika $f : \mathbb{N} \to X$, gde je $\mathbb{N}$ skup prirodnih brojeva, tj. funkcije čiji je domen skup prirodnih brojeva. U programiranju, nizove možemo da posmatramo kao direktne slike matematičkih nizova uz razliko što su nizovi u programiranju ograničeni, a matematički nizovi su funkcije nad beskonačnim (prebrojivim) domenom. ↩
2. Stil koji se koristi u C programskom jeziku, jednom od najuticajnijih programskih jezika. Iako dozvoljen, treba izbegavati ovaj stil pisanja u Javi. ↩
3. Ovo ne znači da svi programski jezici koji automatski održavaju memoriju nemaju mogućnost rada sa pokazivačima. C++ je programski jezik koji će automatski da održava memoriju ako mu drugačije ne naznačimo, a opet moguće je kreirati pokazivače kao tip podatka. ↩

4. Da smo imali neki drugačiji tip podatka, npr. niz short-ova, zauzeli bi smo 5 * 2B = 10B memorije itd. ↩
5. U matematici, svi redovi matrice moraju biti iste dužine i sve kolone matrice moraju biti iste dužine. ↩