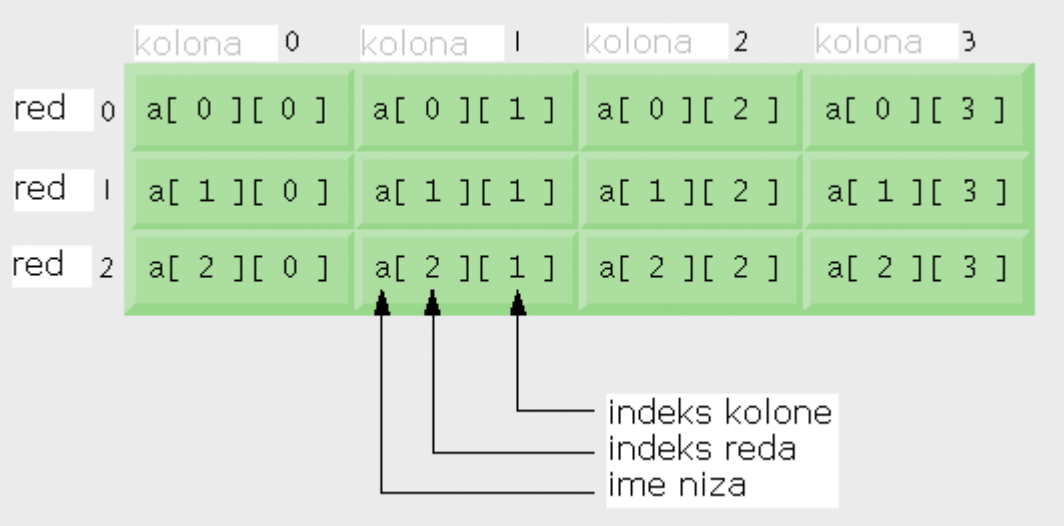
Višedimenzionalni nizovi



Deklarisanje višedimenzionalnog niza

```
int c[][] = new int[3][4]; // 3 reda i 4 kolone
```

U javi se višedimenzionalni niz predstavlja kao niz nizova. Kada je memorija alocirana, podrazumijevane vrijednosti za elemente niza su **0** za brojeve, **false** za tip boolean i **null** za reference.

Primjer: Deklaracija i inicijalizacija dvodimenzionalnog niza b

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

- 1 i 2 inicijalizuju b[0][0] i b[0][1]
- 3 i 4 inicijalizuju b[1][0] i b[1][1]

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

- red 0 sadrži elemente 1 i 2
- row 1 sadrži elemente 3, 4 i 5

Obratite pažnju da dužine redova ne moraju biti iste.

Primjer: Kreiranje dvodimenzionalnog niza pomoću izraza za kreiranje nizova

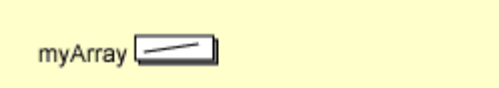
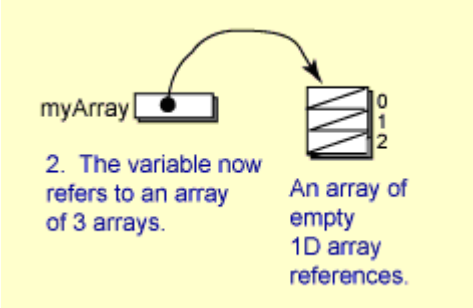
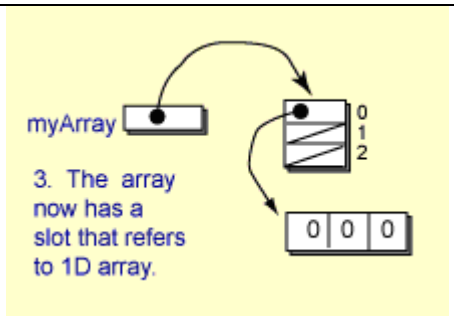
- 3x4 niz

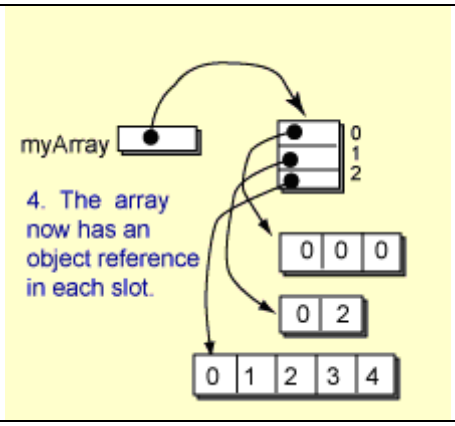
```
int b[][];
b = new int[ 3 ][ 4 ];
```

- Redovi mogu imati različit broj kolona

```
int b[][];
b = new int[ 2 ][ ];   // kreira 2 reda
b[ 0 ] = new int[ 5 ]; // kreira 5 kolona u prvom redu
b[ 1 ] = new int[ 3 ]; // kreira 3 kolone u drugom redu
```

Primjer: U tabeli je prikazan postupak deklarisanja 2D niza i izgleda memorije poslije izvršenja svakog od koraka.

`int[][] myArray;  // 1.` Deklarišemo promjenljivu myArray koja će biti referenca na neki 2D niz. U ovom trenutku nije poznato broj redova i kolona tog niza.	
Da bi kreirali niz od 3 reda: `myArray = new int[3][] ;  // 2.` Sada `myArray` referencira niz od 3 elementa. Svaki od elemenata tog niza biće referenca na niz cjelih brojeva (tj. na int []). Međutim, u ovom trenutku, nijedna od tri elementa ne referencira na objekat, pa svi imaju vrijednost null.	
Jedan način da se kreira red 0 je: `myArray[0] = new int[3] ;  // 3.` Ova naredba kreira 1D niz i postavlja referencu na njega u polje 0 promjenljive `myArray`. Sva polja kreiranog 1D niza su postavljena na 0.	

Moguće je već kreirane 1D nizove pridružiti redovima niza `myArray`:
 	
`int[] x = {0, 2};`
`int[] y = {0, 1, 2, 3, 4} ;`
 	
`myArray[1] = x ;`
`myArray[2] = y ;  // 4.`
	

Promjenljiva `length` za 2D nizove daje broj redova niza. U primjeru iz tabele, `myArray.length` vraća 3. Pri štampanju ili obilasku pojedinačnog reda, mora se koristiti dužina tog reda npr. `myArray[2].length`. U primjeru iz tabele imamo da je `myArray[0].length` jednak 3, `myArray[1].length` jednako je 2, a `myArray[0].length` je 5. Obratite pažnju kako je u sljedećem primjeru riješeno štampanje elemenata 2D niza. Slična tehnika se primjenjuje i u slučaju 3D ili višedimenzionalnih nizova.

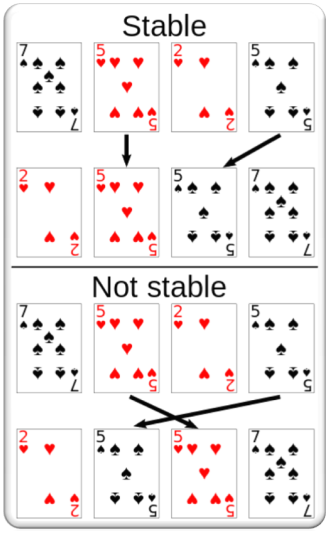
Primjer:

```
// dekaracija i inicijalizacija 2D niza
   int[][] uneven = { { 1, 9, 4 }, { 0, 2}, { 0, 1, 2, 3, 4 } };
```

```
  // stampanje niza, red po red
  for ( int row=0; row < uneven.length; row++ )
  {
    System.out.print("Red " + row + ": ");
    for ( int col=0; col < uneven[row].length; col++ )
      System.out.print( uneven[row][col] + " ");
    System.out.println();
  }
}
```

Sortiranje

- Algoritam sortiranja (Sorting algorithm) - algoritam koji elemente liste postavlja u određeni poredak (najčešće leksikografski)
- Formalnije: ulaz je neki raspored elemenata, a izlaz je neka permutacija ulaza
- Efikasno sortiranje je važno jer daje izlaz koji je ljudima razumljiviji, predstavlja podatke u nekom kanonskom obliku, a često je i neophodna korak za upotrebu drugih algoritama.
- Sortiranje uvodi mnoge važne tehnike programiranja
- Algoritmi sortiranja se često klasifikuju po:
  - Složenosti (engl. Computational complexity)
    - Najgori (worst), prosječni (average) i najbolji (best) slučaj
  - Upotrebi memorije (engl. Memory usage)
  - Rekurzivni ili nerekurzivni
  - Stabilnosti
    - Stable sorting algorithms
    - Čuvaju relativni poredak elemenata sa jednakim vrijednostima
    - Ako su dva elementa jednaka, njihov relativni poredak biće očuvan
    - Vidi sliku desno za primjer stabilnog i nestabilnog algoritma
  - Da li koriste operacije poređenja (engl. comparison sort) ili ne
  - Opšti metod algoritma
    - Umetanje (engl. insertion), razmjena (engl. exchange) – bubble sort i quicksort, selekcija (engl. selection) – heapsort, spajanje (engl. merging), serijski ili paralelni (engl. serial ili parallel)…



Sortiraćemo niz od n elemenata

- **Sortiranje selekcijom (Selection sort)**
  - Veoma jednostavan i neefikasan algoritam
  - Best, worst i average slučaj: n^2
  - Memorija: 1 (konstantna, samo za min element)
  - Stabilan: Ne, Metod: Selection
  - http://en.wikipedia.org/wiki/Selection_sort

```
for (j = 0; j < n-1; j++) {
    /* nadji min element u nesortiranom dijelu a[j .. n-1] */
    iMin = j;
    for ( i = j+1; i < n; i++) {
        if (a[i] < a[iMin]) iMin = i;
    }
    if (iMin != j) swap(a[j], a[iMin]); // zamjena mjesta
}
```

- **Sortiranje uzastopnim razmjenama (Bubble sort)**

- Ponavljamo prolazak kroz listu, poredimo parove susjednih elemenata i zamjenjujemo im mjesta ako nisu u pravom poretku
- Best slučaj: n, worst i average slučaj: n^2
- Memorija: 1, Stabilan: Da, Method: Exchange
- http://en.wikipedia.org/wiki/Bubble_sort

```
procedure bubbleSort( A : list of sortable items )
    repeat
      swapped = false
      for i = 1 to length(A) - 1 inclusive do:
        /* if this pair is out of order */
        if A[i-1] > A[i] then
          /* swap them and remember something changed */
          swap( A[i-1], A[i] )
          swapped = true
        end if
      end for
    until not swapped
end procedure
```

- **Sortiranje umetanjem (Insertion sort)**
  - Kreira finalni sortirani niz, imitirajući dodavanje jednog po jednog elementa
  - Best slučaj: n, worst i average case: n^2
  - Memorija: 1, Stabilan: Da, Metod: Insertion
  - http://en.wikipedia.org/wiki/Insertion_sort

```
for i ← 1 to i ← length(A)-1
{
  valueToInsert ← A[i]
  holePos ← i
  while (holePos > 0 and valueToInsert < A[holePos - 1])
  {
    A[holePos] ← A[holePos - 1] // shift the larger value up
    holePos ← holePos - 1       // move the hole position down
  }
  A[holePos] ← valueToInsert
}
```

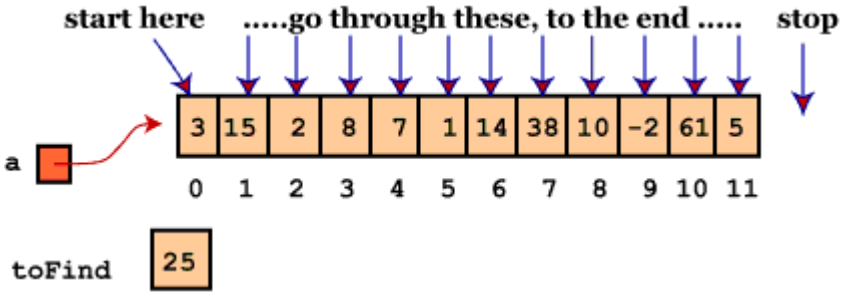
Pretraživanje

- Algoritam pretraživanja (engl. Search algorithm) je algoritam pronalaženja elementa sa datim svojstvima u datoj kolekciji objekata

Kao i kod sortiranja, prtetraživanje izvodimo u nizu

- Linearno pretraživanje (engl. Linear search)
  - Metod pronalaženja određene vrijednosti u listi
  - Provjerava redom, jedna po jedan, sve elemente
  - Sve dok se ne pronađe željeni element
  - Ili dok ne dođemo do kraja liste
  - Worst i average slučaj: O(n)

```
for each item in the list:
     if that item has the desired value,
         stop the search and return the item's location.
 return nothing.
```



- Binarno pretraživanje (engl. Binary search)
  - Pronalazi poziciju željene vrijednosti unutar **sortiranog** niza
  - U svakom koraku, upoređuje vrojendost sa srednjim elementom niza
  - Sada se ponavlja akcija za lijevi ili desni dio niza
  - Average slučaj: O(log(n))
  - Može se napisati rekurzivno ili iterativno
  - Rekurzivna verzija

```
int binary_search(int A[], int key, int imin, int imax)
{
  if (imax < imin)// skup je prazan, vracamo da nista nije nadjeno
     return KEY_NOT_FOUND;
  else
  {
     // odredimo srednji element, dijeljenjem na pola
     int imid = midpoint(imin, imax);
     if (A[imid] > key) // key je u donjoj polovini skupa
        return binary_search(A, key, imin, imid-1);
     else if (A[imid] < key) // key je u gornjoj polovini skupa
        return binary_search(A, key, imid+1, imax);
     else
        return imid; // key je pronađen
  }
}
```

- Iterativna verzija

```
int binary_search(int A[], int key, int imin, int imax)
{
   // nastavi sa traženjem dok [imin,imax] nije prazan
   while (imax >= imin)
   {
      /* odredimo srednji element */
      int imid = midpoint(imin, imax);
      // odredimo koji podniz da pretrazujemo
      if (A[imid] < key)
         // promjena min indeksa za pretragu gornje polovine niza
         imin = imid + 1;
      else if (A[imid] > key)
         // promjena min indeksa za pretragu donje polovine niza
         imax = imid - 1;
      else
         // key je pronađen na poziciji imid
         return imid;
  }
  return KEY_NOT_FOUND;
}
```