OVIJALNIKI TIPOV, NIZI IN POLJA

Programiranje 1 - RIN, BF, MR
Računalništvo 1 - MA, MF

Predavatelj: Niki Hrovatin

Ovijalniki tipov

Primitivni tipi shranjujejo osnovne količine, ki jih podpira jezik

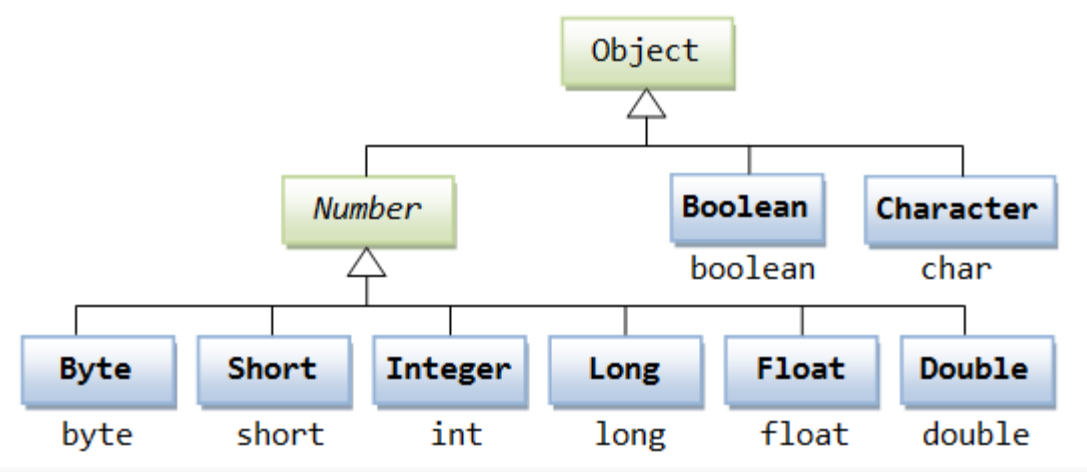
- Zasnovani za učinkovito delovanje
  - Niso predmeti
  - Ponujajo osnovne funkcionalnosti
- Včasih je zaželjena uporaba primitivnih tipov kot predmet
  - Več funkcionalnosti
  - Prenos po referenci
  - Standardne podatkovne strukture običajno delujejo s predmeti

Ovijalniki tipov - Type wrappers

- Razredi, ki ovijejo primitiven tip znotraj predmeta.

Ovijalniki tipov v Javi

Razredi namesto primitivnih tipov (**ovijanje - wrapping**)



Character

Character je ovijalnik okoli `char`.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Character.html

Primer:

```
char[] carArray = {'A', '5', 'B', '3', '9', 'Z', '1'};
int sum = 0;

for (int i = 0; i < carArray.length; i++) {

    if (Character.isDigit(carArray[i])) {  // check if digit
        sum += Character.getNumericValue(carArray[i]); //convert digit to int
    } else {
        System.out.println(carArray[i] + " is not a digit.");
    }
}
System.out.println("Sum of digits: " + sum);
```

Character

Uporabne metode:

```
boolean isLetter(char ch);
boolean isDigit(char ch);
boolean isWhitespace(char ch);
boolean isUpperCase(char ch);
boolean isLowerCase(char ch);
boolean isISOControl(char ch);
char toUpperCase(char ch);
char toLowerCase(char ch);
String toString(char ch);
```

Kaj pa posebni znaki? : `(\t,\b,\n,\r)`

Boolean

Boolean je ovijalnik okoli `boolean`.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Boolean.html

Primer:

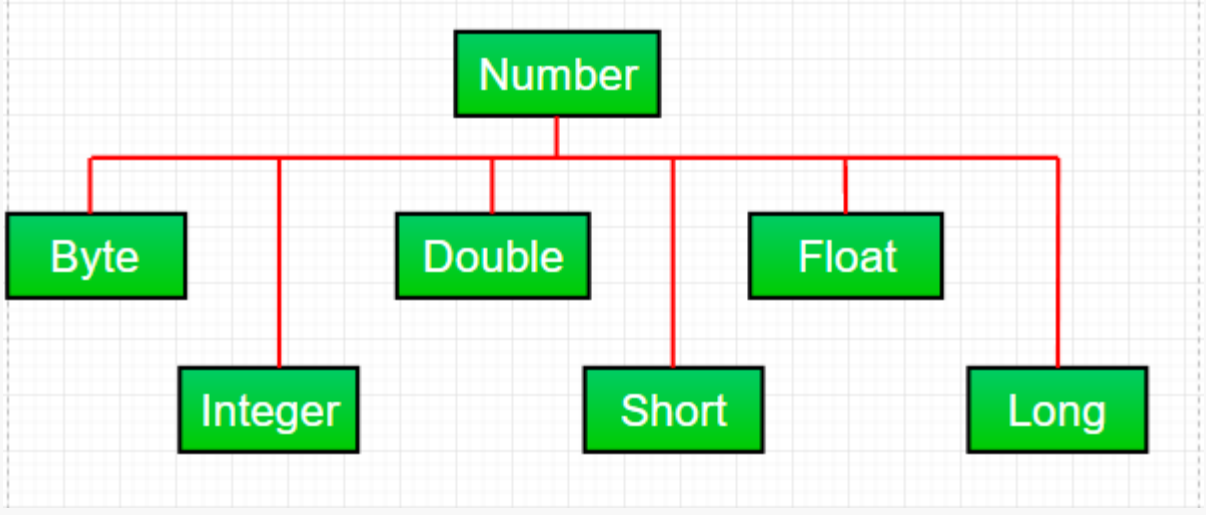
```
String input = "FALSE";

if (Boolean.valueOf(input)) {
    System.out.println("The condition is true.");
} else {
    System.out.println("The condition is false.");
}
```

Number

`Number` je nadrazred platformnih razredov, ki predstavljajo številske vrednosti, kateri ovijajo primitivne tipe:

- `byte`, `double`, `float`, `int`, `long`, and `short`.



Integer

Integer je ovijalnik okoli `int` vrednosti.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Integer.html

Uporabne lastnosti:

```
int a = Integer.MAX_VALUE;    // 2147483647
int b = Integer.MIN_VALUE;    // -2147483648
```

- Podobno velja za druge podrazrede razreda Number

Integer

Uporabne metode:

```
int number = Integer.parseInt("123"); // 123
String str = Integer.toString(123); // "123"

int max = Integer.max(10, 20); // 20
int min = Integer.min(10, 20); // 10

String binary = Integer.toBinaryString(10); // "1010"
String hex = Integer.toHexString(255); // "ff"
String octal = Integer.toOctalString(10); // "12"

int bitCount = Integer.bitCount(10); // 2
```

- Podobno velja za druge podrazrede razreda Number, check the docs!

Autoboxing / auto-unboxing

Autoboxing je samodejna ovijanje (enkapsulacija) primitivnega tipa v njegov ustrezni ovijalnik tipa

- Auto-unboxing pa je proces samodejnega odvijanja (dekapsulacija)

Brez autoboxing:

```
Integer iOb = Integer.valueOf(100);
int i = iOb.intValue();
```

Autoboxing:

```
Integer iOb = 100; // autobox an int
int i = iOb; // auto-unbox
```

Uporaba `null`

Pri programiranju moramo včasih predstaviti **odsotnost vrednosti**.

- `0`, `false` in `""` so dejanske vrednosti, ne manjkajoče vrednosti.
- V Javi **`null` pomeni, da referenca ne kaže na noben objekt**.
- `null` lahko uporabljamo pri **referenčnih tipih**, ne pa pri primitivnih tipih.

```
Integer grade = null; // ocena še ni na voljo
String name = null;   // ime še ni na voljo
int count = null;     // NAPAKA: int je primitiven tip
```

Pomembno:

- Pred uporabo reference, katere vrednost je lahko `null`, preverimo, ali je njena vrednost `null`.

```
Integer test = null;
int a = test;    // NullPointerException
```

Uporaba `null`: Manjkajoče meritve

Predstavljajmo si, da s senzorjem merimo temperaturo. Nekatera meritev morda **ni na voljo**.

Uporaba primitivnih tipov:

```
int[] temperatures = {21, 0, 24, 19};
```

Ali `0` pomeni **0 °C** ali **manjkajočo meritev**?

Uporaba ovojnih razredov:

```
Integer[] temperatures = {21, null, 24, 19};

for (int i = 0; i < temperatures.length; i++) {
    if (temperatures[i] == null) {
        System.out.println("Meritev ni na voljo");
    } else {
        System.out.println(temperatures[i] + " °C");
    }
}
```

Privzete vrednosti elementov polj:

```
int[] a = new int[3];        // {0, 0, 0}
Integer[] b = new Integer[3]; // {null, null, null}
```

Uporaba `null`: Metode brez rezultata

Metoda, ki išče določeno vrednost, je morda **ne najde**.

Primer: vrni prvo pozitivno število v polju.

```
public static Integer findPositive(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] > 0) {
            return arr[i]; // samodejno ovijanje int v Integer
        }
    }
    return null; // pozitivno število ni bilo najdeno
}
```

Uporaba rezultata:

```
int[] values = {-3, -8, -1};
Integer result = findPositive(values);

if (result == null) {
    System.out.println("Pozitivno število ni bilo najdeno");
} else {
    System.out.println("Najdeno: " + result);
}
```

Zakaj `Integer` namesto `int`? Metoda lahko vrne število ali `null`.

Razumevanje `null`

Koncept manjkajočih vrednosti v programiranju

- Programi morajo včasih predstaviti **odsotnost vrednosti ali rezultata**.
- To moramo razlikovati od veljavnih vrednosti, kot so `0`, `false` ali `""` (prazen niz).
- Različni programski jeziki različno obravnavajo manjkajoče vrednosti.

Kaj pomeni `null` v Javi?

- `null` je poseben literal, ki označuje, da **referenca ne kaže na noben objekt**.
- Samo referenčni tipi lahko vsebujejo `null`. Primitivni tipi ga ne morejo.

Strings - Nizi

Znaki

- "Gradniki" za ne-numerične podatke
- 'a', '&', '4'

Nizi

- Zaporedje znakov, obravnavano kot enota
- Predmeti razreda String
- Razred String je nespremenljiv (immutable).
  - Nobena metoda razreda String ne spremeni notranjega stanja predmeta.

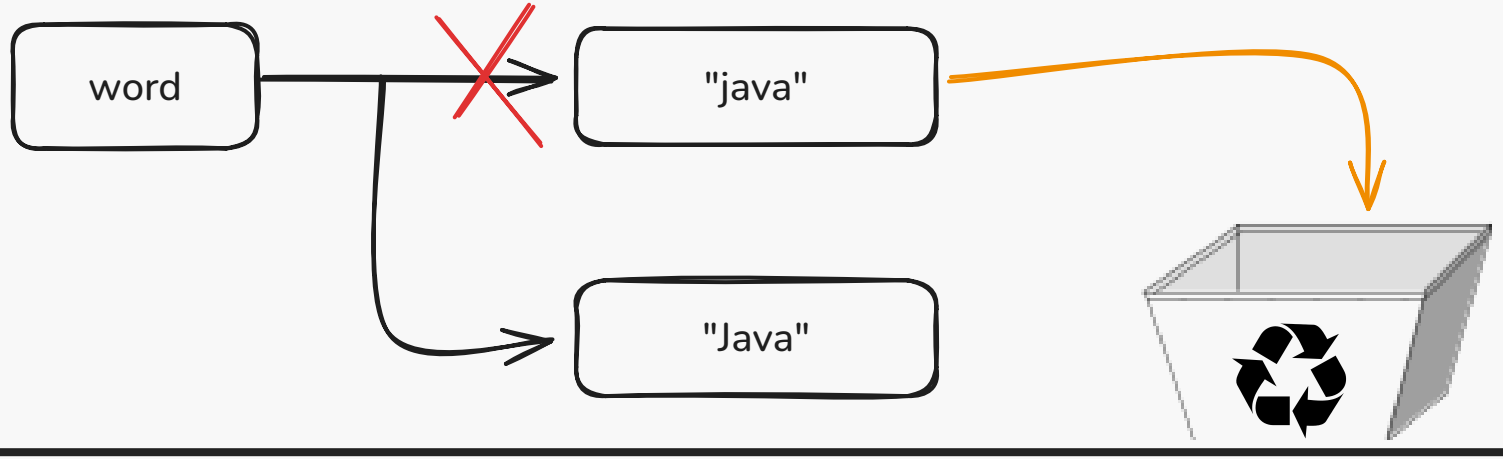
Strings -- nespremenljivost (immutability)

Ko je niz enkrat ustvarjen, ga ni mogoče spremeniti:

- Metode, za manipulacijo vsebine niza, effektivno ustvarijo novi niz z modificirano vsebino

Ponavljajoče spreminjanje nizov (`String`), zlasti znotraj zank, lahko povzroči ustvarjanje nepotrebnih objektov.

```
String word = “java";
char ch = Character.toUpperCase(word.charAt (0));
word =  ch + word.substring (1);
```



Konstrukcija Niza

Nizni **literali**

- Podaj niz v narekovajih
- Java bo samodejno ustvarila objekt tipa String

```
String s1 = "abc"; // uporaba niza kot literala
```

Prazni niz:

```
String s = new String();
```

String iz char polja:

```
char[] chars = { 'a', 'b', 'c' };
String s = new String(chars);  // vsebuje "abc"
```

Iz String predmeta:

```
String s1 = "abc"; // uporaba niza kot literala
String s2 = new String(s1);
```

Nizi - operacije

Dolžina niza:

- Število znakov, ki jih vsebuje

```
String s1 = "abc";
s1.length() // output: 3
```

Konkatenacija nizov:

- združi več objektov tipa String in ustvari nov objekt tipa String
- pretvori primitivne tipe v String predmet

```
int starost = 9;
String s = "On je star " + starost + " let.";
System.out.println(s); // izhod: On je star 9 let.
```

Nizi - operacije

Ekstrakcija znakov:

- `char charAt(int index)`
  - izvleče posamezen znak iz niza (index - mesto znaka v nizu)

```
char ch;
ch = "abc".charAt(1); // 'b'
```

- `char[ ] toCharArray( )`
  - pretvori objekt tipa String v polje znakov

Iskanje:

- `int indexOf( char ch )`
  - Poišče prvo pojavitev znaka ali podniza.
- `int lastIndexOf( char ch )`
  - Poišče zadnjo pojavitev znaka ali podniza.

Nizi - enakost

Nizi so predmeti

- operator == preverja enakost referenc pri uporabi s predmeti
  - preveri, ali dve referenci kažeta na isto mesto v pomnilniku
- za primerjavo vsebine niza uporabi `s1.equals(s2)`

```
String s1 = "hello";
String s2 = "hello";
String s3 = new String("hello");

System.out.println(s1 == s2);  // TRUE, oba kažeta na isti objekt
System.out.println(s1 == s3);  // FALSE, s3 je nov objekt tipa String z isto vsebino
System.out.println(s1.equals(s3));  // TRUE, ker preverja enakost vsebine
```

- `s1` in `s2` kažeta na isti objekt, saj Java **internira** String literale
  - zmanjša pomnilniške zahteve, če je veliko podvojenih nizov

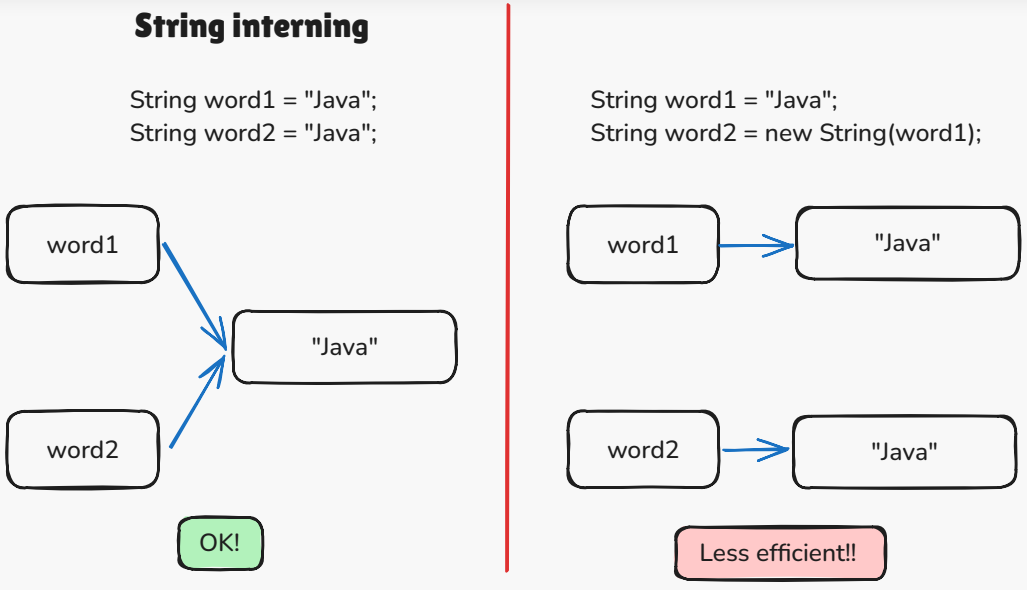
Interniranje nizov - String interning

`s1` in `s2` kažeta na isti objekt, saj Java **internira** String literale

- zmanjša pomnilniške zahteve, če je veliko podvojenih nizov

```
String s1 = "hello";
String s2 = "hello";
String s3 = new String("hello");

System.out.println(s1 == s2);  // TRUE, oba kažeta na isti objekt
System.out.println(s1 == s3);  // FALSE, s3 je nov objekt tipa String z isto vsebino
```



Nizi - operacije

Ekstrakcija podniza:

- `String substring(int startIndex, int endIndex)`

```
String text = "Hello, World!";
String result = text.substring(7, 12);  // Izvleče znake od indeksa 7 do 11
System.out.println(result);  // Izhod: "World"
```

Spreminjanje velikosti znakov:

- `String toLowerCase( )`
- `String toUpperCase( )`

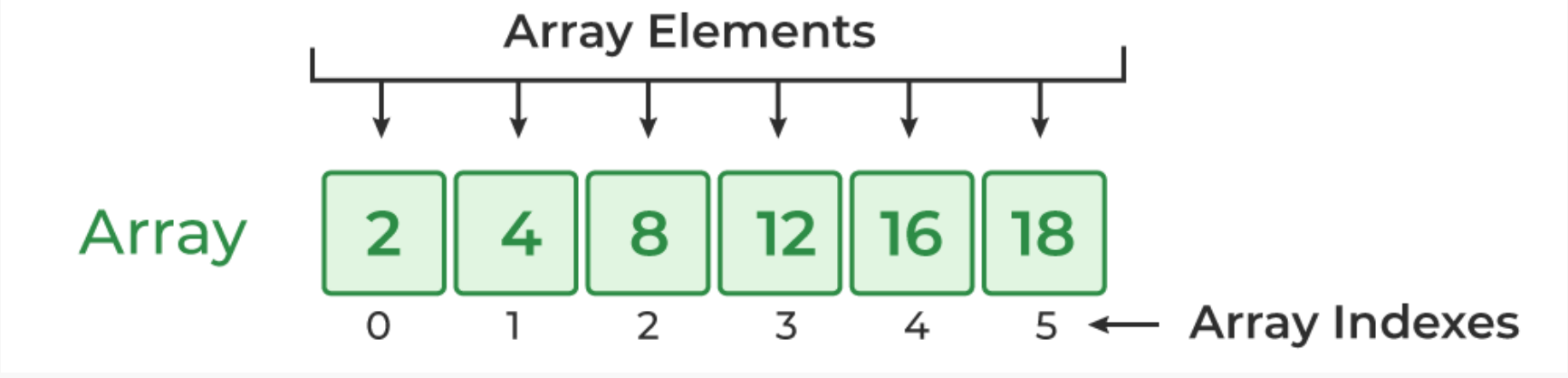
Še veliko drugih metod...

- Check the docs

Polja - Arrays

Polja so predmeti

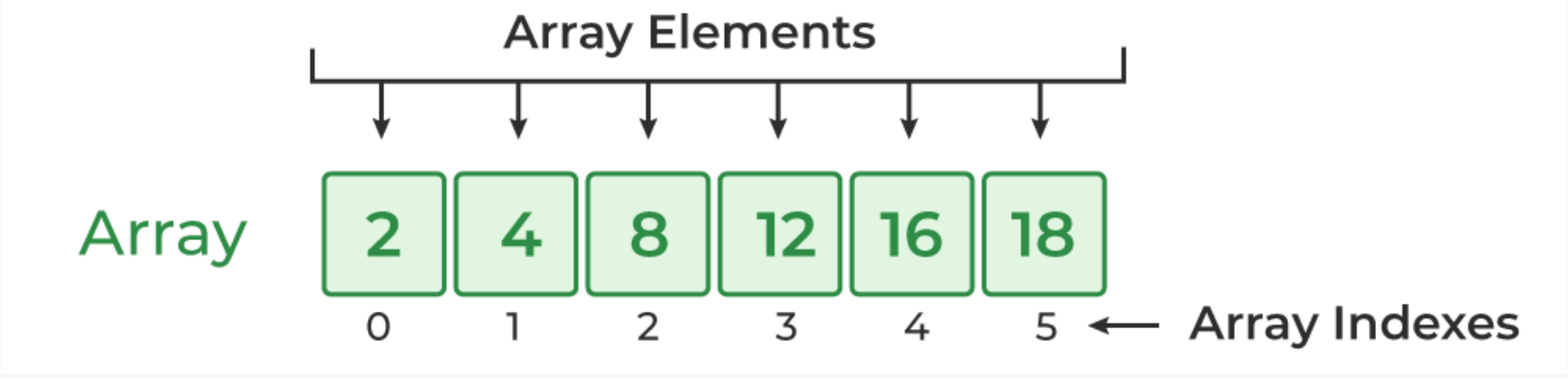
- V Javi so polja *predmeti*, in `array.length` je **final** lastnost.
- `length` je določena ob ustvarjanju polja in je ni mogoče spreminjati - **velikost polja je nespremenljiva**.



Polja - Arrays

Učinkovit dostop do elementov

- **Zaporedno shranjevanje**:
  - Elementi so shranjeni v zaporednih pomnilniških lokacijah za hiter dostop.
- **Primitivi**:
  - Vrednosti so neposredno shranjene na teh lokacijah.
- **Predmeti**:
  - Lokacije vsebujejo reference (kazalce) na dejanske predmete, ki so shranjeni drugje v pomnilniku.



Polja - iskanje

Kako preveriti, ali polje vsebuje določeno vrednost?

```
int[] stevilke = {10, 2, 44, 5, 23, 100, 13, 88, 16, 89};
int cilj = 89;

for (int i = 0; i < stevilke.length; i++) {
    if (stevilke[i] == cilj) {
        System.out.println(cilj + " je vsebovan v polju na indeksu " + i);
    }
}
```

- Linearno iskanje pregleduje elemente enega za drugim, dokler ne najde iskanega elementa ali doseže konca polja.
  - Če iskanega elementa ni v polju, je treba pregledati celotno polje.

Polja - iskanje

Kako preveriti, ali polje vsebuje določeno vrednost?

```
int[] stevilke = {10, 2, 44, 5, 23, 100, 13, 88, 16, 89};
int cilj = 89;

for (int i = 0; i < stevilke.length; i++) {
    if (stevilke[i] == cilj) {
        System.out.println(cilj + " je vsebovan v polju na indeksu " + i);
    }
}
```

- Linearno iskanje pregleduje elemente enega za drugim, dokler ne najde iskanega elementa ali doseže konca polja.
  - Če iskanega elementa ni v polju, je treba pregledati celotno polje.

Ali lahko iščemo boljše / hitreje?

Polja - iskanje

Kako preveriti, ali polje vsebuje določeno vrednost?

```
int[] stevilke = {10, 2, 44, 5, 23, 100, 13, 88, 16, 89};
int cilj = 89;

for (int i = 0; i < stevilke.length; i++) {
    if (stevilke[i] == cilj) {
        System.out.println(cilj + " je vsebovan v polju na indeksu " + i);
    }
}
```

- Linearno iskanje pregleduje elemente enega za drugim, dokler ne najde iskanega elementa ali doseže konca polja.
  - Če iskanega elementa ni v polju, je treba pregledati celotno polje.

Ali lahko iščemo boljše / hitreje?

- Kaj če je polje urejeno?

Polja - urejanje - sorting

Kaj je urejanje?

- Urejanje pomeni razporejanje elementov v določenem vrstnem redu – npr. naraščajoče ali padajoče po vrednosti.

Zakaj ga uporabljamo?

- Olajša iskanje in analizo podatkov
- Pogosto je predpogoj za druge algoritme

```
int[] stevila = {5, 2, 9, 1, 3}; // neurejeno polje

stevila = {1, 2, 3, 5, 9}; // urejeno narascajoce
```

Mehurčasto urejanje (Bubble Sort)

Ideja algoritma:

- Primerjaj sosednje elemente in jih **zamenjaj**, če so v napačnem vrstnem redu.
- Največji elementi (naraščajoči red) se pri tem “dvigajo” proti koncu polja — kot **mehurčki**.

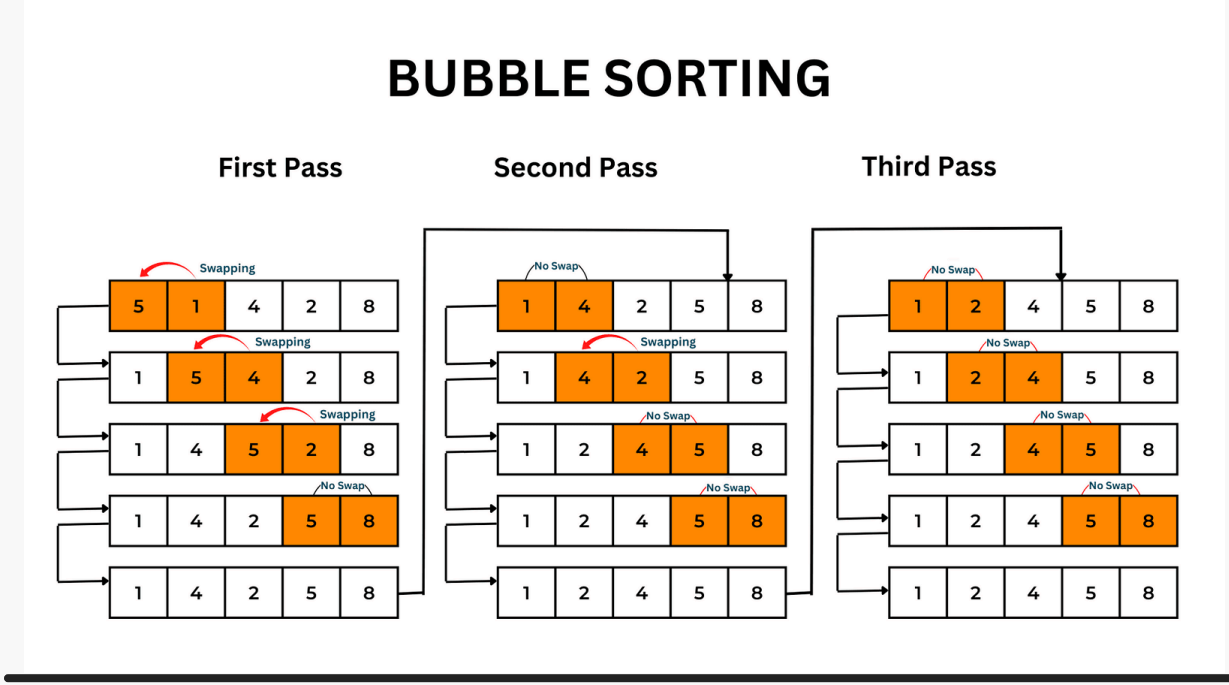
Postopek:

1. Večkrat prehodi polje.
2. V vsakem prehodu:
    - primerjaj sosednja elementa,
    - zamenjaj ju, če nista pravilno urejena.
3. Postopek ponavljaj, dokler ni polje urejeno.

Mehurčasto urejanje (Bubble Sort)

Postopek:

1. Večkrat prehodi polje.
2. V vsakem prehodu:
    - primerjaj sosednja elementa,
    - zamenjaj ju, če nista pravilno urejena.
3. Postopek ponavljaj, dokler ni polje urejeno.



Mehurčasto urejanje (Bubble Sort)

Koda:

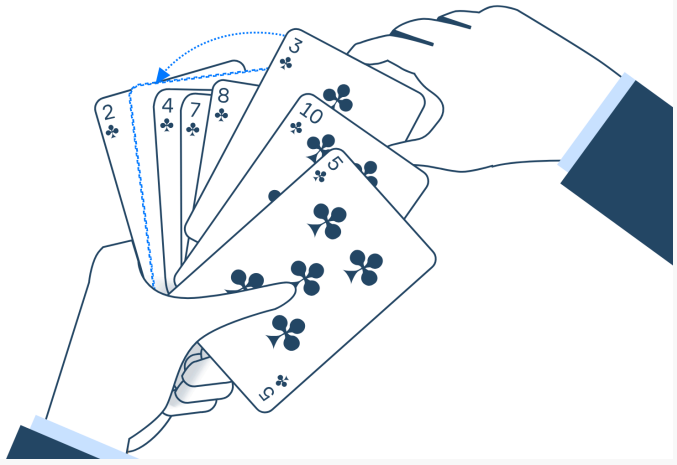
```
int[] a = {10, 2, 44, 5, 23, 100, 13, 88, 16, 89};

for (int i = 0; i < a.length - 1; i++) {   //outer
    for (int j = 1; j < a.length - i ; j++) {  //inner
        if( a[j-1] > a[j] ){  // compare
            int tmp = a[j-1];  //swap
            a[j-1] = a[j];
            a[j] = tmp;
        }
    }
}
```

Urejanje z vstavljanjem

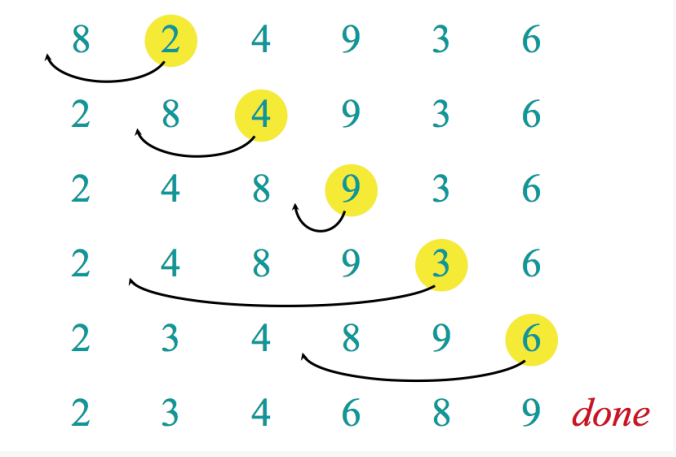
Urejanje z vstavljanjem -- Insertion sort

- Enostaven algoritem za urejanje, ki gradi urejeno polje element za elementom.
- Učinkovit za majhne ali skoraj urejene nabore podatkov.



Kako deluje:

1. Predpostavite, da je polje razdeljeno na dva dela:
   - urejeni del | neurejeni del
   - Predpostavite, da je prvi element v urejenem delu.
2. Vzemite prvi element iz **neurejenega** dela
   - Poišči njegov pravilen položaj v **urejenem** delu polja.
   - **Premakni** elemente enega za drugim, da narediš prostor.
   - **Vstavi** element v ustvarjen prostor.
3. **Ponovite** korak 2, dokler niso vsi elementi v urejenem delu polja.



Urejanje z vstavljanjem

Koda:

```
int[] a = {10, 2, 44, 5, 23, 100, 13, 88, 16, 89};

for (int i = 1; i < a.length; i++) {
    int key = a[i];         // take one element
    int j = i;
    while( j > 0 && a[j-1] > key){  // move elements until the
        a[j] = a[j-1];              // correct position is found
        j--;
    }
    a[j]=key;               // insert the element
}
```

Polja - urejanje

Urejanje podatkov je relevanten problem!

- Obstaja veliko algoritmov za urejanje

Nekateri algoritmi za urejanje podatkov:

- Enostavni za implementirat, vendar počasni:
  - Bubble sort
  - Selection sort
  - Insertion sort
- Hitri, vendar kompleksna implementacija:
  - Merge sort
  - Quick sort
  - Heap sort

Večdimenzionalna polja

Kako predstaviti matriko v računalniškem programu?

2D polja - **Polje polj**

$$\begin{bmatrix} 10 & 2 & 3 \\ 4 & 5 & 8 \\ 2 & 3 & 11 \end{bmatrix}$$

- Polje, ki hrani reference na druga polja

```
int[][] matrika = new int[3][3];  //inicializiraj matriko 3x3

int[] vrstica1 = {10, 2, 3};
int[] vrstica2 = {4, 5, 8};
int[] vrstica3 = {2, 3, 11};

matrika[0] = vrstica1;
matrika[1] = vrstica2;
matrika[2] = vrstica3;
```

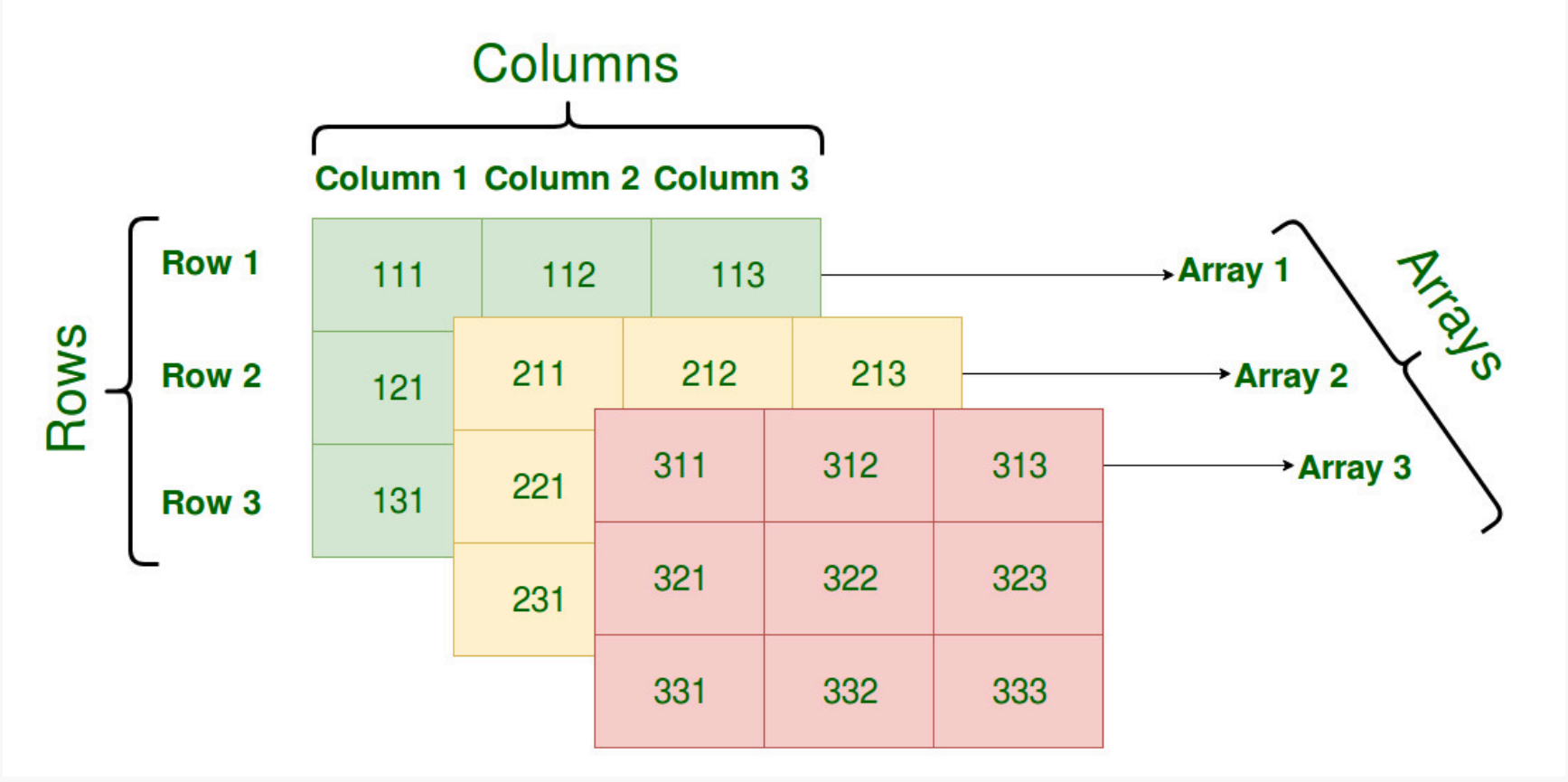
	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

- Dostop do vrednosti v matriki:

```
matrika[vrstica][stolpec]
```

Večdimenzionalna polja

Ali lahko gremo več kot 2D?



- RGB slike

Dinamična polja

Ali nas fiksna velikost polj **omejuje**?

Ali bi lahko imeli polje z **dinamično** velikostjo?

- Ki se širi ali krči, ko je to potrebno

V Javi imamo tak koncept, implementiran kot:

- `ArrayList`
- `Vector`