OSNOVNI GRADNIKI PROGRAMSKIH JEZIKOV 1. DEL

Programiranje 1 - RIN, BF, MR

Računalništvo 1 - MA, MF

Predavatelj: Niki Hrovatin

Tipi in spremenljivke

Podatkovni tip je informacija, ki je povezana z določenim podatkom in računalniku pove, kako ta podatek interpretirati.

- Npr. “int” je primitivni tip v C, C++, Javi in mnogih drugih jezikih. V Javi je to 32-bitno predznačeno celo število.

```
int x = 50;
int y = 30;
int z = x + y;
```

Spremenljivka je ime, uporabljeno v kodi, ki se nanaša na določeno instanco tipa.

- x, y, z so zgoraj spremenljivke.
- Vse so tipa int.

Tipi in spremenljivke - Java je strogo tipiziran jezik

Ključni koncept:

- Vsaki spremenljivki v Javi je potrebno določiti tip.
- Kompatibilnost tipov je **vedno uveljavljena**.

Primer:

```
int jabolka = 10;   // Spremenljivka shranjuje cela števila (integere).
jabolka = "pomaranča"; // Napaka! Neskladje tipov – ne morete dodeliti niza celemu številu.
```

Zakaj je to pomembno:

- Java prevajalnik zagotavlja **varnost tipov**: preprečuje napake pri pretvarjanju tipov med izvajanjem.
- Optimizacija: tipi se lahko ustrezno upravljajo.

Tipi in spremenljivke - Primitivni tipi

Primitivni tipi so vgrajeni v Jeziku Java

- Primitivni tipi so temeljni za hitre, **nizkonivojske operacije**, kjer je zmogljivost ključnega pomena.
- So gradniki za bolj **kompleksne tipe**, ki jih bomo kmalu obravnavali.

Tipi in spremenljivke - Primitivni tipi

Data Type	Size	Description	Values
byte	8 bits	whole numbers	-128 to 127
short	16 bits	whole numbers	-32,768 to 32,767
int	32 bits	whole numbers	-2,147,483,648 to 2,147,483,647
long	64 bits	whole numbers	-2^{63} to $2^{63}-1$
float	32 bits	floating point numbers*	1.4e-45 to 3.4e+38
double	64 bits	floating point numbers*	4.9e-324 to 1.7e+308
boolean	8 bits	logical value	0 (false) or 1 (true)
char	16 bits	characters	character/letter ASCII and unicode

IEEE 754 binary floating point representationSource

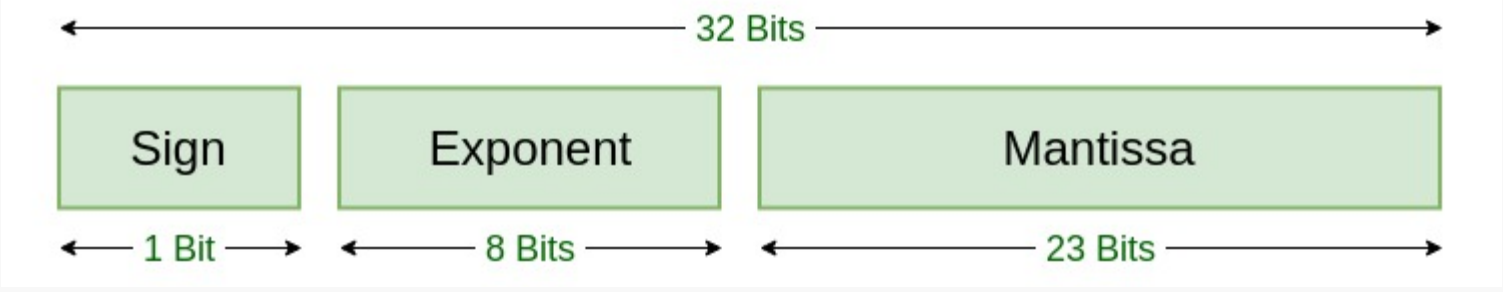
Tipi in spremenljivke - Dvojiški komplement in plavajoča vejica

Dvojiški komplement: Binarna predstavitev števila

- Pozitivno negativno -- poglej prvi bit
  - **Pozitivno:** pretvori v desetiško
  - **Negativno:**
    - Negiraj binarno predstavitev
    - Prištej 1
    - Pretvori v desetiško

IEEE 754: plavajoča vejica

- $mantisa^{exponent}$



Tipi in spremenljivke

Deklaracija spremenljivke

- Deklaracija: Določitev podatkovnega tipa in imena spremenljivke.
- V Javi lahko spremenljivko deklariramo, ne da bi ji dodelili začetno vrednost.

```
int stevilo;  // Deklaracija
```

Inicializacija in prirejanje vrednosti spremenljivki

- Inicializacija: Dodelitev začetne vrednosti spremenljivki.
- Prirejanje vrednosti: Dodelitev vrednosti spremenljivki, vključno s spreminjanjem njene obstoječe vrednosti.

```
number = 10;         // Prva prireditev (inicializacija)
number = 15;         // Ponovna prireditev

int secondNumber = 20; // Deklaracija in inicializacija
```

Tipi in spremenljivke - Pretvorba tipa

```
a = 3;
b = 'a';
c = a + b;
```

Obstajata dva primera:

- Izjava ni veljavna — **Napaka**
- Izjava je veljavna, sistem pa bo samodejno (implicitno) izvedel pretvorbo iz enega tipa v drugega — **Pretvorba tipa (type casting)**

Tipi in spremenljivke - Implicitna pretvorba tipov

Pretvorba podatkovnih tipov (Type conversion) spremeni vrednost iz enega podatkovnega tipa v drugega.

V Javi poznamo dve glavni vrsti pretvorb tipov:

- **Razširitev (Widening)** in **zožitev (Narrowing)**

Razširitev (Widening Conversion) – samodejna pretvorba

- Izvede se samodejno, brez eksplicitne pretvorbe (cast).
- Java omogoča naslednje poti pretvorbe:

  `byte` → `short` → `int` → `long` → `float` → `double`

  `char` → `int` → `long` → `float` → `double`

```
short stevilo1 = 20;
int stevilo2 = stevilo1;
long stevilo3 = stevilo2;

char character = 'r';
int code = character; // 114
```

Tipi in spremenljivke - Eksplicitna pretvorba tipov

Zožitev (Narrowing Conversion) – ročna pretvorba (explicit cast)

- Običajno zahteva eksplicitno pretvorbo: `(type) expression`.
- Lahko povzroči izgubo informacij ali spremembo vrednosti.
- Pogoste zožitvene pretvorbe:

  `double` → `float` → `long` → `int` → `short` → `byte`

```
double a = Math.PI;
float b = (float) a;
long c = (long) a;
int d = (int) a;
short e = (short) a;
byte f = (byte) a;

short x = 114;   // Pretvorba med char in short v obe smeri zahteva eksplicitni cast.
char y = (char) x;
short z = (short) y;
```

Tipi in spremenljivke - Poimenovalne konvencije

Spremenljivke v **camelCase**:

- Imena spremenljivk se začnejo z malo začetnico, in če ime sestavlja več besed, se prva črka vsake naslednje besede zapiše z veliko začetnico.

```
double totalAmount = 1000.42;
```

Pomenska in opisna imena

- Imena spremenljivk morajo biti dovolj opisna, da jasno kažejo svoj namen ali podatke, ki jih shranjujejo.

Slab primer:

```
int pipo = 22;
double popo = 3.3;
```

Dober primer:

```
int age = 22;
double price = 3.3;
```

Tipi in spremenljivke - Konstante

Konstanta je vrednost, ki je ni mogoče spremeniti po tem, ko ji je bila dodeljena.

- Konstante morajo biti inicializirane ob deklaraciji.

Za deklaracijo katere koli spremenljivke kot konstante uporabimo modifikatorja **static** in **final**, znana tudi kot **ne-dostopna** (non-access) modifikatorja.

```
class ConstantExample {

    static final double MAX_VISINA = 2.60; // inicializacija konstante

    public static void main(String[] args) {
        System.out.println(MAX_VISINA);
    }
}
```

Poimenovalna konvencija:

- Vse z velikimi črkami, besede pa so ločene z podčrtaji ("_")

Tipi in spremenljivke - `char` tip

Podatkovni tip **`char`** v Javi se uporablja za shranjevanje **enega znaka**.

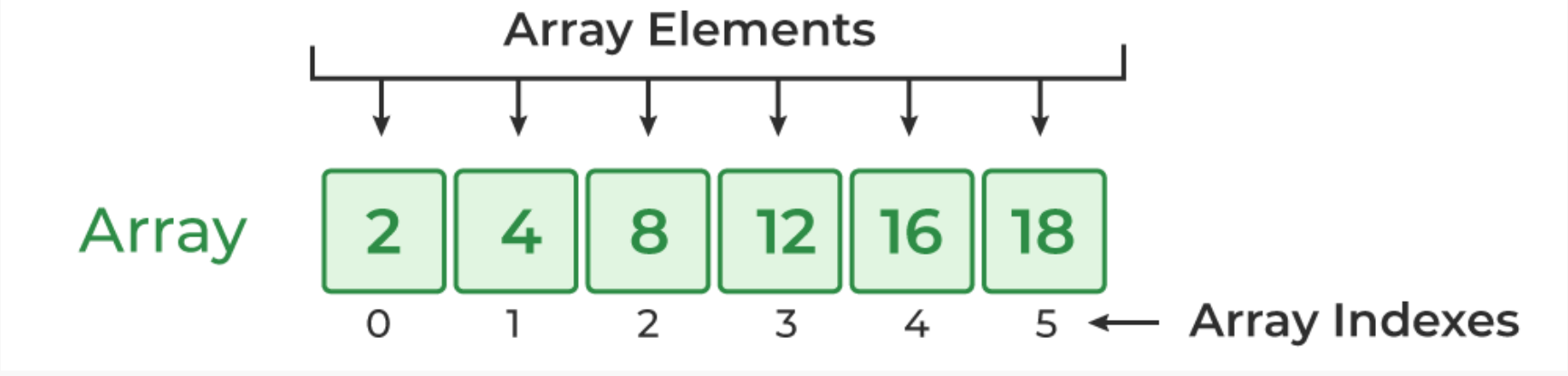
- `char` je 16-bitni nepredznačeni podatkovni tip, ki predstavlja **kodno enoto UTF-16**. Podpira znake **ASCII** in številne znake Unicode.
- Omogoča predstavitev širokega nabora simbolov, vključno s črkami, številkami in posebnimi znaki.
- Podpira z ASCII standard.

```
char crka = 'A';          // Shranjuje en znak 'A'
char stevilka = '1';      // Shranjuje en znak '1'
char simbol = '#';        // Shranjuje poseben znak '#'
char unicodeZnak = '\u263A'; // Shranjuje Unicode znak (☺)
char a = 'A' + 32; // Shranjuje znak 'a'
```

Polja

Polje (array) je **objekt**, ki lahko hrani **določen** število vrednosti **istega tipa**.

- **Dolžina polja** je določena ob njegovem ustvarjanju in se ne more spremeniti.
- Elementi polja so dostopni prek njihovih **indeksov**, ki se začnejo s 0.



Polja - Primer

```java
class ArrayDemo {
    public static void main(String[] args) {
        int[] mojePolje;  // Deklariramo polje celih števil
        mojePolje = new int[10];  // Dodelimo pomnilnik za 10 celih števil
        mojePolje[0] = 100;  // Inicializiramo prvi element
        mojePolje[1] = 200;  // Inicializiramo drugi element
        // ...
        mojePolje[9] = 1000;  // Inicializiramo zadnji element

        System.out.println("Element na indeksu 0: " + mojePolje[0]);
        System.out.println("Element na indeksu 1: " + mojePolje[1]);
        // ...
        System.out.println("Element na indeksu 9: " + mojePolje[9]);
    }
}
```

Operatorji

Operator v Javi je simbol, ki se uporablja za izvajanje operacij.

- Na primer: +, -, *, / itd.

```
int a = 20 * 30 / 2 - 3 + 10;
```

Operatorji

Seznam operatorjev po **prednostnem vrstnem redu**:

Operator type	Category	Symbols
Unary	postfix	*expr*++ *expr*--
Unary	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
Arithmetic	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
Relational	equality	== !=
Bitwise	bitwise	& ^ |
Logical	logical	&& ||
Ternary	ternary	'(a > b) ? a : b;'
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operatorji - Unarni

Unarni operatorji delujejo na en sam operand (spremenljivko)

- **++ increment** operator, poveča vrednost spremenljivke za **1**
- **-- decrement** operator, zmanjša vrednost spremenljivke za **1**
- **++** in **--** se uporabljata samo za **celoštevilske tipe**: `byte`, `short`, `int`, `long`.

Predpona in **pripona** (Prefix in Postfix)

- Predpona: **++i** in **--i** operator se aplicira pred uporabo vrednosti
- Pripona: **i++** in **i--** operator se aplicira po uporabi vrednosti

```
int i = 3;
int j = i--;
// j je 3, i je 2
```

Operatorji - Aritmetični in logični operatorji

Aritmetični operatorji:

- osnovni: `+ - * / %`
- prireditveni: `= += -= *= /= %=`

```
a = a + 1
// enakovredno
a += 1
```

Logični operatorji:

- osnovni: `&& || !`
- primerjalni: `< > <= >= == !=`

Pozor!!!

- za primerjavo dveh spremenljivk uporabite **==**, ne **=**, ki je prireditveni operator.

Operator %

Operator ostanka pri deljenju (ang. Reminder operator) %

- Vrne ostanek pri deljenju.
- a
- Implementacija se lahko razlikuje med programskimi jeziki.

Primer:

```
0 % 3 = 0
1 % 3 = 1
2 % 3 = 2
3 % 3 = 0 // cikel nazaj na 0
4 % 3 = 1
5 % 3 = 2
6 % 3 = 0 // cikel nazaj na 0
```

Uporabnost:

- Prepoznavanje lihih / sodih števil
- Ostanek deljenja
- Omejevanje števil na določen razpon
- Kroženje skozi omejene možnosti (krožno polje)
- Pretvarjanje merskih enot

Operatorji - Bitni

Bitni operatorji: manipulirajo posamezne bite

- osnovni: & | ^ ~
- premiki: >> << >>>
- prireditveni premiki: <<= >>= >>>=

```
int a = 5; // 0101 v binarni obliki
int rezultat = a >> 1; // Rezultat: 0010 (2 v decimalni obliki)
```

Ternarni operator v Javi ? :

- Vrstični pogojni stavek: (pogoj) ? vrednost_če_je_res : vrednost_če_je_napačno

```
int a = 10;
int c = 5;
int b;
b = (a > c) ? a : c;  // b je 10
```

Celoštevilsko deljenje

Kaj je celoštevilsko deljenje?

- Pri deljenju dveh celih števil je rezultat **celo število**, kar pomeni, da je vsak **decimalni del zavržen**.

```
int a = 22 / 7;  // rezultat 3

double a = 22 / 7;  // rezultat 3
```

Kako popraviti:

- eden od operandov mora biti **število s plavajočo vejico**

```
double a = 22 / 7.0;  // rezultat 3.142857142857143
```

Izraz

Izraz je kombinacija **vrednosti, spremenljivk in operatorjev**, ki ob izračunu proizvede vrednost določenega tipa.

Primeri:

```
int x = 5; // izraz, ki vrne 5

int y = x + 2; // x + 2 je izraz, ki se oceni na 7.

boolean enako = x == y;

boolean jeVeljavno = a > b;

String polnoIme = prvoIme + " " + priimek;
```

Sestavljeni izrazi

Sestavljen izraz je oblikovan z združevanjem **več izrazov** z uporabo operatorjev, funkcij ali drugih izrazov.

- Ovrednoteni so glede na pravila prednosti in asociativnosti v določenem programskem jeziku.

Primeri:

```
1*2*3

x + y / 100 // dvoumen izraz; ni priporočljivo

(x + y) / 100 // nedvoumen izraz

x + (y / 100) // nedvoumen izraz
```

Stavki

Stavki v programskem jeziku so primerljivi z stavki v naravnem jeziku:

- Stavek v programskem jeziku tvori celovito enoto, ki se izvede, in ga včasih imenujemo tudi ukaz.
- V **Javi** se vsak stavek zaključi s podpičjem (`;`).

Primeri:

```
aVal = 8933.234; // prireditveni stavek

aVal++; // inkrementacijski stavek

double bVal = 8933.234; // deklaracijski stavek
```

Bloki

Blok je skupina **stavkov**, ki so obdani z zavitimi oklepaji `{}`.

- Bloki se uporabljajo za organizacijo kode in določanje **obsega** (scope) spremenljivk in stavkov.

Primeri:

```
class BlockDemo {
    public static void main(String[] args) {
        boolean pogoj = true;
        if (pogoj) { // začetek 1. bloka
            System.out.println("Pogoj je resničen.");
        } // konec 1. bloka
        else { // začetek 2. bloka
            System.out.println("Pogoj ni resničen.");
        } // konec 2. bloka
    }
}
```

Kontrola toka programa

Zaporedno izvajanje ukazov:

- Stavki programa se običajno izvajajo **eden za drugim**, v **vrstnem redu**, kot se pojavijo.
- To se imenuje **zaporedni (ali linearni) tok izvajanja**.

```
a = 2;   // Shrani vrednost 2 v a
a += 3;  // Dodaj 3 k a
b = a + 4;  // Shrani vrednost a + 4 v b
```

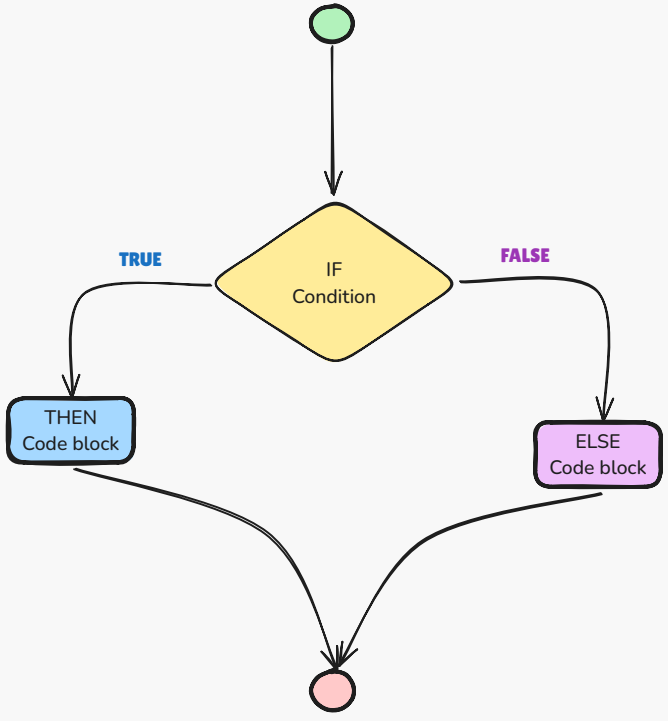
Kontrola toka programa

Spreminjanje toka izvajanja:

- Programski jeziki ponujajo konstrukte, ki omogočajo **prekinitev zaporednega toka**.
- Ti stavki omogočajo:
  - **Vejitve**: Pogojno izvajanje na podlagi logičnih pogojev.
  - **Skoki**: Neposredna sprememba točke izvajanja (npr. `return`, `break`).
  - **Zanke**: Ponavljajoče se izvajanje bloka kode.

Vejitve

Vejitve omogočajo izbiro med **dvema ali več potmi** izvajanja programa na podlagi pogoja.



Pogoj je lahko:

- **Logičen** (izid je true/false) → dve možni poti
- **Na podlagi vrednosti spremenljivke** → več možnih poti

Vejitve - `if - then [- else]`

If-Then-Else se uporablja za **pogojno izvajanje** kode.

- Programu omogoča izbiro med dvema blokoma kode na podlagi **pogoja**.

Struktura:

- **Če** je pogoj **resničen**, se izvede koda znotraj **if** bloka.
- **Sicer** se izvede koda znotraj **else** bloka, če je pogoj **ne-resničen**.

```
if (x > 0) {
    System.out.println("Pozitivno število"); // pogoj RESNIČEN
} else {
    System.out.println("Negativno ali ničelno število"); // pogoj NE-RESNIČEN
}
```

Vejitve - `switch`

Switch stavek omogoča **več pogojnih poti** na podlagi vrednosti ene spremenljivke.

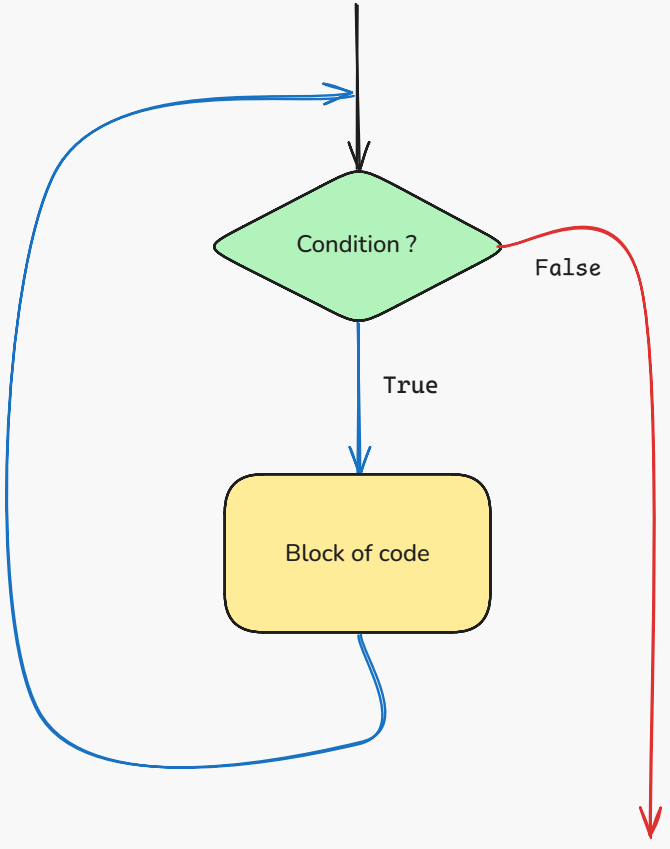
- Ponuja čistejšo alternativo uporabi večih `if-else` stavkov.

```
int dan = 3;

switch (dan) {
    case 1:  System.out.println("Ponedeljek"); break;
    case 2:  System.out.println("Torek"); break;
    case 3:  System.out.println("Sreda"); break;
    case 4:  System.out.println("Četrtek"); break;
    case 5:  System.out.println("Petek"); break;
    case 6:  System.out.println("Sobota"); break;
    case 7:  System.out.println("Nedelja"); break;
    default:  System.out.println("Neveljaven dan");
}
```

Zanke

Zanke, znane tudi kot **iterativni stavki**, se uporabljajo, ko moramo večkrat izvest določen blok kode.



- Izvajanje zank je vedno vezano na nek pogoj

Zanke - `while`

Zanka while je **vhodno nadzorovana zanka.**

- Dokler je pogoj izpolnjen ponavljaj ukaze v zanki
- Ukazi lahko se nikoli ne izvedejo!

Primer:

```
class WhileDemo {
    public static void main(String[] args){
        int stevec = 1;
        while (stevec <= 5) {
            System.out.println("Štetje: " + stevec);
            stevec++;
        }
    }
}
```

Zanke - do - while

Zanka do-while je **izhodno nadzorovana zanka.**

- Ponavljaj ukaze v zanki dokler je pogoj izpolnjen
- Ukazi se izvedejo vsaj enkrat!

Primer:

```
class DoWhileDemo {
    public static void main(String[] args){
        int stevec = 1;
        do {
            System.out.println("Štetje: " + stevec);
            stevec++;
        } while (stevec <= 5);
    }
}
```

Zanke - `for`

Zanka for je ena izmed najpogosteje uporabljenih zank, saj znotraj svoje strukture določa ključne parametre za nadzor iteracije.

Pogosto se uporablja za:

- Prehod skozi elemente polja
- Spreminjanje vrednosti spremenljivke v enakomernih korakih
- Izvajanje iteracij na preprost in strukturiran način

Primer:

```
for (inicializacija; pogoj; korak) {
    [stavki]
}
```

Zanke - `for`

Primer 1:

```
class ForDemo {
    public static void main(String[] args) {
        for (int i = 1; i < 11; i++) {
            System.out.println("Šteli smo do: " + i);
        }
    }
}
```

Primer 2:

```
class EnhancedForDemo {
    public static void main(String[] args) {
        int[] stevila = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for (int i : stevila) {
            System.out.println("Šteli smo do: " + i);
        }
    }
}
```

Zanke - ∞

Neskončne zanke:

- Lahko so napaka
  - vaš program se neskončno ∞ zatakne v zanki
- Lahko so funkcionalnost
  - potreba po čakanju na nekaj, kar spremeni stanje programa

While:

```
while (true) {
    // neskončna while zanka
}
```

For:

```
for ( ; ; ) {
    // neskončna for zanka
}
```

Integer Overflow

Kaj je Integer Overflow?

- Pojavi se, ko celo število preseže svojo največjo ali najmanjšo vrednost.
- Povzroči, da se vrednost "zavrti" na nasprotni konec dovoljenega razpona.

Example:

```
int maxValue = Integer.MAX_VALUE; // 2,147,483,647
int result = maxValue + 1;        // -2,147,483,648 (overflow)
```

Zakaj se to zgodi?

- Celo števila imajo fiksno velikost (npr. 32-bit za `int`), in ko izračun preseže obseg, rezultat preplavi.

Kako to preprečiti?

- Uporabite večje tipe podatkov (npr. `long`).
- Vpeljite preverjanja pred izvajanjem kritičnih izračunov.

Overflow pri plavajoči vejici

Overflow v IEEE 754

- Tako kot cela števila imata tudi **float** in **double** omejitve glede na vrednosti, ki jih lahko shranita.
- Ko izračuni presežejo te omejitve, vrednost postane **neskončnost** (+∞ ali -∞).
- Programu omogoča nadaljevanje brez sprožanja napak.

```
double largeValue = Double.MAX_VALUE;  // 1.7976931348623157E308
double result = largeValue * 2;        // Infinity (overflow)
```

IEEE 754 in Underflow Rezultat postane **0.0**, če je vrednost manjša od najmanjše predstavljive številke.

```
int i = 1074;
double a = Math.pow(2, -i);   // a = 4.9E-324
i++;
a = Math.pow(2, -i);          // a = 0.0
```

Skoki

Skoki omogočajo "skok" na poljubno mesto v programu.

- Spreminjajo običajnega poteka programa.
- Skok je mogoč le na določeno *označeno* mesto s t.i. značko (**label**).

Če skok sledi nekemu (logičnemu) pogoju, mu pravimo pogojni skok.

- Večina skokov v programskih jezikih je pogojnih.

Skoki - `break`

`break` predčasno prekine izvajanje zanke ali switch stavka.

- `break` stavek takoj izstopi iz najnotranješe zanke ali switch bloka.
- Program nato nadaljuje z naslednjim stavkom po zanki ali switch bloku.

```
class BreakDemo {
    public static void main(String[] args) {
        int[] tabela = {32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127};
        int cilj = 12;
        boolean najdeno = false;

        for (int indeks = 0; indeks < tabela.length; indeks++) {
            if (tabela[indeks] == cilj) {
                najdeno = true;
                break;
            }
            System.out.println("Preverjanje vrednosti: " + tabela[indeks]);
        }
        if (najdeno) {
            System.out.println("Najden " + cilj + " v polju");
        } else {
            System.out.println("Cilja " + cilj + "ni v polju");
        }
    }
}
```

Skoki - `continue`

`continue` "preskoči" vse ukaze do konca zanke v trenutni iteraciji

```
class ContinueDemo {
    public static void main(String[] args) {
        int[] tabela = {32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127};
        int cilj = 12;
        boolean najdeno = false;

        for (int indeks = 0; indeks < tabela.length; indeks++) {
            if (tabela[indeks] == cilj) {
                najdeno = true;
                continue;
            }
            System.out.println("Preverjanje vrednosti: " + tabela[indeks]);
        }
        if (najdeno) {
            System.out.println("Najden " + cilj + " v  polju");
        } else {
            System.out.println("Cilja " + cilj + " ni v polju");
        }
    }
}
```

Skoki - `goto` in `labels`

`goto` v kombinaciji z oznakami (**labels**)

- omogoča skok na določeno točko v kodi
- obide normalen potek izvajanja programa.

Stara slaba praksa:
https://en.wikipedia.org/wiki/Goto#Criticism

- `goto` v Javi ni podprt
- `labels` se lahko uporabijo z `break` in `continue` - izogibajte se

Primer:

```
start:
    Initialize counter = 0

loop_start:
    If counter equals 5, go to end
    Print "Counter is: " + counter
    Increment counter by 1
    Go to loop_start

end:
    Print "Loop ended"
```

Skoki - `return`

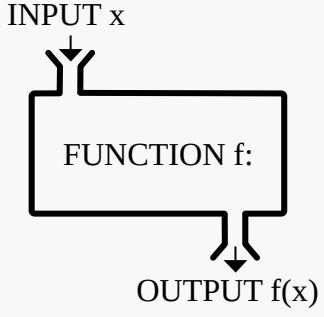
`return` stavek povzroči izstop programa iz trenutne funkcije/metode in nadaljuje izvajanje tam, kjer je bila funkcija/metoda poklicana.

- `return` stavek lahko vrne vrednost ali pa ne:
  - `return vrednost;`
  - `return;`

Funkcije

Matematika:

- **Funkcija** iz množice X v množico Y vsakemu elementu množice X priredi natanko en element množice Y.



Programiranje:

- funkcija (ali postopek, metoda, rutina ali subrutina)
- je klicna enota programske logike, ki ima dobro definiran vmesnik in vedenje ter jo je mogoče večkrat poklicati
- funkcije običajno "sprejmejo" podatke, jih obdelajo in "vrnejo" rezultat.

Funkcije - zakaj?

Zakaj strukturirati programe v funkcije?

- Pisanje velikih programov postane lažje, ko jih razdelimo na manjše, obvladljive enote.
- Vzdrževanje (in razhroščevanje) programov je lažje
- Lažje sodelovanje pri pisanju programov

Splošna struktura funkcije:

```
[tip_rezultata] <ime_funkcije>(parametri) {           // Podpis funkcije
    <deklaracije spremenljivk>;    <stavki>;          // Telo funkcije
    [return(<vrednost>);]                             // Vračanje vrednosti
```

Klic funkcije:

```
rezultat = <ime_funkcije>(parametri);
```

Funkcije v Javi == Metode

Java funkcije se imenujejo **metode!**

- ker so vedno povezane z **objektom** ali **razredom**.

Poimenovalna konvencija:

- Ime mora jasno odražati funkcionalnost
- `camelCase`
  - prva beseda se začne z malo začetnico, naslednje besede so zapisane z veliko začetnico

Sintaksa metode:

```
[modifikatorji] <tip_rezultata> <ime_metode>(parametri) [throws] {
    <deklaracije_spremenljivk>;
    <stavki>;
    [return <vrednost>;]
}
```

Metoda primer

```java
public class Main {
    public static void main(String[] args) {
        int ploscina = izracunajPloscino(4, 5); // Klic metode
        System.out.println("Ploscina je: " + ploscina);
    }

    public static int izracunajPloscino(int sirina, int visina) {
        int ploscina; // Lokalna spremenljivka
        ploscina = sirina * visina;
        return ploscina; // Vrni izračunano površino
    }
}
```

Ali je program le ena velika funkcija?

Ali je program funkcija?

- Kaj natančno je funkcija?
  - Blok kode (popolna logična enota)
  - Lahko sprejme parametre in vrne vrednosti
  - Lahko jo pokličemo večkrat
- Kaj pa program?
  - Ali ni (glavni) program v C/C++, Javi itd. zgolj funkcija `main()`?

```
public static void main(String[] args){ . . . }
```