Programming

Basic building blocks

1st part

2. lecture

2012/13

Overview

- An example of a simple program
- Variables (spremenljivke) (later objects)
- Basic types (osnovni tipi) (later classes)
- What is an array? (polje)
- Operators → expressions (arithmetic, logical, ...)
- Statements and blocks
- Control flow of the program
- Functions and programs

A simple program

```
/**
 * A simple program that produces the output:
 * "Hello :-)".
 */

class Hello {
  public static void main(String[] args) {
    System.out.println("Hello :-)"); //Outputs the string
  string.
  }
}
```

```
A simple program that outputs
class Hello {
  public static void main(String[] args) {
    System.out.println("Hello :-)"); //Outputs the string
  }
}
```

/** ... */ ... main ... (Hello.java) ...

Variables

- A variable is an object that has the property that:
  - **maintains** the **assigned** value**:**

    variable = value

    a = 7

  - any time you can **look** at its value:

    secondVariable = **variable**

    b = **a**

- A constant is a specific type of variable, which cannot change the assigned value

Variables – naming rules

- Variables should have meaningful names:
  - □ *Name_of_a_variable*
  - □ *length*, *maxValue*, *counter*
  - □ …
  - □ Short names for indexes: **`a,b,c,i,j,k,`** ...
- Not all names are legal or good:
  - □ Naming conventions, reserved words of a PL

Variables – (primitive) types

- **type variavle** =
  type of data the variable stores
- Primitive types:
  - □ integers (int) – `1,-4,0, ...`
  - □ floating point numbers (float) – `0,1E+45 (0,1*10`^{45}`); 0,456E-1 (0,456*10`$^{-1}$`); ...`
  - □ bool variables (boolean) – `true, false`
  - □ characters (char) – `‘a’, ‘Z’, ‘3’, ‘@’, ...`
  - □ strings (string) – `“ime”, “EMSO”,...`
- Compound types and classes

Variable types in JAVA

- Java distinguishes 4 basic variable types:
  - □ Integers: *byte*(8), *short*(16), *int*(32), *long*(64)
  - □ Floating point numbers: *float*(32), *double*(64)
  - □ Bool variables (1)
  - □ Characters (unicode 16)
- With 8 subtypes
- Everthing else is a ***class***

Variables – declaration, definition

- Variable declaration has a twofold purpose:
  1. give **name** and **type** to the variables,
  2. tell the compiler how much memory should be allocated for the needs of a particular variable.
- To be more precise, the term declaration stands for the first indent, the 2 indent is called definition of the variable.
- The declaration is implicit in some PL (eg C, FORTRAN, PERL, Python, PHP, ...)
- In the JAVA language, the declaration in explicit

Why do we use types? (1)

- Each variable must be tagged to the set it belongs in the Java programming language,
  - What type (explicit declaration)
  - later: in which class it belongs to
- Why?

Why do we use types?(2)

- What do the following assignments mean:

```
char b;
b = ‘T’;            // OK
a = b;              // maybe, if char a;
```

- Types are used, so we can check if our commands make sense,
  - later we will learn about classes, which one of the roles is exactly the same

Why do we use types?(3)

- What about programming languages, where the type declaration is implicit?
- For these languages, the variables are still in the same set (have a particular type), except that the type is assigned latter implicitly – at the first use:

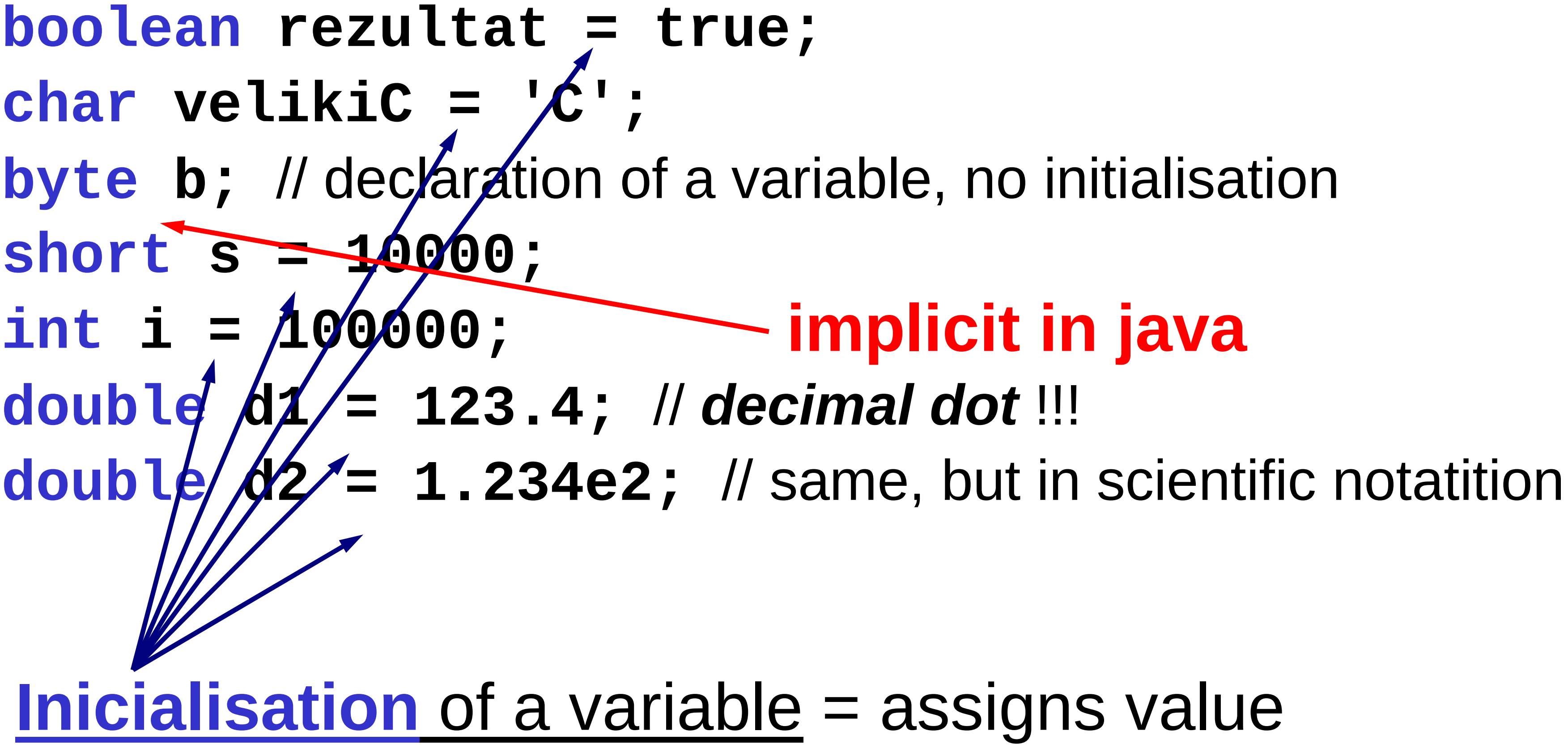
```
a = 3;          // a is integer
b = '4';        // b is char
c = a + b;      // hm, what is c now?
```

Why do we use types?(4)

```
a = 3;
b = '4';
c = a + b;
```

- We have (at least) the following options:
  - □ expression is not allowed - error: this is in JAVA and in most programming languages,
  - □ expression is not an error and the system tries to automatically (implicitly) convert one variable type to another type of variable - type casting,
- b is converted to an integer, the expression is summed and c is then re-integerised
- problem occurs when we have b = 'z'
- Is there any possible sollution?
- Explicit VS. implicit type casting

Variables – examples, **initialisation**



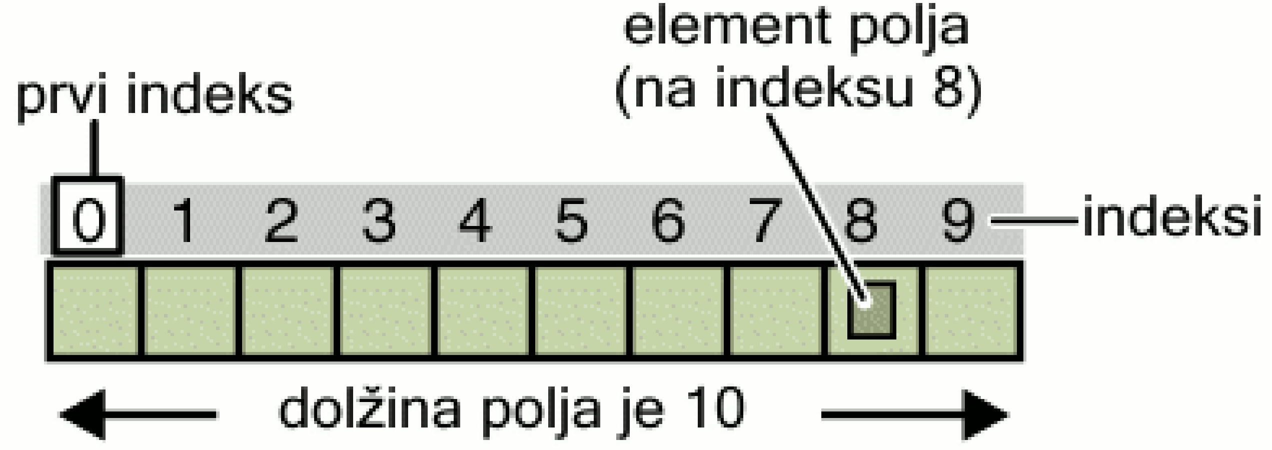
```
boolean rezultat = true;
char velikiC = 'C';
byte b;  // declaration of a variable, no initialisation
short s = 10000;
int i = 100000;
double d1 = 123.4;  // decimal dot !!!
double d2 = 1.234e2;  // same, but in scientific notatition
```

implicit in java

<u>Inicialisation of a variable</u> = assigns value

Array (of *objects*)

- ***Array*** is an object (predmet), that can carry a fixed number of values of the same type
- Array length is assigned at the creation time
- Elements are accessed using indexes



Array – deklaration, definition, initialisation

- Example:

```
class PoljeDemo {
  public static void main(String[] args) {
    int[] toPolje; // declaration of an integer array.
    toPolje = new int[10]; // allocation of the memory for 10 integers
    toPolje[0] = 100; // initialisation of the 1. element
    toPolje[1] = 200; // initialisation of the 2. element
                    ...
    toPolje[9] = 1000; // initialisation of the last element
    System.out.println("Element at index 0: " + toPolje[0]);
    System.out.println(" Element at index 1: " + toPolje[1]);
                    ...
    System.out.println(" Element at index 9: " + toPolje[9]);
  }
}
```

Operators

- We have declared and intialised the variables …
- Operator’s role = execution of ***operations*** on variables (operands) ➔ ***expressions***
- Operators are grouped by:
  - priority
  - Number of operands
  - type
  - …

Operators in JAVI – by priority

- postfix `izr++ izr--`
- unary `++izr --izr +izr -izr ~ !`
- multiplicative `* / %`
- additive `+ -`
- moving `<< >> >>>`
- relational `< > <= >= instanceof`
- equlity `== !=`
- bit AND `&`
- bit XOR `^`
- bit OR `|`
- logical AND `&&`
- logical OR `||`
- ternary `? :`
- assignment `= += -= *= /= %= &= ^= |= <<= >>= >>>=`

določanje prednosti izvajanja operatorjev?

Arithmetic and logical operators

- Arithmetic operators:
  - □ basic: `+  -  *  /  %`
  - □ asignment: `=  +=  -=  *=  /=  %=`
    (`a = a+1  ≡  a += 1`)
- Logical operators:
  - □ basic: `&&  ||  !`
  - □ relational, compare: `<  >  <=  >=  ==  !=`

Pozor !!!

Razlika med ***prireditvijo*** in ***primerjavo***

Bitwise operators and “**? :**”

- Used for bit manipulation
  - basic: **& | ^ ~**
  - moving: **<< >> >>>**
  - asignment: <<= >>= >>>=

- Ternanry operator “**? :**”
  (**b = (a>c) ? a : c**)

Special operators **++** and **- -**

- Operator **++** is called ***increment***, it increments the value of an integer variable by 1
- Operator **- -** is called ***decrement***, it decrements the value of an integer variable by 1

- Operators can be used in two forms:
  - □ prefix: **`++i`** **`--j`**
  - □ postfix: **`i--`** **`j++`**

Example ++

```
class PrePostDemo {
  public static void main(String[] args) {
    int i = 3;
    i++;
    System.out.println(i); // "4"
    ++i;
    System.out.println(i); // "5"
    System.out.println(++i); // "6"
    System.out.println(i++); // "6"
    System.out.println(i); // "7"
  }
}
```

Why 2x 6?

What is an expression?

- An ***expression***:
  1. consists of variables, operators and function/method invocations (calls) obeying the syntax of the programming language;
  2. Can always be assigned a value (of a ceartain type).

1. What type is the value of an expression?

Examples in JAVA

```
int stevec = 0;

tabela[0] = 100;
System.out.println("Element 1 na 0-tem mestu: " + tabela[0]);

int rezultat = 1 + 2; // rezultat je sedaj 3

if(vred1 == vred2)
  System.out.println("vred1 = vred2");
```

Composed expressions

- Expressions can be composed, assembled (nested)
  - □ Priority rules for the operators (and parentheses) are used when determining the value of a composed expression
- Examples:
  - □ **`1*2*3`**
  - □ **`x+y/100`** // ambiguous example – not recommended
  - □ **`(x+y)/100`** // **non-**ambiguous example
  - □ **`x+(y/100)`** // **non-**ambiguous example

Statements

- Statements of a PL can be compared to sentences in a natural language
  - □ ***Statement*** in PL is a closed unit, we could say an ***order,***
  - □ A sentence in JAVA finishes with a semi-colon (`;`)
- Examoples in JAVA:

`aVred = 8933.234;` // assignment statement

`aVred++;` // increment statement

`double aValue = 8933.234;` // declaration statement

Blocks

- ***Blocks*** are sets of 0 or more statements
- Example:

```
class BlokDemo {
  public static void main(String[] args) {
    boolean pogoj = true;
    if (pogoj) { // start of 1. block
      System.out.println("Pogoj je resničen.");
    } // end of 1. block
    else { // start of 2. block
      System.out.println("Pogoj ni resničen.");
    } // end of 2. block
  }
}
```

Flow control (1)

- Commands/statements of a program carried out one after the other (sequential or serial)
- Example:

1.	**`a = 2;`**	*save value **2** in **a***
2.	***`a += 3;`***	*add **3** to **a***
3.	***`b = a+4;`***	*save value of **a** increased by **4** into **b***
4.	**…**	**…**

Flow control (2)

- Programming languages contain commands that allow 'demolition' of sequenced implementation of other commands
- These commands affect the flow of the program to allow:
  - □ Branching
  - □ loops
  - □ jumps

Branching

- Allows the choice between two or more “paths" of the program depending on a condition
- The condition can be:
  - □ logical (**`true/false`**) ➔ 2 possible “paths”
  - □ bound to the value of a variable ➔ multiple “paths”

Branching– “if-then[-els

- If the condition is true, then…[else …]
- Example (*breaking with a bike*):

```
void break() {
  if (isMoving) {
    currentSpeed--; }
  else {
    System.err.println(“The bike is parked!");
  }
}
```

Branching– “switch”

- Allows more than only 2 “paths”:
- Example:

```
class SwitchDemo {
  public static void main(String[] args) {
    int mesec = 8;
    switch (mesec) {
      case 1: System.out.println("Januar"); break;
      case 2: System.out.println("Februar"); break;
                              ...
      case 12: System.out.println("December"); break;
      default: System.out.println("Napaka."); break;
    }
  }
}
```

Loops

- Enable the repetition of the same task (eg do something to all objects)
- The usage of loops is always linked to a condition - similar to the branching

Loops– “while”

- Repeat commands in the loop as long as the condition is true repeat (it can be no repeat at all!)
- Example:

```
class WhileDemo {
  public static void main(String[] args){
    int stej = 1;
    while (stej < 11) {
      System.out.println(“Current number: " + stej);
      stej++;
    }
  }
}
```

Loops– “do ... while”

- Repeat commands in the loop as long as the condition is true repeat (at least one repeat!)
- Example:

```
class DoWhileDemo {
  public static void main(String[] args){
    int stej = 1;
    do {
      System.out.println(“Current number: " + stej);
      stej++;
    } while (stej <= 11);
  }
}
```

Loops– “for”

- for loop is by far the most widely used loop:
  - Visit elements of an array
  - Change the value of a variable in equal steps
  - easily perform iterations
- The general form:

```
for (initialisation; condition; step) {
  [statements]
}
```

Loops– “for” – examples

- Example 1:

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

- Example 2:

```
class RazsirjenForDemo {
  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(" Current number is: " + i);
    }
  }
}
```

Endless loops

- Why would we use them?
- Example 1:

```
while (true) {
   //endless while loop
}
```

- Example 2:

```
for ( ; ; ) {
   // endless for loop
}
```

Jumps

- **Jumping - as the name implies - enable "jump" anywhere in the program**
  - **this place can be identified by a so-called label**
- **A jump that follows a (logical) condition is called a conditional jump**
- **vast majority of jumps in PL is conditional**

Jumps – “**`break`**”

- There are two types of break statement: unmarked and marked
- It serves as an early exit from **for**, **while** or **do** ... **while** loops
  - □ out of the 'current' loop - unlabeled break
  - □ Out of any nested loop - marked break
- <u>immediately</u> "jump" out of the loop

Jumps – “**break**” – example (1)

- Unmarked **break** (search of an element in an array):

```
class BreakDemo {
 public static void
        main(String[] args) {
  int[] polje = {32,87,3,589,12,
         1076,2000,8,622,127};
  int iscem = 12;
  int i;
  boolean nasel = false;
  for (i=0; i<polje.length; i++) {
   if (polje[i] == iscem) {
    nasel = true;
    break;
   }
  }
  if (nasel) {
   System.out.println("Našel " +
     iscem + " na indeksu " + i);
  } else {
   System.out.println(iscem +
     " nisem našel v polju");
  }
 }
}
```

Jumps – “**break**” – example (2)

- Marked **break** (search of an element in a 2D array):

```
class BreakDemo2 {
  public static void
                main(String[] args) {
    int[][] polje2D = { {32,87,3,589},
                        {12,1076,2000,8},
                        {622,127,77,955} };
    int iscem = 12;
    int i;
    int j = 0;
    boolean nasel = false;
isci:
    for (i=0; i<polje2D.length; i++) {
      for (j=0; j<polje2D[i].length; j++) {
        if (polje2D[i][j] == iscem) {
          nasel = true;
          break isci;
        }
      }
    }
    if (nasel) {
      System.out.println("Našel " + iscem +
               " na mestu " + i + ", " + j);
    } else {
      System.out.println(iscem +
               " ni v polju");
    }
  }
}
```

Jumps – “**`continue`**”

- There are two types **`continue`** statement unmarked and marked
- Serves as an early exit from **`for, while`** or **`do`** ... **`while`** loops
  - □ unmarked and marked version are similar to those related version of **break** statement
  - □ **`continue`** "skip" all instructions until the end of the loop (for the current iteration of the loop)

Jumps – “**continue**” – example (1)

- Unmarkes **continue** (occurrences of character in a string):

```
class ContinueDemo {
  public static void main(String[] args) {
    String isciMe = "peter pipec je pobral polno pest pisanih peres";
    int max = isciMe.length();
    int stPjev = 0;
    for (int i = 0; i < max; i++) {
      // zanimajo nas le p-ji
      if (isciMe.charAt(i) != 'p')
        continue;
      // naletali na p
      stPjev++;
    }
    System.out.println("Našel " + stPjev + " p-jev v nizu.");
  }
}
```

Jumps – “continue” – example (2)

- Marked **continue** (Search substring in a string):

```
class ContinueDemo2 {

  public static void main(String[] args) {

    String isciMe = "Najdi podniz v meni.";
    String podniz = "pod";
    boolean nasel = false;
    int max = isciMe.length() -
                        podniz.length();
```

```
test:
    for (int i = 0; i <= max; i++) {
      int n = podniz.length();
      int j = i;
      int k = 0;
      while (n-- != 0) {
        if (isciMe.charAt(j++) !=
                    podniz.charAt(k++)) {
          continue test;
        }
      }
      nasel = true;
      break test;
    }
    System.out.println(nasel ? "našel" :
                        "nisem našel");
  }
}
```

Jumps – “**goto**”

- the **goto** command "immediately jumps" to a designated place:
  - □ sometimes called unconditional jump,
  - □ it is not present in JAVA language.
- Example:

```
void prikazi(int matrika[3][3]) {
  int i,j;
  for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++) {
      if ( (matrika[i][j] < 1) || (matrika[i][j] > 6) )
        goto izven_mej;
      printf("matrika[%d][%d] = %d\n",i,j,matrika[i][j]);
    }
  return;
  izven_mej:
    printf("število mora biti med 1 in 6\n");
}
```

(Non!)usage of jumps

- Using jumps in programs brings with it a number of potential "dangers“:
  - □ More about it:

    http://www.roseindia.net/javatutorials/java_break_to_label_tatement.shtml

    http://en.wikipedia.org/wiki/GOTO
- Using the “jump" commands is considered a poor programmer practise and is not recommended
- instead of “jump" commands one can (almost) always use a branch or loop command

Jumps – “`return`”

- The **`return`** command causes output from the current function/method;
  - □ program continues to undergo, from where the function/method was called
- The **`return`** command can ***return value*** or ***not***

```
return ++stej;          return;
```

Functions (1)

- What is a function?
  - □ Definition of a function in mathematics: (http://en.wikipedia.org/wiki/Function_(mathematics))
  - □ What about computer science?

Definicija (1):

Funkcije (imenovane tudi *podprogrami* ali *procedure*) so kosi kode – deli programov, ki **jih lahko kličemo** iz drugih delov programa. V splošnem funkcije izboljšajo prostorsko učinkovitost kode in naredijo računalniške programe lažje za vzdrževanje.

Functions (also known as *subroutines* or *procedures*) are chunks of code – parts of programs – which **can be called** from another part of the program. Generally, functions greatly enhance the space-efficiency and maintainability of computer programs.

Vir: http://www.truetype-typography.com/ttglossf.htm

Definicija (2):

Funkcije so bloki kode, ki jo lahko ponovno uporabimo na več mestih v programu. Funkcije lahko sprejemajo **parametre** in **vračajo vrednosti**.

Functions are blocks of reusable code that can be passed **parameters** and can **return a value**.

Vir: http://www.123flashchat.com/flash/12_understanding9.html

Functions (2)

- *What are* functions in CS?
  1. Blocks of code / individual parts of the program;
  2. Accept parameters and return values;
  3. They can be called from other parts of the program.
- *The reasons for using the* functions:
  - □ writing large programs becomes easier (if they are disected into smaller units),
  - □ maintenance (and debuging) programs is easier,
  - □ facilitate cooperation when writing programs.

Vir: http://www.cs.utah.edu/~hamlet/release/lessons/fortran08/fortran08/node4.shtml

Function example

- General syntax

```
[tip_rezultata] <ime_funkcije>(parametri) {
  <deklaracija spremenljivk>;
  <stavki>;
  [return(<vrednost>);]
}
```

Podpis funkcije

Telo funkcije

Vračanje (vrednosti)

- Fuction in **C**:

```
double power(double val, int pow) {
  double ret_val = 1.0;
  int i;
  for(i = 0; i < pow; i++)
    ret_val *= val;
  return(ret_val);
}
```

Klic funkcije

```
result = power(val, pow);
```

Functions in JAVI = *methods*

- Syntax

```
[prilastki] <tip_rezultata> <ime_metode>(parametri) [throws] {
  <deklaracija spremenljivk>;
  <stavki>;
  [return <vrednost>;]
}
```

- Example:

```
public static int izracunajPloscino(int sirina, int visina) {
  int ploscina; // To je lokalna spremenljivka
  ploscina = sirina * visina;
  return ploscina;
}
```

```
plscn = izracunajPloscino(4,5);
```

What is a program?

- Do you still remember the introductory lecture?

- (RAČUNAL.) PROGRAM
  - (Računalniški) program je zbirka ukazov, ki opisujejo neko nalogo ali množico nalog, katere naj se izvajajo na določenem računalniku.

- COMPUTER PROGRAM
  - A computer program is a collection of instructions that describes a task, or set of tasks, to be carried out by a computer.

Vir: http://en.wikipedia.org/wiki/Computer_program

(program == function) = 1 ?

- What is a function?
  - □ block (= independent part) of programming code; ✓
  - □ it accepts (parameters) and returns values;
  - □ We invoke it. ✓
- What is a program? ✓

- What is a function?
  - □ Is the main program in (C/C++, JAVA, …) only the **`main()`** function?

Summary

References

- http://en.wikipedia.org/wiki/Computer_program
- http://en.wikipedia.org/wiki/Function_(mathematics)
- http://en.wikipedia.org/wiki/GOTO
- http://java.sun.com/docs/books/jls/
- http://java.sun.com/docs/books/tutorial/java/nutsandbolts/
- http://www.123flashchat.com/flash/12_understanding9.html
- http://www.cs.utah.edu/~hamlet/release/lessons/fortran08/fortran08/node4.shtml
- http://www.roseindia.net/javatutorials/
- http://www.truetype-typography.com/ttglossf.htm

Homewroks