Functions

CCS1115
Programming
Methodology and Design

Dr Ioanna Stamatopoulou

Functions

- Package (Library) functions/methods
- Developer-defined functions
- Function definition
- Function signature
- Function calls

Introduction to Functions

- A function is an independently defined block of code that performs a particular operation/job
- After a function is defined, I can **call** it whenever I need this operation/job to be executed
- We define and use functions so that:
  - Our solutions are well-designed
  - We may reuse our code
  - It is easier to maintain our code

Function Definition

- To define a function I need to define:
  1. The exact job of the function
     - Good practice: Include this as a comment in your.java file
  2. If it is **public** or private, **static** or not
     - For the time being all our functions will be: **public static**
  3. Its name
     - Should use a verb and be descriptive of the job of the function
  4. Its **parameters** inside parentheses
     - The parameters are the pieces of information that should be provided to the function so that it is able to do its job
     - (Parameters are variables)
  5. The **type of the returned result** or **void** if the function does not return a result
  6. The **body** of the function
     - The commands, within the curly brackets, that need to be executed for the function to do its job, to achieve its goal

Function Definition: Example

1. **What is the job of the function?**
2. **public** or private, **static** or not
3. **Name**
4. **Parameters** (the function needs an integer number in order to calculate its cube)
5. **Type of returned result** or **void** (in this case an int because the cube of an int is an int)
6. **Body within {}**

```
1  /* cube is a function that calculates
      the cube of an integer number
   */
   2          5     3      4
   public static int cube (int number)
   {
6          return number*number*number;
   }
```


Function Signature

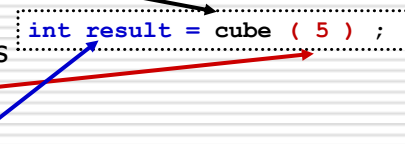
- The function without its body

```
public static int cube (int number) // function signature
```

- Informs the compiler of its existence
- Provides us with all the information that is necessary for us to call it

Function Call

- To call a function:
  - I use its **name**
  - I provide a value (**argument**) for each one of the functions parameters inside the parentheses
  - If the function returns a value, I (usually) **store the returned value in a variable**



Package (Library) Functions/Methods

- Java provides us with an extensive collection of classes and, in consequence, functions/methods
- The provided classes are organised in **packages**
- java.lang is the default package
  - Contains, among others, the class **`System`**, which contains the function **`println`**

```
System.out.println("some text");
```

Package (Library) Functions/Methods (cont'd)

- Functions for mathematical operations are defined in the **`Math`** class, also inside the java.lang package
- Class **`JOptionPane`** is inside the javax.swing package and that's why we have to import the latter in order to use it

```
import javax.swing.JOptionPane;
public class CallLibraryMethods {
  public static void main(String[] args) {
    String s = JOptionPane.showInputDialog("Enter a real number: ");
    double num = Double.parseDouble(s);

    double power = Math.pow(num, 3) ;
    System.out.println("3rd power of " + num + " is " + power);

    double absValue = Math.abs(num) ;
    System.out.println("Absolute value of " + num + " is " + absValue);

    double squareRoot = Math.sqrt(num) ;
    System.out.println("Square root of " + num + " is " + squareRoot);
  }
}
```

`pow` is a function that calculates and returns the result of raising a number to a power (2 parameters)

`abs` is a function that calculates and returns the absolute value of a number

`sqrt` is a function that calculates and returns the square root of a number

Developer-defined functions

- By defining our own functions we break a problem into smaller parts (sub-tasks)
  - Each function is responsible for performing a sub-task
- Whenever necessary we call a function for the completion of the corresponding sub-task

```
public class MyFirstWithFunctions {
   public static void main (String[] args) {
        int n = 3;
        String out = "The cube of " + n + " is " + cube(n);
        System.out.println(out) ;
   }

   // function definition
   public static int cube (int x) {
        return x * x * x ;
   }
}
```

void return type

- A function that does not return a result is declared as void:

```
public void display (String x)     // function signature
```

- The execution of a function finishes:
  - Either when a `return` statement is executed
  - Or, if there is no `return` statement, at the end of the function body

```
public class SimpleOutput {
   public static void main (String[] args) {
        int n = 3;
        String out = "The number is " + n;
        display(out);
   }
   // function definition
   static void display (String x) {
        System.out.println(x) ;
   }
}
```


No arguments

- A function that does not receive any arguments is declared and called with an empty parameter list ()

```
public static void displayDefaultMessage () // function signature
```

```
public class SimpleDefaultOutput {
   public static void main (String[] args) {
        displayDefaultMessage();
   }
   // function (function) definition
   static void displayDefaultMessage () {
        System.out.println("The default message") ;
   }
}
```

Check list

- What is the problem in the following code snippets?

```
public static int half(int x) {
    int h = x / 2 ;
}
```

```
public static void fun(int x, int y) {
    …
}

public static void main(String[] args) {
    fun(3) ;
}
```


More advanced function-related issues

Variable scope:
Local variables vs. Class variables

Pass-by-value when calling functions

Local variables

- All variables have block scope: They exist within the block they are declared

block {} scope

```
{  // block begins
   int x = 9;// local in block
   System.out.println(x) ;
}  // block ends
```

- When this block is the block of a function or the block of a statement (such as **`if`**, **`for`**, etc.), we say the variable is a local variable of this block
- Local variables in different blocks may have same names

Local variables: Example

- What is the output of the following program?

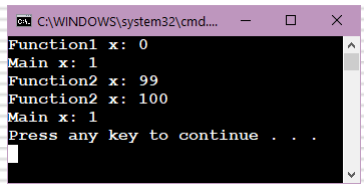
```
public static void main (String[] args){
   int x = 1;   // local in main

   System.out.println("Main x: " + x);
   function1() ;
   System.out.println("Main x: " + x);
   function2() ;
}

public static void function1(){
   int x = 0 ; // local in function
   System.out.println("Function1 x: " + x);
}

public static void function2(){
   int x = 2 ; // local in function
   System.out.println("Function2 x: " + x);
}
```

Class variables

- Variables declared outside of any function block, i.e. within the block of the entire class, are called class variables and have class scope therefore
- **class variables are accessible by all functions of the class**
- ... Unless they are hidden by a local variable with the same name

Class variables: Example

- What is the output of the program?




```
public class ClassVariables {
  static int x = 99 ; // class variable

  public static void main (String[] args) {
    int x = 1; // local in main
    System.out.println("Main x: " + x);

    function1() ;
    System.out.println("Main x: " + x);

    function2() ;
    System.out.println("Main x: " + x);
  }

  public static void function1() {
    int x = 0 ; // local in function1
    System.out.println("Function1 x: " + x);
  }

  public static void function2 () {
    System.out.println("Function2 x: " + x);
    x++ ;
    System.out.println("Function2 x: " + x);
  }
}
```

Nested blocks

- Local variables in nested blocks cannot have the same name as variables declared in outer blocks
- The program on the right will not compile if 2 of the 3 variables named **x** are not renamed

```
public class NestedBlocks {
  public static void main (String[] args) {
    int x = 1; // local in main
    System.out.println( "Main x " + x );

    for ( int i = 0 ; i < 2 ; i++ ) {
      int x = 10 ; // local in outer for
      System.out.println("1st For x " + x) ;
      x++ ;
      for ( int j = 0 ; j < 2 ; j++ ) {
        int x = 0 ; // local in nested for
        x++ ;
        System.out.println("2nd For x " + x);
      }
      System.out.println("1st For x " + x );
    }
    System.out.println("Main x " + x);
  }
}
```


Compiler error! (of previous program)

```
C:\Users\istamatopoulou\Desktop\JAVA
   tests\NestedBlocks.java:8: error: variable x is
   already defined in function main(String[])
      int x = 10 ; // local in outer for
          ^
C:\Users\istamatopoulou\Desktop\JAVA
   tests\NestedBlocks.java:12: error: variable x is
   already defined in function main(String[])
        int x = 0 ; // local in nested for
            ^
2 errors

Tool completed with exit code 1
```

Pass-by-value

- In Java when we call a function and provide a variable as an argument, we provide a copy of its value (not access to the variable itself)
- What will be the output of the following program?

```
public class CallingFunctions {
   public static void main (String[] args) {
      int num = 0 ;
      System.out.println(increase(num));
      System.out.println(num);
   }

   public static int increase (int x) {
      x++ ;
      return x;
   }
}
```

```
C:\WINDOWS\System32\cmd.exe
1
0
Press any key to continue . . .
```


Check list

- How do I define a function?
- What is the signature of a function?
  - What information does the signature contain?
- What are the parameters of a function?
- How do I call a function?
- What is the scope of a variable?
- What is class variable?