TP6: Functions in Java

1 Functions

Functions allow to decompose a program in multiple parts. The behavior of a function is dictated by a series of variables: the parameters of the function. At the end of the execution, a function can return a result: its return value. Functions facilitate writing and reading programs. They allow to re-use easily some parts of the code.

A function is defined by:

- *name*
- *list of parameters*: a list of variables with prescribed types which will be written when using the function
- *return type*: the type of the value returned by the function after its execution
- *function body*: the code describing the behavior of the function

1.1 Defining a function

In Java a function definition has the following form:

```
static return_type function_name (list of parameters}{
    // body of the function
}
```

The functions must be defined outside all other functions. It is not possible to define a function inside a function.

Return type. The return type is the type of variable returned by the function. All usual types (`int`, `float`, `boolean`, etc) are possible. In the particular case where a function does not return a value we may use the keyword `void` instead of the return type.

Name of the function. The choice of function names follows the same rules as the variable names: the name must start with a letter or an underscore, and afterwards must be formed of letters, digits or underscores. It is recommended to use *significant names*, which indicate clearly what the function does.

List of parameters. The parameters are variables whose values are known before using the function. These variables are given in the list of parameters. Each parameter is defined by its type followed by its name. When a function takes multiple arguments their definitions are separatesd using commas. When a function does not take any parameters the name of the function is simply followed by a pair of void parantheses.

The variables defined by the parameters exist only while the function is executed. They are created when the function starts its execution and they disappear when the function returns its value.

Function body. The function body is written in a bloc delimited by accolades {}. It consists of a series of Java instructions. Like in any instructions block, it is possible to declare variables which will only exist inside the given block.

The instruction `return` stops immediately the execution of the function. This keyword is followed by an expression whose evaluation will give the value returned by the function. The type of this expression must correspond with the return type of the function. If the function does not return anything (of type `void`) the keyword `return` is not followed by anything (except ;).

A `return` code is implicit at the end of each function. It is therefore possible to write functions which do not return anything.

Examples.

```
// Fonction which does not take any parameters and does not return a
    value
static void hello () {
System.out. println ("Hello world!");
}
```

```
// Function without parameters returning the golden ratio
static double goldenRatio() {
    return (1.0 + Math.sqrt(5.0)) / 2.0;
}
```

```
// Function taking a real parameter, returing the square of the
   parameter
static double square(double x){
    return x * x;
}
```

```
// Function returing the absolute value of an integer parameter
static int abs(int x) {
    if (x<0) {
        return -x;
    }
    else{
        return x;
    }
}
```

```
// Function taking a time period in hours, minutes, seconds
// and outputs the period in seconds
static int time2seconds(int h, int m, int s) {
    int m_total = m + h * 60;
    int s_total = s + m_total * 60;
    return s_total;
}
```

```
// Function computing the greatest common divisor of two positive
   integers
static int gcd(int a, int b) {
```

```
    if (a<b) {
        // change a and b
        int c = a;
        a = b;
        b = c;
    }
    int r = a % b;
    while (r!=0) {
        a = b;
        b = r;
        r = a % b;
    }
}
```

2 Calling a function

When a function is used we say *function call*. To call a function it is enough to write its name followed by a list of parameters between parantheses, separated with commas. These expressions will give the values that the parameters of the function take during the execution. A function returning a value may be used in an expression.

Some examples using functions defined previously:

```
public static void main(String [] args) {
    hello ();
    // using the fonction hello ()
    int x, y;
    System.out. print ("Enter two numbers: ");
    x = input. nextInt ();
    y = input. nextInt ();
    int z = pgcd(x, y); // using the function gcd()
    System.out. println ("The GCD of " + x + " and " + y + " is " +
       z);
    double phi2 = square(goldenRatio()); // using the square() and
       goldenRatio() functions;
    System.out. println ("phi^2 = " + phi2);
    int a;
    System.out. print ("Enter a number (a) : ");
    a = input. nextInt ();
    System.out. println ("a + |a| = " + (a + abs(a))); // function
       abs()
}
```

In Java parameters are always passed using *copies*. The function recieves a copy of the values given as parameters. If an object is given as an input then its reference value (memory address) is the argument given to the function.

This may generate side effects: even though a copy is passed as an argument, since it is a memory address, when using it, the object at that address will be modified by the function.

3 Some advice

Here is a list whith advice to use functions correctly:

- The name of the function should be chosen carefully: it must reflect the behavior of the function
- Just as we do for the name of the function, the names of the parameters should also be explicit
- It is advised to only put one `return` instruction in a function. Sometimes we must work a bit algorithmically to manage this, but it is always possible. The objective is to have a code easy to read and understand. `return` can be just like a go-to instruction, jumping at the end of the current block.
- The body of a function must not be longer than a few dozen lines. If a function is too long, maybe it is worth splitting it into multiple functions (for clarity)
- We will indicate in each function using a comment the role of that function, a description of the parameters and of the returned value.

4 Exercises

Exercise 1 Forest of trees

Re-take what was done for the TP4 and write a function for which the height is given as a parameter. Use this function to output 12 trees with variable random height between 3 and 8.

Exercise 2 Tabulation

Write a function taking as parameter θ expressed in degrees and returning the value of $\sin\theta/\cos\theta$. Use this function to print the returned value, giving as parameters successively $0, 10, 20, ..., 350$.

We may use the predefined functions `double Math.sin(double)` and `double Math.cos(double)` to compute the sine and cosine. Attention, the angle must then be given in radians.

Exercise 3 Mystery algorithm

Consider the following algorithm **Variables**

- a: integer, given by user
- b: integer, given by user
- c: integer, result
- d: integer, intermediary
- e: integer, intermediary

Algorithm
$d \leftarrow b$
$e \leftarrow a$
$c \leftarrow 1$
while $d > 0$ **do**
if $d \mod 2 = 0$ **then**
$d \leftarrow d/2$
else
$d \leftarrow (d-1)/2$
$c \leftarrow c \times e$
end if
$e \leftarrow e \times e$
end while

Write a function implementing this algorithm. The data a and b are the parameters of the function. Output the different values of the function varying the parameter a from 1 and 10 and the parameter b between 0 and 5.

Can you deduce what the algorithm does in the general case?

Exercise 4 Truth table

In this exercise we will divide the program into functions which are well chosen.

Write a program which outputs the truth value of a boolean expression with 3 variables A, B, C. Recall that $\overline{A}$ is the negation of A, logical AND is represented with $\wedge$, logical OR is represented with $\vee$.

Modify the program to output the truth table for the following expressions:

$$(\overline{A} \wedge \overline{B} \wedge C) \vee (\overline{A} \wedge B \wedge \overline{C}) \vee (A \wedge \overline{B} \wedge \overline{C}) \vee (A \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge \overline{C})$$

$$(A \wedge \overline{B}) \vee (B \wedge \overline{C}) \vee (\overline{B} \wedge C).$$

What can you conclude?