ITI 1121. Introduction to Computer Science II

Module 1 — Laboratory 1

Winter 2016

Part I
Editing, compiling and running Java programs

Objectives

- Learning the requirements for this course regarding assignments,
- Becoming familiar with the programming environment, and
- Editing, compiling and running Java programs.

1 Review of the main code conventions of Java

Take a few minutes to review the code conventions for the Java programming language. Refer to the conventions when solving your assignments. Up to 20% of the assignments' grades concerns the conventions and the clarity of your code.

- java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html.
- www.site.uottawa.ca/~turcotte/teaching/iti-1121/assignments/directives.html

2 Source code editor

You should clearly understand concepts such **compiler**, **source code editor** and **Java Virtual Machine — JVM**. Programming environments such as Netbeans and Eclipse integrate these components into a so-called "Integrated Development Environment" (IDE), which makes it easier to develop code but makes the boundaries between the components fuzzy.

A **source code editor** is a text editor designed specifically to edit source code of programs – see for example Figure figure1. These software usually have the following characteristics:

- Language syntax highlighting. Typically, languages elements such as keywords, parameters and strings are displayed differently. It makes it easier to read and debug code. For example, if you forget to close a string, the text that comes afterward will still be highlighted as a string (try it!).
- Source code indentation. You editor understands a set of rules to indent your code. This too facilitate reading and debugging code. For example, if you forget the parenthesis around an **else** block, only your first line of code will be right-indented.

- Bracket matching ({},(),[]). When you position your cursor on a bracket, the corresponding opposite bracket will be highlighted, again helping with reading and debugging code.
- Language keywords auto-complete.

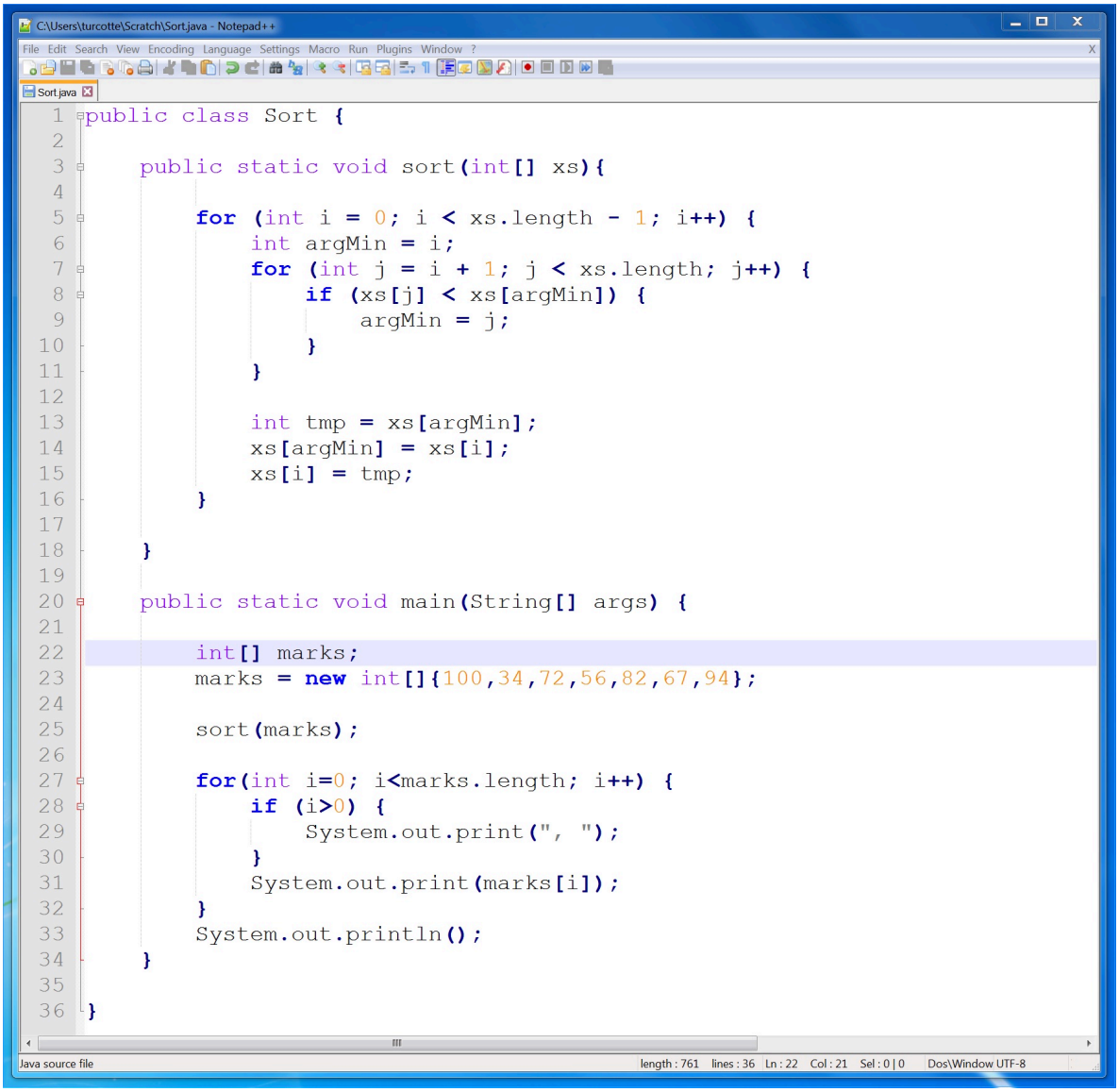

Figure 1: Editing a "selection sort" program using the source code editor Notepad++ in Windows.

Here are some popular source code editor:

- Notepad++ (Windows, http://notepad-plus-plus.org)
- Sublime Text (Mac OS, Windows, Linux, http://www.sublimetext.com)
- TextWrangler (Mac OS, http://www.barebones.com/products/textwrangler/)

- TextMate (Mac OS, http://macromates.com)
- Emacs (Mac Os, Windows, Unix/Linux, https://www.gnu.org/software/emacs/)

Use your favorite editor or development environment (Notepad++ is the recommended environment here) and create a simple "Hello World" program. A "Hello World" program simply consists of a main method that displays the **String** "Hello World". Name this class **HelloWord**. Copy the code from Figure figure2 into your editor and save it as **HelloWorld.java**.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Figure 2: HelloWorld program in Java

3 Compiling and executing a program from the command shell

Open a command shell (a.k.a. command line interface). In the teaching laboratories, the procedure to obtain a shell for running Java commands is simple. Go to the "Start" Menu, select the sub-menu "All Programs", then submenu "Programming", then submenu "Java Development Kit", and you will find an entry "Java-Cmd-xxxx"[1], launch this, and voilà!

```
Start -> All Programs -> Programming -> Java Development Kit->Java-Cmd-xxxx
```

Otherwise (that is, outside the machines in the teaching labs), on Windows, use the"Start" Menu, select "Run" and type "cmd".

Make sure that the current directory is the one where the file **HelloWorld.java** has been created (use the command **cd** followed by the path of that directory, the command **dir** lists the content of the current directory, make sure that you can see your file). See Figure figure3.

Compile the program **HelloWorld.java**

```
> javac HelloWorld.java
```

the symbol ">" is not part of the command, this is simply the prompt, in all my examples. If there were errors, fix them, and compile your program again. If the compilation is successful then you should see a new file in that directory. Its name will be **HelloWorld.class**. The **.class** files contain "byte-code" programs akin to the machine code for microprocessors.

Byte-code is executed by an interpretor called the **Java Virtual Machine** (JVM).

In order to execute your program, type the following in your command window (making sure that the current directory contains the file "HelloWorld.class").

```
> java HelloWorld
Hello World!
```

The result of your **println** statement can be seen in the command window. Here, **java** is the "Java Virtual Machine".

The JVM reads your program's byte-code, one instruction at the time, and performs the necessary actions. Figure figure3 provides an overview of the process.

[1] At the time of writing, "xxxx" is "8u46" but that will probably change overtime.

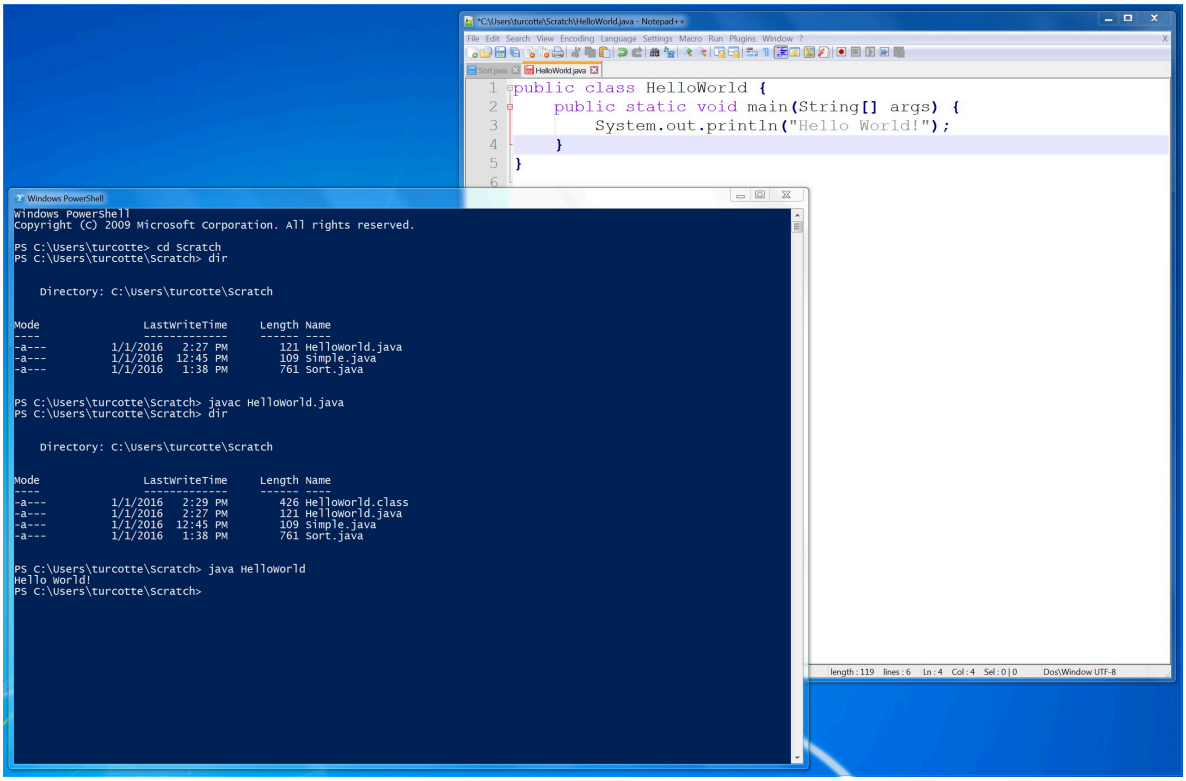

Figure 3: Compiling and running a program from the command line.

4 Integrated Development Environment (IDE)

Eclipse and Netbeans are among the most popular IDEs (http://pypl.github.io/IDE.html). Although we strongly advise that you familiarize yourself with these IDEs prior to your first interviews for a COOP term or for a job, we think that simpler tools such as DrJava or BlueJ are more adapted for now:

- The learning curve of Eclipse and Netbeans is quite steep.
- These powerful environments are really interesting to develop very large applications involving lots of people.
- As mentioned earlier, these environments do hide the boundaries between the components (editor, compiler, virtual machine etc.)
- Code auto-completion increases productivity but hinder learning the language syntax. You put yourself at risk of suffering from the "blank page syndrome" during exams.
- These environments are powerful, they automatically detect and correct some types of mistakes, without providing any explanations. You won't have access to anything like that during exams.
- Finally, DrJava has a interactive console which lets you execute Java instructions without having to create classes.

Redo now the exercise above, but using DrJava. Open the file **HelloWorld.java** using DrJava, compile the program (Figure figure4), and execute it (Figure figure5).

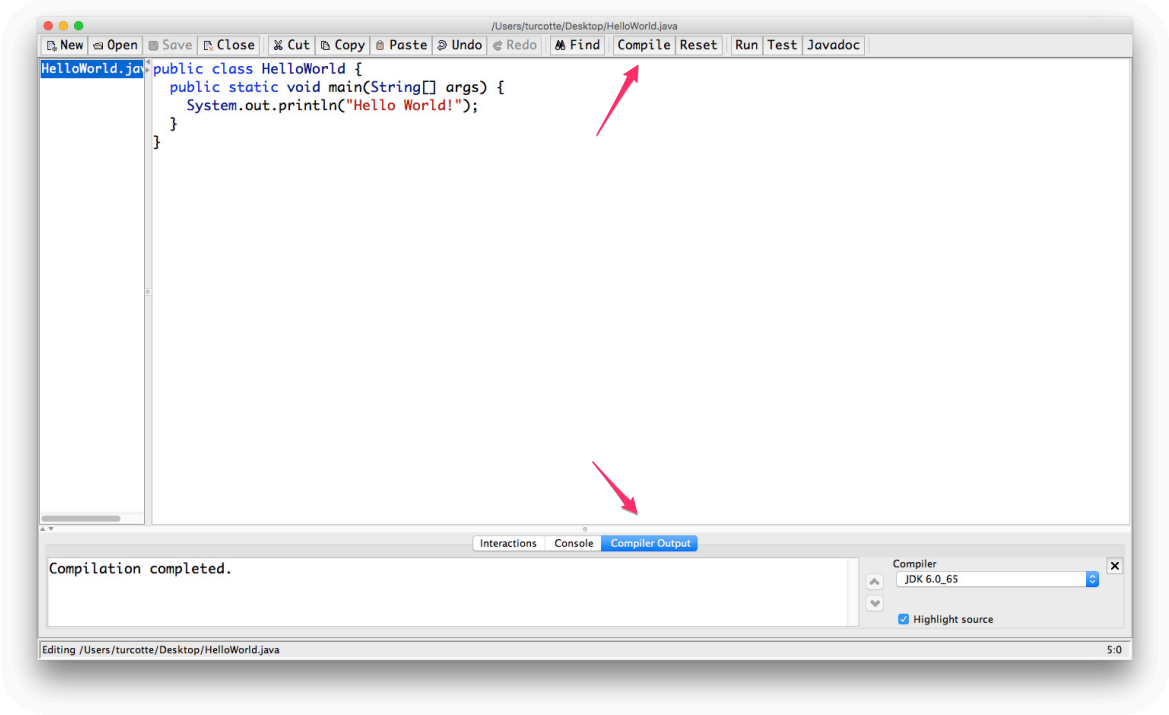

Figure 4: Compiling the program HelloWorld using DrJava.

- See the video on Youtube : https://youtu.be/w1i026fDA_I.

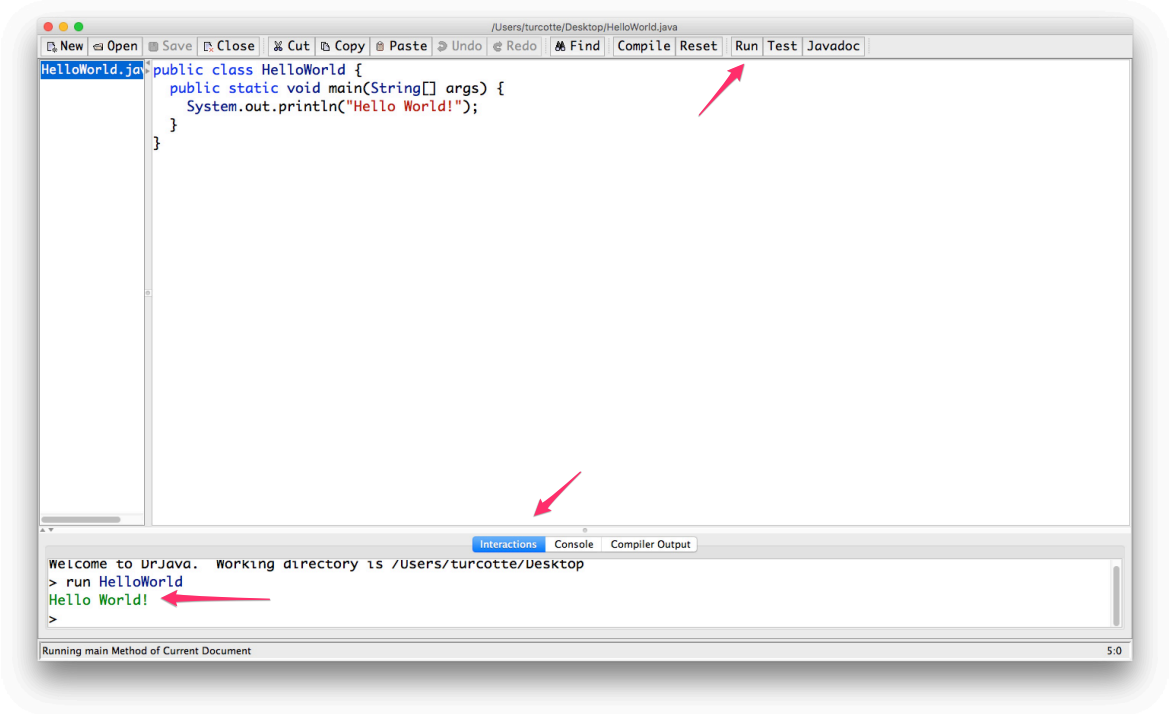

Figure 5: Executing the program HelloWorld using DrJava.

5 Introduction to Java

Many of you have taken ITI1120 in the fall, where you have used Python. Here is an overview of the main differences:

- Java is an Object-Oriented language. All the code is within classes (as can be seen in the HelloWord program).
- Every program must contain a main method, **public static void main(String[] args)**.
- Java is a strongly typed language. You must declare the type of each variable.
- Every instruction ends with a semi-column ";".
- Instructions that belong to the same block must be between curly brackets "{...}".

Here are some resources specifically designed to migrate from Python to Java:

- From Python to Java by Ken Lambert
- Java for Python Programmers by Brad Miller
- Java Tutorials — Lesson: Language Basics

5.1 Selection Sort

Let's use a selection-sort algorithm to familiarize ourselves with Java syntax. This sorting algorithm first identified the location of smallest element in an array, and exchange the value at this location with

the value at the location 0. Then, it identifies the second smallest value in the array, and exchange the value at this location with the value at location 1. The algorithm keeps on going iteratively until the values are sorted in increasing order.

- In Figure figure6, the method **sort** declared at line 3 implements our algorithm. This method has one parameter, called **xs**. This parameter is of type reference, a reference to an array of integers **int**[].
- Line 5, we declare four variables of type integer (**int**).
- Line 7, we have the outermost **for** loop. A **for** loop test contains three parts: initialization, halting criteria and incrementation. Here, in the initialization part we initialize the variable **i** to the value. The loop will stop (halting criteria) once the value of **i** is more than or equal to the length of the array referenced by the (reference) variable **xs**. Finally, at each iteration, the value of **i** is incremented by **1** (**i++**, which is a short cut for **i=i+1**).
- Line 8, we assign the value of **i** to the variable **argMin**. The value of variable **argMin** is going to be used to record the position of the smallest entry in the array between **i** and **xs.length-1**.
- Line 9, we have a nested **for** loop. This loop is used to traverse positions from **i+1** to **xs.length-1** of the array: the value of **j** is initialized with the value of **i+1** and the loop finishes once **j** is larger than or equal to the length of the array. At each iteration, the value of **j** is incremented by 1.
- Line 10, we have an **if** statement. Note that the test is between parentheses. Here, we compare the values of the positions **j** and **argMin** in the array. If the value at position **j** is less than the value at position **argMin**, we store the value at position **j** in the variable **argMin** (line 11). It's the smaller value so far.
- Lines 15, 16 and 17. Once the nested loop is finished, **argMin** contains the location of the smallest value for the part of the array between locations **i** and xs.length-1. We exchange the values at location **i** and **argMin**.
- At line 18 we have the closing bracket for the block starting at line 7.
- At line 20 we have the closing bracket for the block starting at line 3.
- The main method is declared at line 22. This method has one parameter, called **args**, a reference to an array of Strings.
- At line 24, we define a reference variable, which points at an array of integers. This variable is called **marks**.
- At line 25. The right hand side of the statement created an array of integers. The reference of the array is stored in a variable called **marks**.
- At line 27. Call to the method **sort**. The value of the actual parameter **marks** is copied into the formal parameter **xs**.
- In lines 29 35. Printing of the array referenced by the variable **marks**. Note the declaration and initialization of a new variable and its type (**int i=0**) right inside the initialization of the **for** loop.

```java
public class Sort {

    public static void sort(int[] xs){

        int i, j, argMin, tmp;

        for (i = 0; i < xs.length - 1; i++) {
            argMin = i;
            for (j = i + 1; j < xs.length; j++) {
                if (xs[j] < xs[argMin]) {
                    argMin = j;
                }
            }

            tmp = xs[argMin];
            xs[argMin] = xs[i];
            xs[i] = tmp;
        }

    }

    public static void main(String[] args) {

        int[] marks;
        marks = new int[]{100,34,72,56,82,67,94};

        sort(marks);

        for(int i=0; i<marks.length; i++) {
            if (i>0) {
                System.out.print(", ");
            }
            System.out.print(marks[i]);
        }
        System.out.println();
    }

}
```

Figure 6: Java code for a select sort.

6 Command line arguments

Reading data using the input/output system requires creating several objects, but also requires understanding the **Exceptions**, which is the framework for handling error situations in Java. This will be introduced in the lectures, but only once the **Stacks** have been presented.

In the meantime, there is an easy way to pass information to your programs. You have been instructed to write your **main** methods using a signature, and return value, that looks something like this.

```
public static void main( String[] args );
```

When the Java Virtual Machine is instructed to execute your program,

```
> java HelloWorld
```

It looks for a public static method called **main** that has exactly that signature: `public static void main( String[] args )`. However, what exactly is this reference **args** called? First, this is a parameter so it is up to you to select the name of the parameter, you can call it **xxx** if you want to. I prefer using the name **args**, which is the customary way of naming this parameter in the Unix operating system. Now, what is it? It is simply a reference to an array of **String** objects. Have you ever attempted to print its content? Write a new program called **Command** containing a main method. It will be easier to see what is going on if your program prints some information when it starts its execution and when it ends (the exact content of these print statements is not important so be creative!). In between these two statements, using a loop to traverse the array and print the content of each cell. Your program will be more informative if you print the elements of the array one per line. You might as well print the position of the element within the array. Here is the result of my experiments.

```
> javac Command.java
> java Command bonjour true 1121 "java intro" bravo
Start of the program.
Argument 0 is bonjour
Argument 1 is true
Argument 2 is 1121
Argument 3 is java intro
Argument 4 is bravo
End of the program.
> java Command
Start of the program.
End of the program.
> java Command 1
Start of the program.
Argument 0 is 1
End of the program.
> java Command dude
Start of the program.
Argument 0 is dude
End of the program.
```

As you can see, the operating system has handed in to your program an array (of **Strings**) containing all the arguments following the name of the class (here **Command**) on the command line. Quotes can be used for grouping strings together, e.g. “java intro” above. The other important concept to understand is that all the elements of the array are all **String**, although one element is spelled **true**, this is the **String** that contains the letters t, r, u, e. Similarly, although 1121 is a number, in the above context, this is the **String** made of 1, 1, 2, 1.

Question 1

Write the program Command described above

7 Arrays (1)

Let's try something different. We now want to do some simple array manipulations. We are going to write a new program **ArrayTool** which can do a few things on arrays. We are interested in arrays of **double**. Specifically, we will use the following array in this program:

```
double[] valuesArray;
valuesArray = new double[]{100.0,34.0,72.0,56.0,82.0,67.0,94.0};
```

We can base our code on the class **Sort** provided above. The first step is to create a method **printHigherOrLower** which has two parameters, one array of double **xs**, and one value **cutOffValue**, of type double. So the declaration of the method is as follows:

```
public static void printHigherOrLower(double[] xs, double cutOffValue){

     // code goes here

}
```

This method goes through the values stored in "xs", from the first one to the last one, and prints for each value a message stating if that value is lower or higher than the parameter "cutOffValue".

For example, with the array "valuesArray" above, and a "cutOffValue" of 72.5, the method will print:

```
The entry 0 (100.0)  is higher than 72.5
The entry 1 (34.0) is lower than 72.5
The entry 2 (72.0) is lower than 72.5
The entry 3 (56.0) is lower than 72.5
The entry 4 (82.0)  is higher than 72.5
The entry 5 (67.0) is lower than 72.5
The entry 6 (94.0)  is higher than 72.5
```

Question 2

Write the program ArrayTool described above. Test with several values for "cutOffValue" to ensure that your program works.

8 Average of an array

We now want to add a new method to our class. The method, **computeAverage**, has one parameter, an array of type double. It returns a value of type double which is the average of the values contained in the array.

From the **main** of your program, you need to call the method **computeAverage** and store the returned value in some variable of type double. You then need to print out the result.

When I call it with the array **valuesArray** as parameter, mine prints out the following:

```
The average is 72.14285714285714
```

Question 3

Modify the program ArrayTool to include the method computeAverage described above. In your main, after calling the method computeAverage, call the method printHigherOrLower using the computed average as "cutOffValue".

9 Arrays (2)

We now want to **dynamically** create two new arrays of doubles. The first one, **smallerValuesArray**, will contain the values that are lower than the average value, while the other one, **largerValuesArray**, will contain the other values.

To dynamically create an array of double that can accommodate exactly **size** elements (for some integer value **size**), we have to first declare a reference variable to an array of doubles as follows:

```
double[] myArrayOfDouble;
```

Then, once the size is known (that is, once the variable of type **int size** has the correct value), we can create the array as follows:

```
myArrayOfDouble = new double[size];
```

The first thing we will need is to know how many elements of the array have a value lower than a given value. We will create a method **countNumbersLessThan** which has two parameters, one array of double **xs**, and one value **cutOffValue**, of type double (just as the method **printHigherOrLower**). the method will return a value of type **int** which is the number of elements in the array **xs** that have a value strictly lower than **cutOffValue**.

Once we have this method working, we can call it passing on the array "valuesArray" and its average as actual parameters. That will tell us how many elements the array "smallerValuesArray" will have. We can now dynamically create that array.

From these values we can also deduce the necessary size for the array "largerValuesArray". So we can dynamically create that array as well.

Now, we need to create a new method **split** that has four parameters: three arrays of double **double[] xs,double[] low and double[] high** and one double value **cutOffValue**. The goal of this method is to populate the array "low" with the values of "xs" that are strictly less than "cutOffValue", and the array "high" with the other values. In the method, we are going to assume that the arrays "low" and "high" have the necessary size.

Question 4

Modify the program ArrayTool to include the methods countNumberLessThan and split described above. In your main, use the average found by computeAverage as "cutOffValue" for both methods. The main should then print off the values of the elements in both arrays "low" and "high".

In my implementation, I see the following output:

```
The average is  72.14285714285714
The entry 0 (100.0)  is higher than 72.14285714285714
The entry 1 (34.0) is lower than 72.14285714285714
The entry 2 (72.0) is lower than 72.14285714285714
The entry 3 (56.0) is lower than 72.14285714285714
The entry 4 (82.0)  is higher than 72.14285714285714
The entry 5 (67.0) is lower than 72.14285714285714
```

```
The entry 6 (94.0)  is higher than 72.14285714285714
Lower values: 34.0, 72.0, 56.0, 67.0
Larger values: 100.0, 82.0, 94.0
```

Part II
Create and submit a zip file (1 point)

Instructions

- Create a directory **lab1_123456**, where 123456 is replaced by your student number.
- Inside this directory create four directories called **Q1, Q2, Q3** and **Q4**.
- In each directory, copy the file of your program for the corresponding question. In each directory, only put your source code. Do not include any **class files** nor any other files, only Java files.
- At the root of the directory **lab1_123456**, create a file **README.txt** which is a text file containing your name, student number and a brief description of the content of the directories:

  ```
  Student name: Jane Doe
  Student number: 123456
  Course code: ITI1121
  Lab section: B-2

  This archive contains the 5 files of the lab 1, that is, this file (README.txt),
  the file Command.java in the directory Q1 and versions of the file ArrayTool.java
  correpsonding to questions 2 to 4 in directories Q2, Q3 and Q4.
  ```

- Create a **zip** file **lab1_123456.zip** containing the directory **lab1_123456** and all its subdirectory
- Verify that your archive is correct by uncompressing it somewhere and making sure that all the files and the directory structure are there.
- submit the archive using https://maestro.uottawa.ca/

Last modified January 15, 2016