Recap: Fundamentals of Java

Table of Contents

1. Session 0 (lab): The command line

1.1. Compile and execute programs using the command line

1. Session 0 (lab): The command line

1.1. Compile and execute programs using the command line

In this exercise you will learn how to use the command line (a.k.a command prompt) to compile and execute Java programs. Make sure you read this brief tutorial about the Windows command line before starting the exercises.

Section 1

Save the following code in a file called `HelloWorld.java`:

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

Using the Windows command line, compile and execute the code. Use the command `javac` to compile the code and the command `java` to run the program.

```
C:\Users\Alumno>javac HelloWorld.java
C:\Users\Alumno>java HelloWorld
Hello World!
```

As you can see, the text *"Hello World!"* is shown on the command line window. This is because the command line will show anything you print to the standard output (`System.out`).

Run the program again, but redirecting the standard output to a file. You can use the standard output redirect operator "`>`".

Section 2

One way to fetch data to your program is by using command arguments. Command arguments are commonly used to change how your program works or as a way to fetch input data to your program without using the standard input.

The user will chose these arguments when he executes the program. In Java, you can read these arguments from the `main` method, as they are turn into an `array` of `String`s and passed as an argument to the `main` method at runtime.

Practice reading command arguments with the following code. Save it to a file and compile it:

```
public class HelloName {
    public static void main(String[] args) {
        String name = args[0];
        System.out.println("Hello " + name);
    }
}
```

The command arguments are passed to a Java program just by appending them after the call to the Virtual Java Machine invocation, using one or more whitespace as the argument separator:

```
C:\Users\Alumno>java HelloName Juan
Hello Juan
```

You can easily pass more than one argument by appending extra strings separated by whitespaces. Try the following program:

```
public class Adder {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < args.length; i++) {
            sum += Integer.parseInt(args[i]);
        }
        System.out.println(sum);
    }
}
```

This program accepts an arbitrary number of arguments from the command line, each of them interpreted as an integer. As Java programs read their arguments as strings, `Integer.parseInt()` is used to turn the numbers from its textual representation into integers. Play a little bit with this program, using several sets of arguments, like in the example below:

```
C:\Users\Alumno>java Adder 2 -3 6 4
9
```

Section 3

Modify the previous program to instead return the maximum number among its arguments. If no arguments are provided, return the smallest Java `int` (this is, the value of the Java constant `Integer.MIN_VALUE`).

Section 4

Combine the last two programs into a single one. This new program will receive as its first argument the string "sum" or the string "max". The rest of its command line arguments will be numbers. If the first argument is "sum", the sum of the numbers will be returned. If the first argument is "max", the biggest argument will be returned. If the first argument is none of these strings or no arguments are provided, the program will show a sensible error message through its *standard error* (`System.err`).

Here are some examples on how this program should work:

```
C:\Users\Alumno>java AdderOrMax sum 2 -3 6 4
9
C:\Users\Alumno>java AdderOrMax sum
0
C:\Users\Alumno>java AdderOrMax max 2 -3 6 4
6
C:\Users\Alumno>java AdderOrMax max
-2147483648
C:\Users\Alumno>java AdderOrMax blah
Invalid argument: blah
```

Section 5

Refactor the code of the previous program to make the `main` method smaller, by moving the code portions responsible for the sum and the max calculations to separated methods. This will make your program more elegant, reusable and easier to modify in the future. Here is an example on how to do it (you will need to fill the blanks, though):

```
public class NumbersCruncher {

    private int[] integers;

    public NumbersCruncher(int[] integers) {
        this.integers = integers;
    }

    public int sum() {
        int total = 0;
        for (int i = 0; i < integers.length; i++) {
            total += integers[i];
        }
        return total;
    }

    public int max() {
        // TO DO: code this method
        return 0;
    }

    public static void main(String[] args) {
        int[] values;
        if (args.length >= 2) {
            values = new int[args.length - 1];
            for (int i = 1; i < args.length; i++) {
                values[i - 1] = Integer.parseInt(args[i]);
            }
        } else {
            values = new int[0];
        }
        NumbersCruncher cruncher = new NumbersCruncher(values);
        /*
         * TO DO: print the result of calling the sum() and
         * max() methods, or show some error messages if
         * necessary
         *
         */
    }
}
```

Section 6

Add the following functionalities to the previous program, each of them in a separate method of the class `NumberCruncher`:

1. If the first argument is "sum-squared", return the sum of the squares of the numbers passed as arguments. For example:

   ```
   C:\Users\Alumno>java NumbersCruncher sum-squared 3 -4 -1 0 2
   30
   ```

2. If the first argument is "is-neg", return "yes" if at least one of the numeric arguments is a negative number, return "no" otherwise. For example:

   ```
   C:\Users\Alumno>java NumbersCruncher is-neg 3 -4 -1 0 2
   yes
   C:\Users\Alumno>java NumbersCruncher is-neg 3 0 1 7 2 2
   no
   ```

3. If the first argument is "neg", return each numeric argument negated. For example:

   ```
   C:\Users\Alumno>java NumbersCruncher neg 3 -4 -1 0 2
   -3 4 1 0 -2
   ```

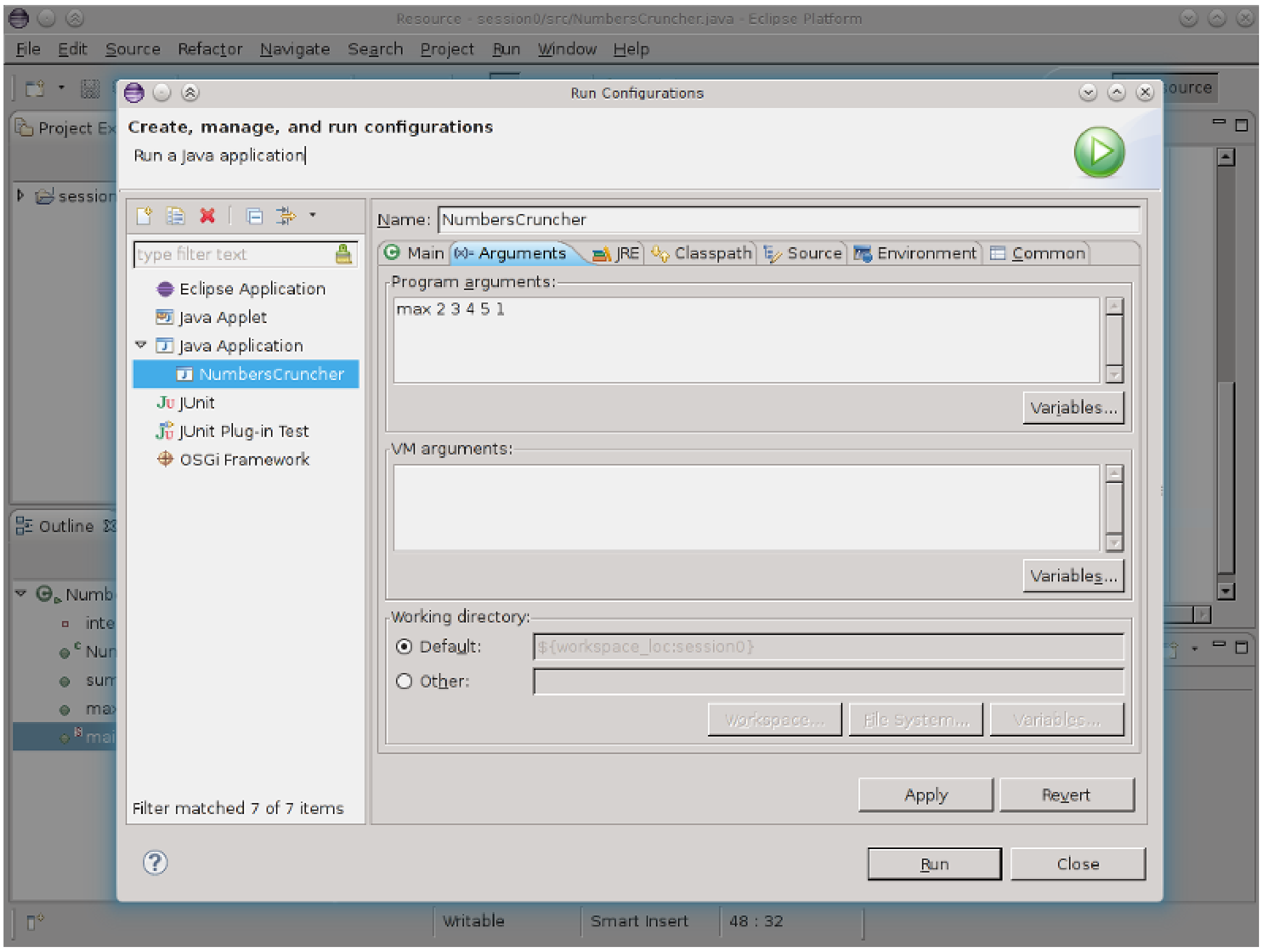
4. If the first argument is "diff", return a new list of numbers each of them being the difference between two of the original numerical arguments, from left to right. For instance:

   ```
   C:\Users\Alumno>java NumbersCruncher diff 3 6 2 10 7
   3 -4 8 -3
   ```

   If less than 2 integers are passed as arguments, print an error message on the standard error stream.

Section 7

If you use Eclipse as your Integrated Programing Environment (IDE), you can also execute your programs with arguments, even if you are no longer using the command line. This is very useful to test your programs without leaving your IDE. To pass arguments to your program in Eclipse, you should open the *Run As / Run Configurations* menu, select the *Arguments* tab and fill in the text field *Program Arguments* as show on the following screenshot:



Name: NumbersCruncher

Program arguments:

```
max 2 3 4 5 1
```

VM arguments:

Working directory:
Default: ${workspace_loc:session0}
Other:

Eclipse shows the standard output and the standard error in a window called the *Console*. You can easily identify data written to the standard error because it is printed in red.

Practice this by executing some of the previous programs inside Eclipse.