Universidad Autónoma de Madrid

Escuela Politécnica Superior

Software Analysis and Design 2019-2020

Assignment 1: Introduction to Java

Starting date:	The first week of February.
Duration:	1 week.
Submission:	**1h before the start of the next assignment, via Moodle.**
Weight in the grade:	5 %

The goal of this assignment is to learn the functionality of the basic tools of the Java Development Toolkit (JDK), including the Java Virtual Machine (JVM) and to write your first Java program.

Part 1: Hello World:

Using a text editor write the following program and save it under the filename HelloWorld.java

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello world!");   // shows the string using stdout
   }
}
```

Compile the program by executing the java compiler (`javac`) using the command line:

```
javac HelloWorld.java
```

Execute it using the JVM interpreter:

```
java HelloWorld
```

Note: The name of the .java file must be the same as the class that it contains and is case sensitive. Make sure that both `javac` and `java` are in your execution path (`PATH` environment variable).

Part 2: Generating programming documentation

The `javadoc` command allows generating HTML documentation on the basis of the java source files. Additional information for the documentation, not intrinsic to the Java code, can be given in pseudo comments starting with `@<tag_name>`. As an example, we will change the previous source file, adding the author name:

```
/**
 * This program prints "Hello world!" at the console
 *
 * @author student-name estudiante.eps@uam.es
 *
*/
public class HelloWorld {

 /**
  * Entry point invoked by OS.
  *
  * This method prints "Hello World!"
  *
  * @param args The command line arguments
  */
   public static void main(String[] args) {
      System.out.println("Hello world!");   // shows the string using stdout
   }
}
```

The HTML documentation can be generated using the `javadoc` command line command:

```
javadoc –author HelloWorld.java
```

Note: It is convenient and good programming practice to generate the documentation in a folder that is different from the one of the source files. For example, if our sources are in the **src** directory, it is better to have the documentation in the **doc** directory:

```
javadoc -d ../doc –author HelloWorld.java
```

Open the index.html in the documentation directory with a WEB browser to see the documentation you have just generated.

You can learn more about javadoc in http://en.wikipedia.org/wiki/Javadoc

Part 3: Using basic libraries

The following program shows how we can obtain and use command line parameters and how to use the class `SortedSet` to keep ordered a set of numbers.

```
import java.util.SortedSet;
import java.util.TreeSet;

/**
 * This class keeps ordered a set of integer numbers
 */
public class Order {
    // we use an ordered set, implemented by TreeSet
    private SortedSet<Integer> numbers= new TreeSet<>();

    /**
     * Constructor, with the array of strings
     * @param strings to insert, after converting them to numbers
     */
    public Order(String ... params){
        for (String s: params){              //we traverse the array
            int n= Integer.parseInt(s);      //convert to integer
            numbers.add(n);                  //add to set
        }
    }
    /**
     *
     * @return numbers
     */
    public SortedSet<Integer> getNumbers(){
        return numbers;
    }

    /**
     *
     * @return String representing this object
     */
    public String toString(){
        return "There are "+ numbers.size()+ " numbers"
                + ": " + numbers; //or +numbers.toString()
    }
    /**
     * Application entry point.
     *
     * This method orders the numbers of the command line
     * @param args The arguments of the command line: should be at least two numbers
     */
    public static void main(String[] args) {
        if (args.length<2) {
            System.out.println("At least two numbers are expected");
            System.out.println("Returns the ordered set");
        }
        else {
            Order c = new Order(args);
            System.out.println(c); // We print the ordered set through the console
            // In java object destruction is automatic
        }
    }
}
```

Save the file under the name *Order.java*.

Execute the program with different parameters e.g. numerical or not, etc. or even without parameters. What happens?

Part 4: Your first Java program (10 points)

We want to create a class named `Primes` that allows us to know if a given number is prime. This class uses an ordered set to store all the prime numbers that are found. It also has a variable named 'max', which is the largest number that has been checked to be prime. Initially we add two simple methods: the first one collects the primes calculated so far, and the second is the method `toString()` that returns the content of the object's data as a string.

```
public class Primes {
   // we use an ordered set, implemented by TreeSet
   private SortedSet<Integer> primes= new TreeSet<>();
   private int max=1;

   /**
    *
    * @return cache with calculated primes
    */
   public SortedSet<Integer> getPrimes(){
       return primes;
   }

   public String toString(){
       return "Primes up to "+ max+ " = "+primes;
   }
```

The most important method of the class is "`boolean isPrime(int n)`", which returns if n is a prime number. You can use the following code as a starting point:

```
public boolean isPrime(int n){
   if (n<2) return false;
   if (n>max) updatePrimes(n);
   return primes.contains(n);
}
```

Next, we only need to write the methods of the lower level. The first one, "`updatePrimes(int n)`", adds the missing prime between `max+1` and `n` to the set of primes and changes `max` accordingly, setting it to `n`. The second one, "`boolean checkPrime(int n)`", is the most basic method, which checks if a number is a prime assuming that the set of all primes up to `n-1` is already calculated, and calculates if `n` is prime.

To iterate through the set of the primes you can use a for loop like this:

```
for (int p:primes) { /* p takes in each iteration an element of the primes set */ }
```

Java also allows a 'classic', C-like for loop structure with explicit indexes like this:

```
for ( <initialization>; <condition>; <iterating the index> ) {
   <loop body>
}
```

The classic form of the loop is useful for arrays or lists, but the sets do not distinguish the elements by their position.

We use + operator to concatenate strings. This operator can be used with any data type, and will convert the operand to a string if any of the two operands is a string. This conversion will also be performed by default if for example, we execute `System.out.println(object)`, because object should be a string. When an object A should be converted to a string the method `toString(A)` is used and is invoked automatically.

As we will learn during this course, the garbage collection in Java is automatic; therefore we do not need to explicitly destroy objects or free memory when they are not used any more.

The source code of the class `Primes` should be in a file named `Primes.java`.

To test it, we must create a `main()` method in the class `Primes`, similar to that in Part 3 of this assignment, expecting more than one number as command line argument. The program should check and print if each number is prime and it should also print all the prime numbers between 2 and the number passed as argument.

Notes:

- To indicate whether some method or attribute is public or private, we must add the keyword `public` or `private` before their declaration. In this case, the methods you need to complete are internal, and have to be declared `private`.
- In order to use library classes (`TreeSet`, `SortedSet`, etc.) without explicitly writing the name of the package as name prefix (`java.util`) every time they are used, we should import them. You can add the following line at the beginning of the source code for that purpose:

```
import java.util.*; //imports all classes form the package java.util
```

Part 5: Optional exercise (1 extra point)

Add a method `primeDivisors` that returns a set of the divisors of the number given as an argument.

```
public SortedSet<Integer> primeDivisors(int n)
```

You should also modify the `main()` method of the `Primes` class to invoke the method and print the divisors of each command line argument, if it is not prime.

Note:

- In the following assignments, it is preferable not to change the existing classes as in the case of `Primes`, but to create subclasses using inheritance.
- To simplify, we do not factorize the number, because we do not need to know how many times each prime factor appears in the argument's factorization.

How to submit:

- You should submit Part 4 and optionally Part 5.
- The names of the authors should be written in all files supplied.
- The submission is via Moodle, by one member of the team.
- The submission should be packaged in one ZIP or RAR file, that has the following name: GR<group_number>_<student_names>.zip
  For example, the group composed by Marisa and Pedro, of group 2261, should submit a file named: GR2261_MarisaPedro.zip
- The structure of the submitted files should be composed of 2 directories:
  - **src**. Source files
  - **doc**. Documentation generated by *JavaDoc*