UNIVERSITÉ DU LUXEMBOURG

Hello World

Programming Fundamentals 2

Goals

- ✯ Understanding the basics syntax of Java.
- ✯ Review simple constructions from PF1 through many small exercises.
- ✯ Using a simple programming workflow: code, compile and run, commit, push.
- ✯ **Relevant videos**:
  - From Python to Java.
  - Object-oriented programming top-down.

Deliverables

1. The code on your Github repository generated by clicking here: `https://classroom.github.com/a/CQ89J6_q`
2. **Reviewer**: Gustavo Koglin (k0glin on Github).
3. **Automated review**: In order to obtain meaningful feedback from the automated correction tool, you must stick to the output shown in the exercises.

Exercise 1 – Automated tests

Before submitting, make sure you pass all the tests provided, otherwise we will not review your code! At anytime, you can use (in the command line, see "Hacking in Bash" lab) `python3 blackbox_test.py hello_world-tests.json` to check how many tests you pass. You should use this script each time you complete an exercise to verify you got it right!

Exercise 2 – Getting Started in Java!

1. Open the terminal and type:

```
mkdir -p BICS/PF2
cd BICS/PF2
git clone Put here the link to your Github repository for this laboratory
cd hello_world
# Open the current directory in a new window (option "-n") of Sublime Text.
subl -n .
```

At this point, you should have an example project opened in Sublime Text.

2. We provide a Java file pre-completed just for you, so you can feel the style and syntax of Java. You can compile and run the project in the terminal by typing:

```
javac -d target src/hello_world/HelloWorld.java
java -cp target hello_world.HelloWorld
```

You should see the output of the execution of the program on your screen.

- Each time you modify your program, you need to recompile it by calling `javac`.
- The option `-d target` specify the directory in which the compiler will put the compiled classes. It mirrors the folder structure of `src` automatically, *e.g.*, `target/hello_world`.
- The compilation produces a file `HelloWorld.class` that can be executed using `java`.
- The option `-cp target` indicates in which directory to find the classes.

Pitfall 1: Be careful about `hello_world.HelloWorld`, the folder and the class are separated by a dot and not a slash!
Pitfall 2: When running the commands `javac` and `java`, you must be at the root of the repository just cloned! If you type `ls`, you should see the `src` directory.

3. Let's now create from scratch a simple Java program.

   1. Create a file `src/hello_world/MessageToTheWorld.java` in the terminal by typing:

```
touch src/hello_world/MessageToTheWorld.java
```

   2. This file is not known to `git`, thus you need to add it:

```
git add src/hello_world/MessageToTheWorld.java
```

      It is good practise to add your file in git immediately after creating it, so you don't forget it.

   3. Using Sublime Text, create a simple program that prints a message to the world of yours, *e.g.*,

```
package hello_world;

class MessageToTheWorld {
  public static void main(String[] args) {
    System.out.println("My name is <your name>, and I'm gonna be a coding legend.");
  }
}
```

      Pitfall 3: Don't forget to write `package hello_world;` at the top of the file. It must match the folder path in which the file is stored (below `src/`). For instance, if you have a file `src/myapp/gui/App.java`, then the file `App.java` must start with `package myapp.gui;`.
      Convention 1: The name of the class must start with an uppercase, and be exactly the same as the name of the file without the `.java` extension.
      Compile and execute this program using the command `java` and `javac` described previously. You should see your message to the world printed on the terminal.

   4. Now you can commit and push your changes on the git repository:

```
git commit -a -m "Completed exercise 1: MessageToTheWorld."
git push
```

   5. At any time, you can check if some files are *untracked* (not known by git because you did not write `git add File.java`) or if some changes are not committed yet using `git status`.

   6. Awesome! You successfully completed the first exercise, it seems you are now ready for the real deal.

Exercise 3 – Soon-a-coding-legend-but-not-today

You should read and understand the `HelloWorld.java` file to complete the following exercises. You can reuse the code of this file. This is especially useful to know how to to read user inputs using `Scanner`.

When you finish this exercise, you should obtain a program that behaves as follows:

```
> java -cp target hello_world.MinMax
Type two integers: 8 4
4
8
How many numbers do you want to type? 4
2
10
0
4
0
10
Type a sequence of numbers (end the sequence with -1):
99
8
2
-1
2
99
```

For the auto-grader to work properly, you must write exactly the same questions (*e.g.*, *Type two integers:*) with the same formatting, spacing, etc. Otherwise the auto-grader might not recognize your answers as valid.

1. Create a file `MinMax.java` with the following code:

   ```
   package hello_world;
   public class MinMax {
     public static void main(String[] args) {
     }
   }
   ```

   Compile the file with `javac -d target src/hello_world/MinMax.java` and execute it with `java -cp target hello_world.MinMax`.

2. Ask the user two integers and print them.
   Pitfall 4: Don't forget to import the class `Scanner` with `import java.util.Scanner`.

3. Print the minimum then the maximum of these two integers.

4. Extract the minimum and maximum of two integers inside the functions `public static int min(int a, int b)` and `public static int max(int a, int b)`. Use these functions to compute the minimum and maximum and printing them (that replaces the previous question).

5. Next, ask the user how many numbers they want to type. Then, read those numbers in a loop, and keep track of the minimum and maximum elements. At the end of the loop, print the minimum and maximum.

6. Same question as the previous one, but the user now ends the sequence of numbers by "-1" when the sequence is fully entered.

Exercise 4 – What?! Already creating objects?

1. Create a file `AdvancedGeometry.java`. Ask the user for the height and width of a rectangle and print it with stars. Example:

```
javac -d target src/hello_world/AdvancedGeometry.java
java -cp target hello_world.AdvancedGeometry
Enter the height of the rectangle: 2
Enter the width of the rectangle: 3
***
***
```

Hint: use `System.out.print` to print something without a newline.

2. Now, we are going to refactor the previous question in an object-oriented style! We want to create a class representing a rectangle with a method enabling us to draw this rectangle. When creating a class, the recipe is always the same:

   1. Create a file `Rectangle.java`.
   2. Add the package line `package hello_world;`
   3. Add the class `class Rectangle { ... }`.
   4. Add private attributes to the class, here `private int height;` and similarly for the width.
   5. Add a constructor which will create an object:

      ```
      public Rectangle(int h, int w) {
        height = h;
        width = w;
      }
      ```

   6. To compile a project with several files:

      ```
      javac -d target src/hello_world/Rectangle.java src/hello_world/AdvancedGeometry.java
      java -cp target hello_world.AdvancedGeometry
      ```

   After performing those steps, you obtain a simple class that can be instantiated to an object. For instance, we can now create two different rectangles in the main method (in `AdvancedGeometry`):

   ```
   public static void main(String[] args) {
     // We create two objects "r1" and "r2" with different height and width.
     Rectangle r1 = new Rectangle(3, 5);
     Rectangle r2 = new Rectangle(10, 50);
   }
   ```

   Clearly, this program does not do a lot, but now we can *ask some services* to these objects, such as printing themselves! For that, we must implement this "service" first. We can add a method `public void draw() {...}` to the class `Rectangle`, in the implementation of this method, we can reuse the code already written in the previous question—the one that print the rectangle! Now that it is done, we can ask this service to the object we created:

   ```
   public static void main(String[] args) {
     // We create two objects "r1" and "r2" with different height and width.
     Rectangle r1 = new Rectangle(3, 5);
     Rectangle r2 = new Rectangle(10, 50);
     // Ask the objects to print themselves.
     r1.draw();
     r2.draw();
   }
   ```

   As a last step, you should rework this main method so it behaves exactly as question 3.1 (that is, asking width and height to the user and printing a single rectangle).

3. Our advanced geometry program is a great success, a massive number of users is now using it daily. But users are eager for more functionalities... They now want to be able to draw a triangle, and they show you some use case scenarios:

```
java -cp target hello_world.AdvancedGeometry
Please select what shapes we can draw for you:
1. Rectangle
2. Triangle
3. Cross
Please enter your choice: 2
Enter the base of your triangle: 6
*
**
***
****
*****
******
```

Another scenario is as follows:

```
java -cp target hello_world.AdvancedGeometry
Please select what shapes we can draw for you:
1. Rectangle
2. Triangle
3. Cross
Please enter your choice: 3
Enter the height of your cross: 5
Enter the width of your cross: 2
Error: the width must be an odd number!
Enter the width of your cross: 3
Enter the intersection height: 2
 *
 *
 *
***
 *
```

You sight... But that could have been much more complicated, especially to draw the triangle... After realizing that, you feel much happier and get it done in the blink of an eye!

Exercise 5 – Scientific computing

Scientists love to manipulate array of numbers, transform it, write it, cut it, paste it, save it, load it, check it, quick rewrite it. Oops, I got carried away. Anyways, you want to create a class that helps your fellow scientists to deal with arrays of numbers. A use case scenario is as follows:

```
java -cp target hello_world.ScientificComputing
How many numbers do you get today? 4
Please enter an array of 4 numbers: 2 1 4 8
2 1 4 8
1
8
2 3 7 15
2
15
2 1 4 8
```

1. Create a class `ScientificComputing` with a main method. Create a class `ArrayComputing` with an array as a private attribute, and a constructor `ArrayComputing(int size)` which initializes this array with zeroes. Provide a method `public void readArray(Scanner scanner)` that reads the array from the terminal, and `public void print()` which prints the array to the terminal. In the main method, ask the user the size of the array, instantiate an object `ArrayComputing`, call the method `read()` on it, and then `print()`.

2. Add two methods `public int minimum()` and `public int maximum()` to get the minimum and maximum elements in the current array. Print the minimum and maximum of the array entered by the user at the previous question.

3. Create a method `public ArrayComputing accumulate()` which creates a new object `ArrayComputing` and store inside the accumulated sum of the numbers. For instance, if we read the array `2 1 4 8`, then this method computes the array `2 3 7 15`, that is, the i^{th} number contains the sum of the number from the index 0 to i. In the main method, call this method, print the accumulated array and the minimum and maximum of this array. Finally, print the initial array again.

   Hint: You should create a new object `ArrayComputing` inside this method, *i.e.*:

   ```
   ArrayComputing accumulatedArray = new ArrayComputing(size);
   ```

   Then you can access and modify the private attributes of `accumulatedArray` because you are in a method of the object `ArrayComputing`, *e.g.*,

   ```
   accumulatedArray.data[3] = 42;
   ```

   (I supposed you named the attributes `size` and `data` but you can adapt if not.)