Homework 1: Getting Started

The Java Tutorials: If you haven't done it so far, we recommend going through the "Hello World" Application lesson and the Closer Look at the "Hello World" Application lesson in the Java Tutorials.

The Java Tutorials were written by the folks who developed the language. Consulting them is not a required part of the course, but consider using it whenever you feel that you need some more Java practice.

Part I: Experimenting with Errors

When writing and compiling Java programs, you will run into all sorts of problems. As you will soon discover, the error messages that the Java compiler generates are sometimes cryptic (which is true for most programming languages and compilers). One way to get used to, and understand, these error messages, is to force common programming errors *intentionally* and then read and figure out the resulting error messages. That's what this first exercise is all about.

Take a look at the `PrintSomeNumbers` program that was introduced in class. In this exercise you are required to make some changes to this program, observe how the Java compiler and run-time environment react to these changes, and explain what is going on.

To get started, type the program's code using some plain text editor, and save it using the file name `PrintSomeNumbers.java`. Make sure that your editor does not change the file name into `PrintSomeNumbers.java.txt` or something like this. Next, you have to compile and run the program.

Windows users: double-click the supplied `commandshell.cmd` file. This batch file will open the "command line" (also known as the "Windows console") from your current folder.

Mac users: use “Spotlight” or some other means to find and activate the "terminal" app. Then drag the current folder onto the terminal icon.

Next, compile the program. If the program compiled successfully, proceed to execute (run) it. Your session should look something like this:

```
% javac PrintSomeNumbers.java

% java PrintSomeNumbers
0
1
2
```

```
3
4
5
Done
```

If you don't get similar results, correct your code (using the text editor), and rerun it until the program produces the results shown above.

You now have to make ten changes to the code, one change at a time. For each one of the ten changes, proceed as follows:

a. Make the change, using a text editor.
b. Compile the changed program using the command:
   "`javac PrintSomeNumbers.java`"
c. The program will either compile successfully, or you will get a compilation error message.
d. If the program compiled successfully, execute it using the command:
   "`java PrintSomeNumbers`"
e. If you think that there's a problem, start by identifying what kind of problem it is: compile-time error? Run-time error? Logical error? Write a sentence that describes what went wrong. If you think that there is no error, say it also.
f. Important: Fix the program (undo the change), in preparation for the next change. Or, keep a copy of the original error-free `PrintSomeNumbers.java` file and always start with it.

Here are the changes that we want you to make and observe (one at a time):

1. Change the class name to `printSomeNumbers` (but don't change the file name. Keep it `PrintSomeNumbers.java` like before).
2. Change `"Done"` to `"done"`
3. Change `"Done"` to `"Done` (remove the closing quotation mark)
4. Change "Done" to Done (remove both quotation marks)
5. Change main to man
6. Change `System.out.println(i)` to `System.out.printline(i)`
7. Change `System.out.println(i)` to `println(i)`
8. Remove the semicolon at the end of the statement `System.out.println(i);`
9. Remove the last brace (`}`) in the program
10. Change `i=i+1` to `i=i*1`
11. You are welcome to try other changes, as you please.

This is a self-practice exercise: there is no need to submit anything. Do it for you own sake. Write down the answers (for yourself), and make sure that you understand what went wrong.

Stopping a program's execution in the command shell: In some cases, typically because of some logical error, a Java program will not terminate its execution, going into an *infinite loop*. In

such cases, you can stop the program's execution by pressing `CTRL-c` on the keyboard (press the "`CTRL`" key; while keeping it pressed, press also the "`c`" key).

Part II: Programs

In the remainder of this homework assignment you will write five simple Java programs. The purpose of this first exercise is to get you started with Java programming, learn how to read API documentation, and practice submitting homework assignments in this course. The programs that you will have to write are relatively simple. That is because we haven't yet covered the programming idioms `if`, `while`, and `for`, which are essential for writing non-trivial programs.

API comment: When you write programs in Java, you often have to use library constants and functions like, say, `Math.PI` and `Math.sqrt`. If you want to read the API documentation of any of these things, you can google, say, "`java 10 Math`". This will open the API documentation of Java's library's `Math` class, and you can then proceed to search the constant or method that you wish to use within this web page. The "10" is the Java version we'll be using.

1. Coins

Assume that there are two coins only. A coin of 25 cents, called *quarter*, and a coin of a single cent, called *cent*. Write a program (`Coins.java`) that gets a quantity of cents as a command-line argument, and prints how to represent this quantity using as many quarters as possible, plus the remainder in cents. Here is an example of the program's execution:

```
% java Coins 132
Use 5 quarters and 7 cents

% java Coins 50
Use 2 quarters and 0 cents

% java Coins 28
Use 1 quarters and 3 cents
```

Note: the output of your program should be formatted *exactly* like the output shown above. The text "1 quarters" doesn't sound great, but so be it (for now).

2. Future Value Calculator

Write a program (`FVCalc.java`) that computes the future value of a sum of Dollars, invested over a period of *n* years at a fixed annual interest rate of *rate* percents. The relevant formula is $futureValue = currentValue \cdot (1 + rate)^n$. The program gets the current value, the interest rate, and the number of years as command-line arguments, and computes the future value:

```
% java FVCalc 100 5.0 2
After 2 years, $100 saved at 5.0% will yield $110.25
```

Note: the output of your program should be formatted *exactly* like the output shown above. This requirement is applicable to all the programs that you will have to write in this course, and we will not repeat it from now on.

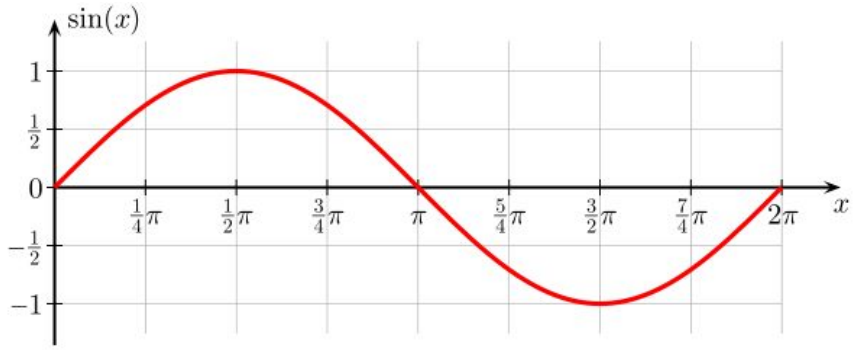
3. Linear Equation Solver

Write a program (`LinearEq.java`) that solves linear equations of the form $a * x + b = c$. The program gets a, b, and c as command-line arguments, computes x, and prints the result. Assume that a is not zero, and that all the supplied values are valid. The program treats the three arguments as well as the solution as `double` values. Here is an example of the program's execution:

```
% java LinearEq 2 5 19
2.0 * x + 5.0 = 19.0
X = 7.0
```

4. Sine function

As shown in the graph below, the Sine function has a period of 2π:



Write a program (`Sine.java`) that demonstrates this property by generating the following output:

```
% java Sine
sin(0)      = 0.0
sin(1/4 PI) = 0.7071067811865475
sin(2/4 PI) = 1.0
sin(3/4 PI) = 0.7071067811865475
sin(4/4 PI) = 0.0
sin(5/4 PI) = -0.7071067811865475
sin(6/4 PI) = -1.0
sin(7/4 PI) = -0.7071067811865475
sin(8/4 PI) = 0.0
```

The purpose of this exercise is to become familiar with the *Sine* function, which will come to play later in the course. Use Java's `Math.PI` constant and `Math.sin()` function.

The code of this program is a bit boring – you'll have to write 9 similar statements (copy-paste will come handy here).

Java uses *floating point arithmetic* to calculate operations on *double* values. We will have more to say about it later in the course. For now, note that your program may evaluate *sin*(0), *sin*(π) and sin(2π) not as 0 (the correct theoretical value), but as a tiny value like 2.4492935982947064E-16 = -2.449 * 10^{-16}, which is very close to 0. That's ok.

5. Gen3

Write a program (`Gen3.java`) that generates three integers, each in a given range [*a*,*b*), i.e. greater or equal to *a* and less than *b*, prints them, and finally prints the minimal number that was generated. Here is an example of the program's execution:

```
% java Gen3 10 15
14
11
10
The minimal generated number was 10

% java Gen3 10 15
12
12
14
The minimal generated number was 12

% java Gen3 90 200
198
95
112
The minimal generated number was 95
```

Note: If you want, you can use Java's `Math.min()` method.

6. Piazza

In this course we use a Q&A forum called "Piazza". As a student in this course, you are expected to visit this forum occasionally. To get started, go to the Q&A forum, click '`new post`', and select the '`meet`' tag. Then write something about yourself: hobby, favorite animal, whatever, and upload a picture. **IMPORTANT:** Don't forget to select the '`meet`' tag for your post.

Submission

Before submitting your solution, inspect your code and make sure that it is written according to our **Java Coding Style Guidelines**. Also, make sure that each program starts with the program header described in the **Homework Submission Guidelines**. Both documents can be found in

the course website, in Moodle. Any deviations from these guidelines will result in points penalty. Submit the following five files only:

- `Coins.java`
- `FVCalc.java`
- `LinearEq.java`
- `Sine.java`
- `Gen3.java`

Deadline: Submit Homework 1 no later than Thursday, October 25, 23:55. You are welcome to submit earlier.