

Homework 1

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 people who developed the language. Consulting them is not a required part of the course, but is recommended when you feel that you need some more Java knowledge and practice.

Q&A Forum: The course's Q&A forum operates on the "Moodle" platform. Students in this course 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. Don't forget to select the 'meet' tag for your post.

This homework assignment has two parts. Part I is a self-study exercise that should not be submitted. Part 2 consists of five programs that you have to write, test, and submit.

Part I: Experimenting with Errors

When writing and compiling Java programs, you will run into all sorts of problems. The error messages that compilers generate are sometimes cryptic and confusing. 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 lecture 1-1. In this exercise you are asked 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, load the given `PrintSomeNumbers.java` file into a text editor, complete it's missing code, and save it. Make sure that your editor does not change the file name into `PrintSomeNumbers.java.txt` or something like this.

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 the text editor.
- b. Compile the modified 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 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.
- 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 you have to make and observe, one at a time:

1. Change the first line to `class Print5` (but keep the file name as is: `PrintSomeNumbers.java`).
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.println(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 curly brace `}` in the program
10. Change `i = i + 1` to `i = i * 1`
11. You can try other changes, as you please.

This is a self-practice exercise: Write down the answers, for your own learning, and make sure that you understand what went wrong. There is no need to submit anything.

Stopping a program's execution: 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 some 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.

When you write programs in Java, you often have to use library constants like `Math.PI` and library functions like `Math.sqrt`. If you want to learn more about a member of some library, like `Math`, you can google, say, "java 16 Math". This will open the API documentation of Java's `Math` class, and you can then proceed to search the constant or method that you wish to use within this web page. The "16" is the Java version that we use in this course.

1. AddTwo

Write a program (`AddTwo.java`) that adds two given integers and prints the result in a fancy way. The command line is: `java AddTwo a b`. Here are two examples of the program's execution:

```
% java AddTwo 3 5
```

```
3 + 5 = 8
```

```
% java AddTwo 9281 719
```

```
9281 + 719 = 10000
```

Notes

1. The output of your program should look *exactly* like the outputs shown in these examples.
2. You can assume that the user-supplied inputs supplied by the user represent two valid integers, and you don't have to write code that tests this assumption.
3. From now on, you are to operate according to these two default guidelines in every program written in this course, unless the instructions say otherwise.

2. 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 are three independent examples 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 text “1 quarters” doesn’t look great, but that’s ok for now.

3. Linear Equation Solver

Write a program (`LinearEq.java`) that solves linear equations of the form $a \cdot x + b = c$. The program gets a , b , and c as command-line arguments, computes x , and prints the result. Assume that a is not zero. The program treats the three arguments as well as the computed value as double values. The program prints the equation, and its solution. Examples:

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

```
% java LinearEq 3 4 5
3.0 * x + 4.0 = 5.0
x = 0.3333333333333333
```

4. Triangle

Three sides can form a triangle if the sum of the lengths of any two sides is greater than the length of the remaining side. This is known as the *Triangle Inequality Theorem*. For example, the three numbers 3, 4, 5 form a triangle, and the three numbers 2, 3, 6 don’t form a triangle. Write a program (`Triangle.java`) that tests if three given integers form a triangle. Examples:

```
% java Triangle 3 4 5
3, 4, 5: true

% java Triangle 2 3 6
2, 3, 6: false
```

5. Gen3

Write a program (`Gen3.java`) that generates three random integers, each in a given range $[a,b)$, i.e. greater than or equal to a and less than b , prints them, and then prints the minimal number that was generated. Examples:

```
% 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
```

If you want, you can use Java's `Math.min()` method (at this stage of the course we use the terms *method* and *function* interchangeably).

Submission

Before submitting your work for grading, make sure that your code is written according to our [Java Coding Style Guidelines](#). In addition to the code files, create a PDF document that lists the code of all the programs that you wrote, with the indentation, one program per page. You may want to experiment with several “paste special” options, for transferring the code from your code editor to your word processing software. Whatever you do, the printed code must be well indented, and easy to read a review. Use the font Consolas, or Arial, size 12. The name of this file should be `HW1Code.pdf`.

Make sure the following files exist in your repository on GitHub:

- `AddTwo.java`
- `Coins.java`
- `LinearEq.java`
- `Triangle.java`
- `Gen3.java`
- `HW1Code.pdf`