Introduction to CS: Homework 1

Getting started

The Java Tutorials: We recommend taking a 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 extra Java knowledge and practice.

This homework assignment has two parts. Part I is a self-study exercise that should not be submitted. Part 2 describes five programs that you have to write, test, and submit. Part I can be done following Lecture 1-1. Part II can be done following Lecture 1-2.

Important: Once you've retrieved the code for this homework from the GitHub Classroom link (see the instructions on the course website), you can begin working by following the steps outlined below.

Part I: Experimenting with Errors

When writing and compiling Java programs, you will run into all sorts of issues. 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 `Demo0` 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 understand what is going on.

To get started, open the `HW1/code` in the IDE, and select the `Demo0.java` file. Complete the missing code using the IDE's editor, and save the edited file (when we say "IDE", we mean "VS Code").

Next, use the IDE's terminal to compile the program. If the program compiled successfully (no errors), proceed to execute (run) it. Your terminal session should look something like this:

```
% javac Demo0.java

% java Demo0
0
1
2
3
4
5
Done
```

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

Getting Started: In order to do all of the above (loading a program, editing, compiling, running), you must first go over the *From Zero to Code* tutorial (from the course website).

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 IDE's editor.

b. Compile the modified program using the command:
```
javac Demo0.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 Demo0
```

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 `Demo0.java` file and always start afresh 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: `Demo0.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.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 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 error descriptions, 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 `Integer.parseInt(String)`. If you want to learn more about some function, say `Integer.parseInt(String)`, you can write, say, "`java 23 Integer`", in a search engine. This will open the API documentation of Java's `Integer` class, and you can then proceed to search the `parseInt` documentation within this page. The "`23`" is the Java version that we use in this course.

General Note about all the programs that you have to write in all the homework assignments in this course: Unless we say otherwise, you don't have to write code that checks the inputs. In other words, the code that you write can assume that the inputs are valid, as specified in the program's description.

1. Bill Three

(10 points) The `Bill3` program splits a restaurant bill equally among three diners. The program reads four inputs, supplied as command-line arguments. The program should handle the first three inputs as strings, and the fourth as an `int` value. The program divides this value by 3 and prints an output message. Here is an example of the program's execution:

```
% java Bill3 Ron Lisa Dan 100
Dear Dan, Lisa, and Ron: pay 34.0 Shekels each.
```

Implementation notes:

1. The amount that each diner pays is treated as a `double` value, which is rounded up. Therefore, it is possible that the three diners will end up paying together a little more than the required sum, which is just fine. Note: to round up a `double` value, you can use Java's function `Math.ceil(double)`, whose name comes from "ceiling". For example, `Math.ceil(25.19)` returns 26.0.

2. Note: The names should be displayed in reverse order. In this example, Dan (who was passed as the last argument) will be printed first, followed by Lisa, and then Ron.

3. The generated output is somewhat "wordy", since we want you to practice building strings. We suggest to start by writing a simpler output, as follows:

```
Pay 34.0 Shekels each.
```

Once you get this output right, proceed to implement the final version of the program.

4. If needed, you can split long Java statements across multiple lines, by inserting line breaks between valid elements of the statement. These line breaks are treated as "white space" and ignored by the Java compiler.

For example:

```
// Proper line break:
System.out.println("There are " + numDiners +
                   " diners around the table.");

// Invalid line break that causes a compilation error:
System.out.println("There are " + numDiners + " diners
                    around the table.");
```

Complete and test the given `Bill3` program.

2. Future Value

(10 points) This program computes the future value of a sum of money which is put in a saving account (חשבון חיסכון) that earns interest (ריבית) over time. The relevant formula is:

$$futureValue = currentValue \cdot (1 + rate)^n$$

The program gets three inputs. The *currentValue*, which is the invested sum, is assumed to be a nonnegative integer. The annual interest *rate* is assumed to be a percentage (an integer between 0 and 100, inclusive). The number of years *n* is assumed to be a nonnegative integer. These values are supplied as command-line arguments. Here are two program run examples:

```
% java FVCalc 100 10 2
After 2 years, $100 saved at 10.0% will yield $121
% java FVCalc 25000 6 25
After 25 years, $25000 saved at 6.0% will yield $107296
```

Implementation notes: (1) The inputted interest rate is treated as a `double` value. This can be done using the function call `Double.parseDouble(String)`, which is very similar to using `Integer.parseInt(String)`. The resulting value can then be divided by 100. (2) The future value is also treated as a `double`, but should be printed as an `int`. This can be done using a statement like `System.out.println((int) x)`. The `(int)` part of this code is called a *casting operator*. It treats the type of the next variable (here, `x`) as an `int`. (3) To compute the operation "raise x to the power of y", use Java's `Math.pow(double,double)` function.

Financial note: It may be surprising to realize how interest accumulates over time. For example, consider the second program execution above. You are welcome to experiment with other values.

Complete and test the given `FVCalc` program.

3. Numbers in words

The `NumWords` program gets a single integer input, and prints its value in terms of hundreds, tens, and ones.

For example:

```
% java NumWords 517
5 hundreds, 1 tens, and 7 ones.

% java NumWords 35
0 hundreds, 3 tens, and 5 ones.

% java NumWords 8
0 hundreds, 0 tens, and 8 ones.
```

Implementation notes:

1. Outputs like "`1 tens`" and "`0 hundreds`" can be improved, but let's leave it like that for now.

2. What happens if you enter a number that has more than 3 digits? Most likely, your program will still get it right. Give it a try.

Complete and test the given `NumWords` program.

4. Ascending order

(30 points) This program generates three random integers in the range [0,*lim*), where the *lim* input is an integer greater than 0. The program then prints the generated numbers followed by the same numbers in ascending order. Here are three program run examples:

```
% java Ascend 100
91 3 38
3 38 91

% java Ascend 100
12 71 44
12 44 71

% java Ascend 100
82 38 8
8 38 82
```

To make things interesting, you are not allowed to use Java's `if` statement.

Implementation notes:

1. In addition to `Math.random()`, you can use Java's functions `Math.min(int,int)`, which returns the smaller or equal of the two `int` values, and `Math.max(int,int)`, which returns the larger or equal of the two `int` values.

2. To save clutter, we suggest naming the variables that contain the three randomly generated values `a`, `b`, `c`.

3. We suggest starting by first figuring out how to calculate *min* = *min*(*a*, *b*, *c*) and *max* = *max*(*a*, *b*, *c*) using `min(int,int)` and `max(int,int)` function calls. Next, figure out how to find the middle value. Finding these three values is a little riddle which is best solved on paper, away from the computer.

Complete and test the given `Ascend.java` program.

5. Formatting time

(30 points) The `TimeFormat` program gets a time input given in the 24-hour *hh*:*mm* format, and prints the time using the 12-hour AM/PM format. Here are some program run examples:

```
% TimeFormat 10:15
10:15 AM
% TimeFormat 12:05
12:05 PM
% TimeFormat 17:08
5:08 PM
% TimeFormat 23:50
11:50 PM
% TimeFormat 00:00
0:00 AM
```

Input rules: The inputted *hours* and *minutes* must be, respectively, between `0` and `23` and between `0` and `59`, inclusive (midnight is represented as "`00:00`"). Both *hours* and *minutes* are represented by two characters, with a leading '`0`', as needed. For example, the time "5 hours and 8 minutes" is represented as the string "`05:08`".

Output rules: `hours` is printed as a value between 0 and 12, inclusive. If hours is less than 10, say 8, it is printed as a single digit. If the `minutes` value is less than 10, say 8, it is printed with a leading 0, becoming "`08`". Then comes the `AM` / `PM` suffix.

Implementation tip: One way to build the output is using several print statements, one after the other. Java's `print` and the `println` statements do exactly the same thing, except that `println` adds a new line at the end of the printing, and `print` does not. Therefore, `print` can be used to create an output line incrementally. Another possibility is building a string incrementally, and then printing the string using a single `println` statement.

Complete and test the given `TimeFormat` program. Note: In order to write this program you have to wait for lecture 2-1.

Submission guidelines

Before submitting your work for grading, make sure that your code is written according to our Java Coding Style Guidelines.

Submission is done through Github Classroom (GS). Instructions on how do so are provided on the course website. You should *commit* the following files only:

- `Bill3.java`
- `FVCalc.java`
- `NumWords.java`
- `TimeFormat.java`
- `Ascend.java`

Submission deadline: 6.11.2025, 23:55