Assignment 02: Basic Programming

CS101 - Intro To Computer Science

Fall 2023

In the last assignment we experienced the command-line commands that are used to compile and run Java code. In this assignment, we will learn how to use a integrated development environment (IDE) to do this process with the press of a button, as well as a vital tool for any programmer, the debugger.

First, you will go through an exercise in configuring a Java project in an IDE and practice using the debugger. Then you will complete three small programming tasks that print some text to the screen.

Prerequisites:

- The Java Paradigm
- Starter Code

1. Create a Java Project

From now on, we will not compile and run the code on the terminal anymore. We let our IDE take care of that for us. In case you are wondering, our IDE is VS Code.

1.1. Go to the folder that contains your assignments.

1.2. Open that folder in VS Code. You can do this either by starting VS Code and selecting *File –> Open Folder...* from the menu (top left corner), or by executing `"code ."` in the terminal after navigating to your folder.

1.3. Open the command palette from the menu (top left corner) *View –> Command Palette.*

1.4. Enter "Java: Create Java Project..."

1.5. Choose "No build tools", accept the project folder (it should show the folder you just created), and enter the project name: *02_basic_programming.*
VS Code will open up a new window with your new project.

1.6. As you might see, the project already contains a file named *App.java* that contains a very similar code to the one you wrote the last time. However, this file is not needed so you should delete it. This can be done from within VS Code by opening up the explorer (first item on the left side), right-click *App.java* and delete it.

1.7. Add a new file in the *src* folder named *DebuggerBasics.java* and add the following code (you might want to copy&paste it):

```
1  public class DebuggerBasics {
2      public static void main(String[] args) {
3          System.out.println("Hello Debugger!");
4          int a = 1;
5          a = 5;
6          a = a + a * a;
7          a /= 5;
8          a = a * a / a;
9          a = a++ * a++;
10         a = (a * 1428571) / 10000000 + 1;
11         a = ++a * ++a;
12         int b = a - a * a;
13         a = b;
14         a = (a < 0) ? 1 : 0;
15         a = a << 4;
16     }
17 }
```

1.8. On the left side switch to *Run and Debug*.

1.9. Press *Run and Debug*.

The terminal will pop up and print the message "Hello Debugger!". As you can see, the IDE compiled and executed the code for you.

FYI: Knowing your IDE simplifies your life as a programmer drastically. Take your time, watch some videos or read about helpful shortcuts and extension. It's worth it!

2. Debug

Now that we have our project set up, we can try using a debugger on it.

2.1. When you hover over the line numbers, a red dot will appear next to them. Pressing on the red dot will activate a "break point." Go to the line with the print statement (Line 3) and press the red dot to activate a break point.

2.2. Press *Run and Debug*

2.3. You should now be in debug mode, please take note of how your IDE has changed: On the left side you see information about the current program's state, including the values of every variables (red box). You should also see a menu bar with icons has appeared (yellow box). This menu bar is the debugger's navigation bar, which will allow you to step through the code line by line, stop execution, or continue running.

2.4. Execute the line that is currently highlighted by pressing *Step Over* in the navigation bar. The debugger will move to the next line and "Hello Debugger!" should be printed in the terminal.

2.5. Step through the code line by line and write the different values of the variable "`a`" to a text file called `a_output.txt`. To start things off, the first three lines of this file should look like

```
1
5
30
```

By the end you should have 12 numbers stored in the file.

2.6. To stop debugging, press the Stop button in the navigation bar.

Now you know the very basics of how to use a debugger. Use it as much as possible. It will safe you hours of time when you are fixing bugs.

3. Compound Inflation

The value of a dollar changes according to inflation. Sadly, today's dollars are worth more than future dollars, since they lose their purchasing power each month. This problem computes how the value of a dollar depreciates over a period of six months using a constant inflation rate. This question is inspired by Question 2-13 in Liang's Intro to Java textbook and current economic events.

3.1. Add a new file in the *src* folder named *CompoundInflation.java* and add the following code:

```
import java.util.Scanner;

public class CompoundInflation {
    public static void main(String[] args) {
        // complete this function to solve the problem
    }
}
```

3.2. Prompt the user to an amount the user adds to their savings account each month. Once the user has passed an `120` to the prompt, the first two lines in your terminal should show:

```
Please enter your monthly savings rate:
120
```

Use the `Scanner` class's `nextLine()` method to get user input as a `String`. Assume the user will enter a valid number, which may have a decimal. For this assignment we will not worry about handling erroneous user inputs.

3.3. Display the equivalent value of the account, in today's dollars, after the sixth month of savings mixed with heavy inflation. Assume a 0.9% monthly inflation rate.

Use `(int) Math.round(...)` to round all displayed numbers to the nearest dollar. Do not output any decimal places.

Assuming the previous user input, the output should look like this:

```
If you save $120 per month with 0.9% monthly inflation, after
6 months, your account will hold an amount equivalent to $114
today.
```

3.4. Validate the correctness of your program by testing several inputs

input	output
120	114
1600	1516
2100	1989
45.15	43

If you have trouble obtaining the correct results, do not hesitate to use the debugger to figure out where the bug appears. Try doing the computations by hand and then comparing against what the debugger outputs.

4. Population Projector

This question is based on Question 1-11 in Liang's Intro to Java textbook.

The program calculates and displays the projected population for each of the next five years. The U.S. Census Bureau projects population based on the following assumptions:

- Current population: 334,300,850
- Each year has 365 days
- One birth every 9 seconds
- One death every 10 seconds
- One new immigrant every 32 seconds

4.1. Add a new file in the *src* folder named *PopulationProjector.java* and add the following code:

```
public class PopulationProjector {
    public static void main(String[] args) {
        // complete this function to solve the problem
    }
}
```

4.2. Compute the projected population for the next five years and output it exactly like this:

```
Here are the projected population numbers for the next five
years:
- Year 2024: 335636750
- Year 2025: 336972650
- Year 2026: 338308550
- Year 2027: 339644450
- Year 2028: 340980350
```

Consider the following:

- There are no fractional people so all population projections must be integers. If you store people using floating point values you will get the wrong results.
- In Java, if two integers perform division, the result is an integer. The fractional part is truncated. Therefore, in order to get an accurate result, one of the values in the division must be a floating point number (use a double).

5. Running Speed Calculator

This question is based on Question 1-10 in Liang's Intro to Java textbook.

The program computes the average running speed in miles per hour based on user input in kilometers and minutes.

5.1. Add a new file in the *src* folder named *RunningSpeedCalculator.java* and add the following code:

```
import java.util.Scanner;

public class RunningSpeedCalculator {
    public static void main(String[] args) {
        double kmPerMile = 1.609344;
        // complete this function to solve the problem
    }
}
```

5.2. Prompt the user how many kilometers they have run:

```
How many kilometers did you run?
10
```

5.3. Prompt the user how many minutes it took them to run that distance:

```
How many minutes did it take you?
30
```

5.4. Display the average speed of the runner in miles per hour:

```
Your average speed was 12.5 miles per hour.
```

5.5. Validate your program by making sure it starts with the same digits below:

km	min	mph
10	30	12.4...
10	200	1.8...
20.5	375.2	2.0...
15.75	37.6	15.6...

Make sure that the running speeds you show match up to at least the first decimal place,.

6. Submission

Submit the following files:

- DebuggerBasics.java
- CompoundInflation.java
- PopulationProjector.java
- RunningSpeedCalculator.java
- a_output.txt