CSCI165 Computer Science II
Lab Assignment
Spring 2020

Objectives:

- Install and configure Java
- Install and configure JavaFX
- Define environment variables
- Write the classic Hello World program
- Push code to a remote git repository

Read the following document closely. Whenever you see a section marked **TASK,** this is something you need to do. When you see the word **SUBMISSION**, this means the following should be added to your final submission. The instructions may ask for screen shots. These screenshots are required to prove that you correctly followed the instructions. **In my opinion the superior screen shot tools are**

- **Windows:** Greenshot
- **Linux:** Flameshot
- **Mac:** Comes with a nice screen shot tool

I expect you to be able to figure out how to perform screen shots. Do not submit screen shots of your entire desktop or take pictures of your computer with a phone or camera. These will not be accepted. Only capture active windows or snippets of active windows.

TASK: Create a document (Word, Libra, OpenOffice, Pages) named **<your last name>_LabOne. Save this file into the lab-1 directory inside the cloned repository on your machine.** Anytime a screen shot is required you will paste it into this document and **clearly mark what it represents.** Don't screen shot your entire desktop, just the appropriate active window.

TASK: Download and Install Java 13

The first step in learning a new language is the installation and configuration of the language, followed by the classic Hello World program. The Hello World program allows us to ensure that all language features are configured correctly and we can write, build and run programs.

Begin by downloading and configuring the latest version of Java SE. This download will include the entire Java SE code base as well as a host of tools to compile, debug, run and deploy Java programs.

There are many tutorials online that will walk you through this installation. I have linked some videos below . . . choose the video that fits your operating system and follow along. If you need additional information on this task an Internet search will turn up ample results. Here are some important things for you to keep in mind while watching these videos

1. All code examples in this course will be compiled and executed with Java 13
2. Some post installation configuration may have to happen. This may include setting the following environment variables: PATH, CLASSPATH and JAVA_HOME

Windows: https://www.youtube.com/watch?v=bIl48gbFiEc

Mac OSX: https://www.youtube.com/watch?v=y6szNJ4rMZ0

Linux: https://www.youtube.com/watch?v=88_a3h2rTk8

When you have completed the installation and configuration you can check your version using the following terminal command: **java --version**

```
kenneth@dragonborn:~$ java --version
java 13.0.1 2019-10-15
Java(TM) SE Runtime Environment (build 13.0.1+9)
Java HotSpot(TM) 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing)
```

If you get output similar to that shown above then all is well.

If you get a **Command Not Found** error message after installing the language that means that you need to configure your PATH environment variable.

Oracle has some great info on this: https://java.com/en/download/help/path.xml

SUBMISSION: Capture a screen shot similar to the one above showing that you have successfully installed Java 13. Paste this into your submission document.

TASK: Another common configuration is to set a JAVA_HOME variable. Also do this

https://www.baeldung.com/java-home-on-windows-7-8-10-mac-os-x-linux

Make sure you understand this concept as it is common in library configuration. Matter of fact it directly applies to the installation and configuration of JavaFX. You can check your variable using the **echo** command. Windows users will need to place **%%** around the variable name.

```
File  Edit  View  Search  Terminal  Help
kenneth@dragonborn:~$ echo $JAVA_HOME
/usr/lib/jvm/java-13-oracle
```

SUBMISSION: Capture a screen shot similar to the one above showing that you have successfully created the JAVA_HOME variable. Paste this into your submission document.

JavaFX

JavaFX is a software platform for creating and delivering desktop applications, as well as rich Internet applications (RIAs) that can run across a wide variety of devices. JavaFX is intended to replace Swing as the standard GUI library for Java SE, but both will be included for the foreseeable future. JavaFX has support for desktop computers and web browsers on Microsoft Windows, Linux, and macOS. Since the JDK 11 release in 2018, JavaFX is part of the open-source OpenJDK, under the OpenJFX project.

JavaFX 1.1 was based on the concept of a "common profile" that is intended to span across all devices supported by JavaFX. This approach makes it possible for developers to use a common programming

model while building an application targeted for both desktop and mobile devices and to share much of the code, graphics assets and content between desktop and mobile versions. To address the need for tuning applications on a specific class of devices, the JavaFX 1.1 platform includes APIs that are desktop or mobile-specific. For example, JavaFX Desktop profile includes Swing and advanced visual effects.

TASK: Download and Install JavaFX here: https://gluonhq.com/products/javafx/

Be sure that you remember where you extracted the archive as we will need this information for the configuration. I extracted to my **home directory**.

Linux:	/home/<username>
Windows:	C:\Users\<username>
Mac OSX:	/Users/<username>

You can put these libraries where you like, **but you need to know where they are.** We also need to inform the JDK tools where these libraries are. We will do that with another environment variable.

JavaFX Configuration
This is the tricky part. JavaFX used to come bundled with the JDK and as such was included in the default build path. This made compiling and executing GUI applications quite easy. Now that JavaFX is being developed as part of the OpenJFX project, the download and configuration is separate from the JDK.

Let's create a new environment variable to conveniently hold the path to the JavaFX installation location. On my Linux system this path is:

~/javafx-sdk-11.0.2/lib

What is your path? I'll tell you a secret . . . **I do not know.** Revisit the material on setting environment variables from before and create a new variable called JAVAFX

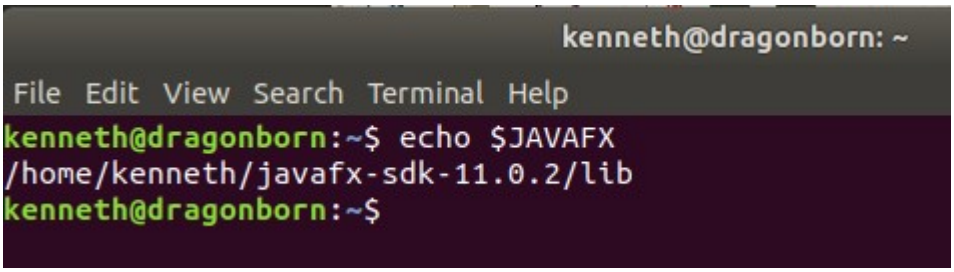
Linux and Mac
Add the following line to the **.bashrc or .profile** file found in your home directory. These files are sourced whenever you start a new terminal session:

JAVAFX="/home/kenneth/javafx-sdk-11.0.2/lib"

IMPORTANT: Obviously you cannot put **/home/kenneth** in this path. Adapt this concept to your machine.

Windows
Click through the labyrinthine menus to find the environment variables window. Add a new variable called JAVAFX and provide the appropriate path.

Check your variable. Linux and Mac



Windows users execute: **echo %JAVAFX%**

SUBMISSION: Capture a screen shot similar to the one above showing that you have successfully created the JAVAFX variable. Paste this into your submission document.

TASK: Test your installation

1. Open a terminal
2. Navigate to the folder containing the file **HelloFX.java** (it came with the repository)
3. Compile and run using the following syntax.

```
kenneth@dragonborn:~/OneDrive/Spring2020/165/JavaFX$ javac --module-path $JAVAFX --add-modules javafx.controls HelloFX.java
kenneth@dragonborn:~/OneDrive/Spring2020/165/JavaFX$ java --module-path $JAVAFX --add-modules javafx.controls HelloFX
```

- The first command compiles the source code to byte code
- The second command launches the application. If all goes well you should see a window resembling the following

Don't worry . . . I know this is a complicated workflow. We will be making this easier when we switch to Eclipse.

SUBMISSION: Capture a screen shot similar to the one above showing that you can successfully execute a JavaFX application. Paste this into your submission document.

TASK: Hello World

Using the code editor of your choice key in the following code and save the file as **HelloWorld.java** Save the file into the cloned repository in the **lab-1** directory. Take care to spell everything correctly and match case. **Java is a case-sensitive language.**

HelloWorld.java

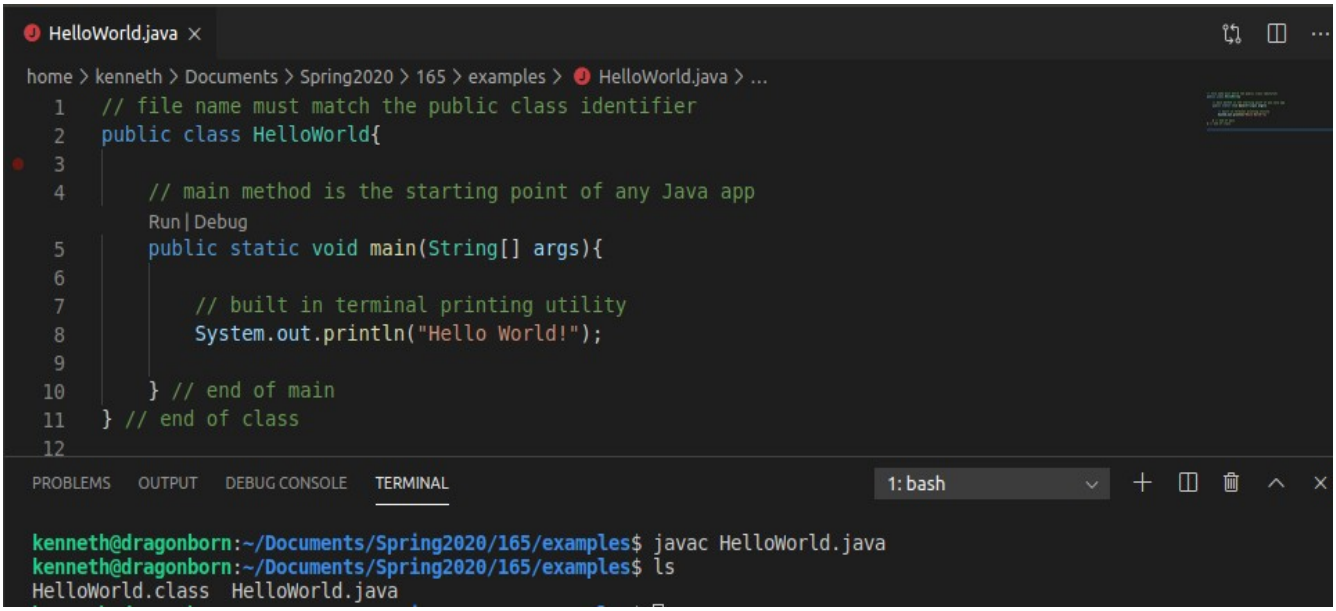
```java
// file name must match the public class identifier
public class HelloWorld{

    // main method is the starting point of any Java app
    public static void main(String[] args){

        // built in terminal printing utility
        System.out.println("Hello World!");

    } // end of main
} // end of class
```

Compile:

In order to execute Java code you must first compile it. This translates the code to an intermediary type called **byte-code**. Byte code is the language that the Java Virtual Machine speaks. The Java compiler is invoked from the terminal using the command **javac.**

Understand that if you did not configure your Java installation successfully, you will receive an error on this step.

1. Make sure you are in the correct directory.
2. Execute **javac HelloWorld.java**



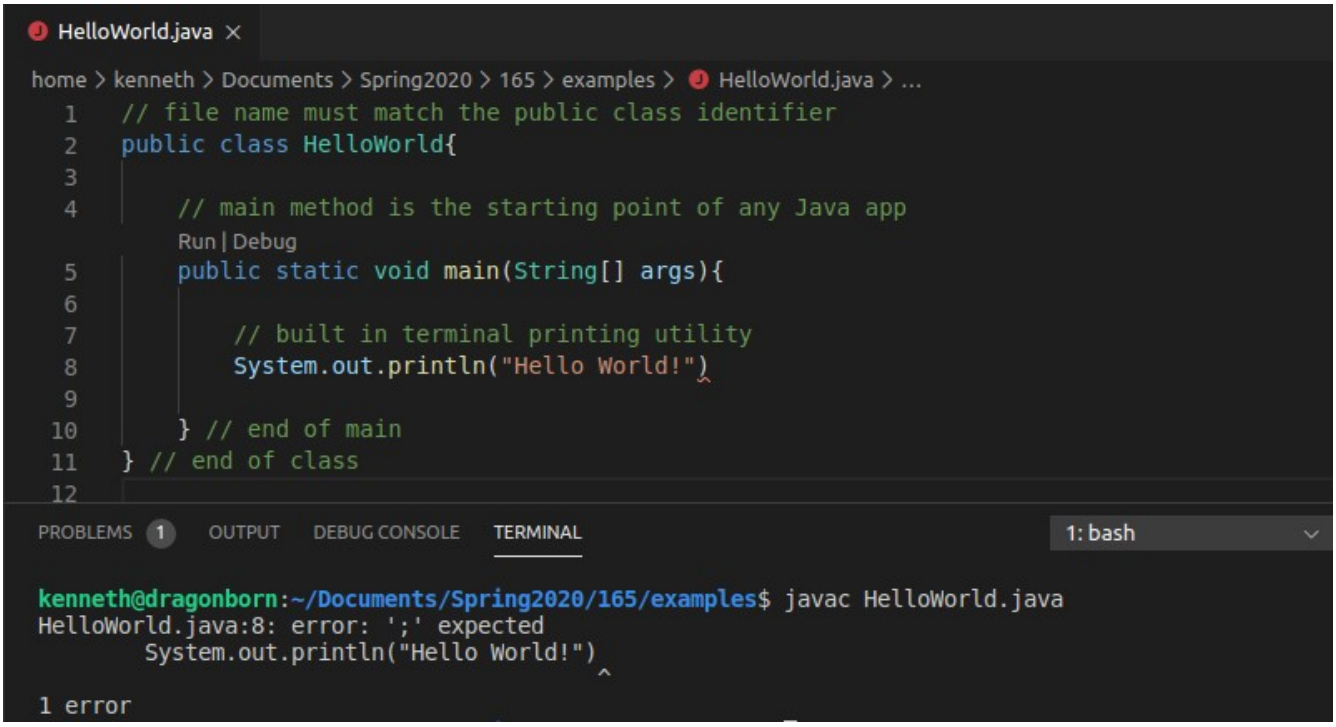
The compilation step is an important one. This is where you will be notified of any syntax errors that may occur. If your code builds with no errors, there will be no messages. You could then do a directory listing and see that a new file was generated. This file will have the same name as the original but it will have a **.class** extension. This class file is the byte-code version of the original source code, ready for consumption by the JVM.

Compilation With a Syntax Error

In this version I have purposefully included a syntax error to show the compiler output. If you receive syntax errors when you build your code you need to

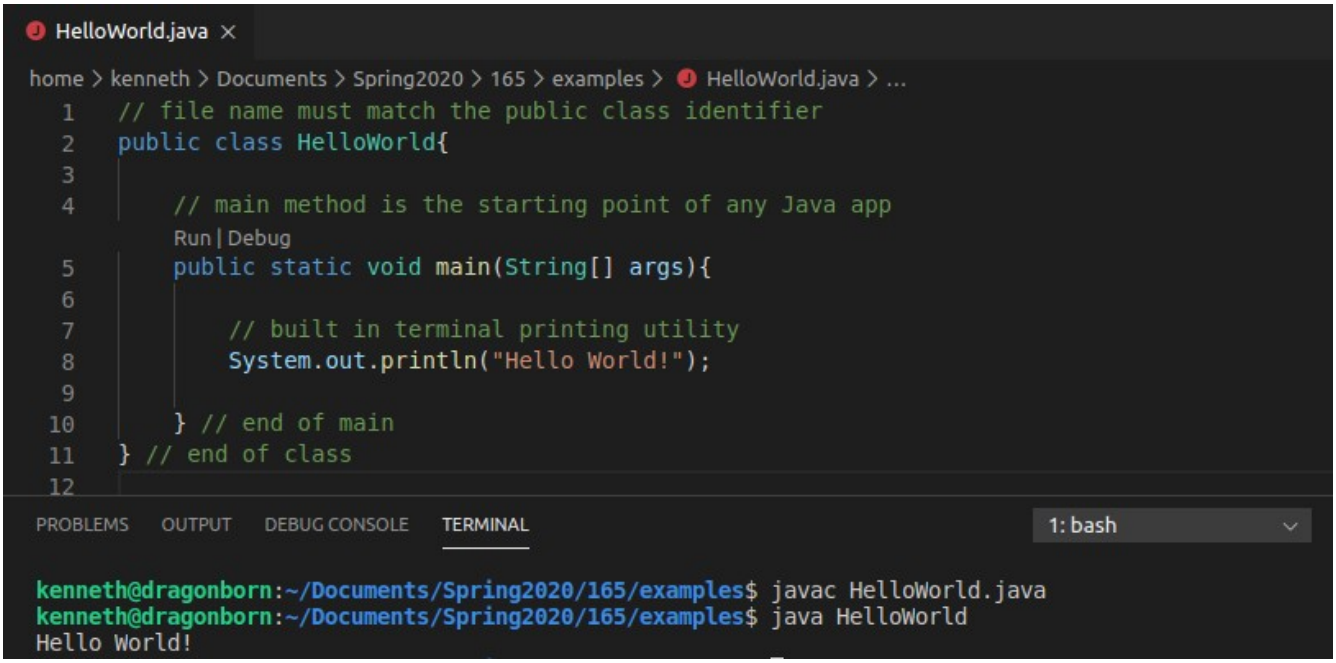
1. **Carefully read the error messages.** Lots of helpful information here
2. Fix the error(s) by editing the source code file.
3. Save your changes and recompile

This error message is telling me I have an error in **HelloWorld.java on line 8** and that the error is due to a missing semi-colon. This is an easy error for the program to spot, but that is not always the case. Pay very close attention and always read error messages.



Program Execution:

Once we have a successful build we can execute the program. Because Java is a compiled and interpreted language, we must invoke the interpreter to execute code. This is accomplished with the **java** command followed by the name of the class file containing a valid **main method.** When executing Java code you leave off the **.class** file extension



SUBMISSION: Take a screen shot of your program being compiled and executed from the terminal. Paste this into your submission document.

Notes on the program structure

Java is known as a verbose language, and it does take a bit of syntax to organize even the simplest of programs into a runnable state; especially compared to a language like Python. Here are some things to focus on understanding at this point. (Understand that some of these items will be discussed in more detail as the course progresses.

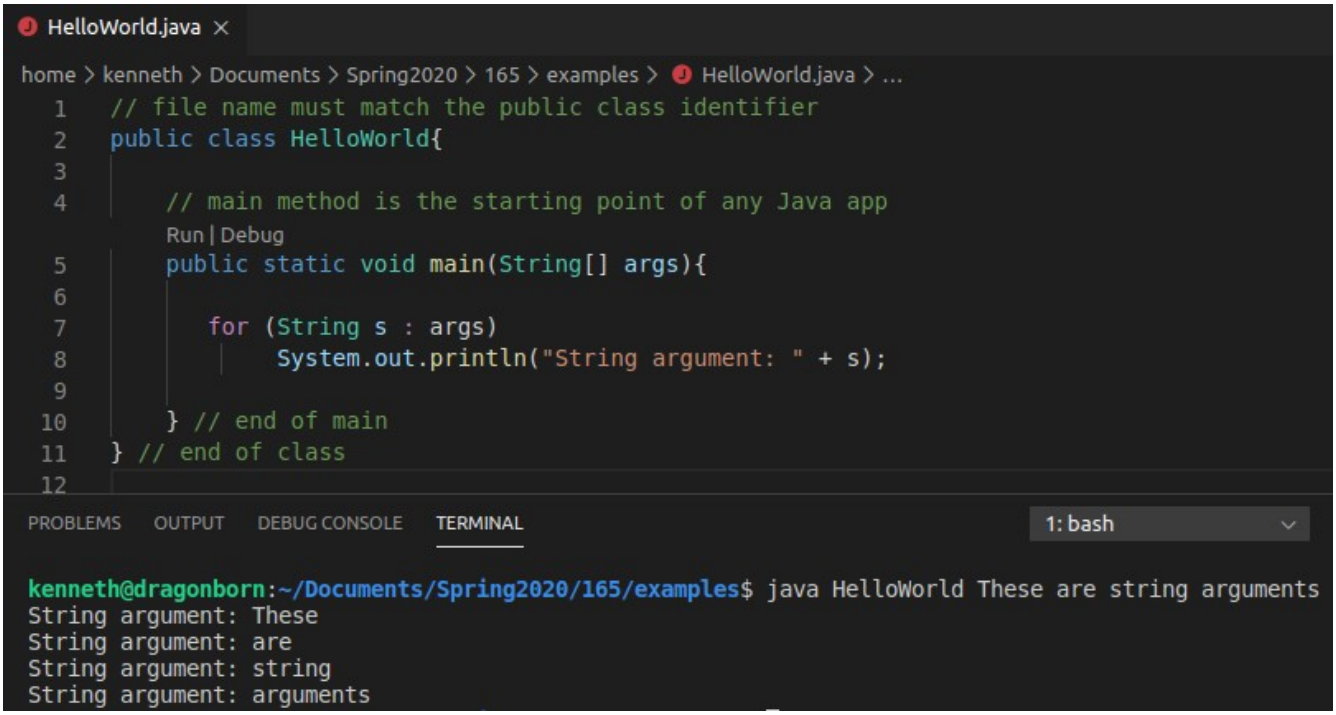
1. **Everything is a class:** Well . . . almost everything. As Java is an object-oriented programming language the **class** is the basic structure. We will be spending the remainder of the semester discussing classes, so at this point just understand that you need a class to create a Java program. Source code file names must match the public class name with a **.java** extension.
2. **{} mark scoped blocks of code:** If you have experience with C/C++, JavaScript or C# you are used to curly braces marking the beginning and ending of a block of code. Where Python uses indentation to mark code blocks, Java uses a **{** to mark the beginning of a code block, and **}** to mark the ending of a code block. These items also mark variable scope. Start paying attention to these braces early on because they are the source of countless syntax errors for Java beginners. Also look at the source code and see if you can determine the difference between class scope and method scope. Java also does not require indentation. You could feasibly write an entire Java program on a single line of code but it would be impossible to read and even more impossible to debug and maintain. In this class you are required to use appropriate indentation to enhance readability.
3. **main is the ignition:** While every class does not require a main method, every Java application does. An application can consist of tens or even hundreds of Java classes, but there is only one main method. This method defines the starting point for an application and is the method that is invoked from the Java Virtual Machine when an application is executed. The signature of this method must be exact or your code will not run. We will be discussing the keywords **public**, **static** and **void** as the course progresses.

4. **Comments are important:** Single line comments in Java are noted with **//.** Internal documentation is important for a well defined program. While I will not expect you to comment obvious things, you are expected to included appropriate documentation. I have found that commenting the ending curly brace of a code block is a helpful reminder.

Command line Arguments:

Java allows programs to accept input from the command line execution of a program. Notice the **String[ ] args** parameter on the main method on line 5 above. This is saying that the main method can accept a list of strings as arguments to the program execution.

In the example below I have included a simple **for . . . each loop** (more on these later) that iterates through the **args array** printing each command line argument. Pay close attention to the output and the command line syntax I used to run the program.



TASK: Define a new Java source file called **HelloWorld2.java** This program will utilize command line arguments to achieve the following results.

```
kenneth@dragonborn:~/Documents/Spring2020/165/examples$ javac HelloWorld.java
kenneth@dragonborn:~/Documents/Spring2020/165/examples$ java HelloWorld Ken Whitener
Hello Ken Whitener. Nice work processing the arguments
```

Note: Java arrays can be indexed in the same fashion as Python lists and C/C++ arrays. Use the overloaded + operator to perform concatenation.

Additional Problems

For problems 1 – 6: Answer the questions in the word processor document. I am not asking you to write code. I am asking to describe an algorithm; but I am looking for a ***detailed explanation***
For Problems 7 – 8: Insert an image of the UML diagrams into the same document
For Problem 10: Create the Java source file in the **lab-1 repository** and add a screen shot of you compiling and running the program from the terminal into your document.

1. **Puzzle Problem:** Suppose your little sister asks you to show her how to use a pocket calculator so that she can calculate her homework average in her science course. Describe an algorithm that she can use to find the average of 10 homework grades.

2. **Puzzle Problem:** A Caesar cipher is a secret code in which each letter of the alphabet is shifted by N letters to the right, with the letters at the end of the alphabet wrapping around to the beginning. For example, if N is 1, when we shift each letter to the right, the word daze would be written as ebaf. Note that the z has wrapped around to the beginning of the alphabet. Describe an algorithm that can be used to create a Caesar encoded message with a shift of 5.

3. **Puzzle Problem:** Suppose you received the message, "sxccohv duh ixq," which you know to be a Caesar cipher. Figure out what it says and then describe an algorithm that will always find what the message said regardless of the size of the shift that was used.

4. **Puzzle Problem:** Suppose you're talking to your little brother on the phone and he wants you to calculate his homework average. All you have to work with is a piece of chalk and a very small chalkboard—big enough to write one four-digit number. What's more, although your little brother knows how to read numbers, he doesn't know how to count very well so he can't tell you how many grades there are. All he can do is read the numbers to you. Describe an algorithm that will calculate the correct average under these conditions.

5. **Determine the Output:** Suppose N is 15. What numbers would be output by the following pseudocode algorithm?

   0 => Print N.
   1 => If N equals 1, stop.
   2 => If N is even, divide it by 2.
   3 => If N is odd , triple it and add 1.
   4 => Go to step 0.

6. **Determine the Output:** Suppose N is 6. What would be output by the algorithm in that case?

   Draw.io is the best option for creating UML diagrams. Be sure to export the diagram as an image

   https://about.draw.io/uml-class-diagrams-in-draw-io/

8. Draw a UML class diagram representing the following class:

    1. The name of the class is Circle.
    2. It has one attribute, a radius that is represented by a double value.
    3. It has one operation, calculateArea(), which returns a double.
    4. Its attributes should be designated as private and its method as public

9. To represent a triangle we need attributes for each of its three sides and operations to create a triangle, calculate its area, and calculate its perimeter. Draw a UML diagram to represent this triangle.

10. **Write a Java program:** To print the following shapes one after another heading down the page. Impress me by using loops. You won't get any extra points but boy I'll be impressed.

```
*****     *****     *****
****      *   *      * * *
**        *   *       * *
**        *   *      * * *
*         *****     *****
```

SUBMISSION: In your **lab-1** directory you should have the following new files

1. **<your last name>_LabOne**
2. **HelloWorld.java**
3. **HelloWorld2.java**

SUBMISSION REQUIREMENTS:

From the terminal, inside the **tc3-csci165-main** repository, run the following git commands

1. **git status:** This should show you that you have un-staged files in your repository. These will show up in red.
2. **git add <file>:** Stage the files for committing by running the **git add** command. You will have to either run this for each file by name, or using the wildcard **git add *** The wildcard is fine but you don't always want to stage **every file . . .** for instance, I don't care about your .class files, **so please do not submit those. This means do not stage these files.** For the inquisitive look into the **.gitignore** file
3. **git commit -m "Submission for Lab 1":** Commits require that you include a message about the commit. Use messages that are descriptive and understand that I will be able to see these. The commit command simply updates your personal repository. The changes are not pushed to the remote repo on GitHub, but the changes are documented as part of your project history.
4. **git push:** This will push your changes to your remote repository on GitHub. You will probably be asked for your GitHub credentials. Enter them
5. You should now be able to see your new files through the GitHub UI

I will grade your work by fetching updates from your personal repository. You do not need to submit anything to Blackboard unless explicitly instructed to.

Rubric

Requirement	Points
Correctness: Code functions as required. Output matches example format	10 points
Puzzle Problems: Obvious effort was put into writing detailed description of algorithms. Descriptions are logically correct	10 points
Style: File naming, Indentation and appropriate comments	2 points
Screenshots: Submitted and complete	3 points
Submission: Followed submission requirements	5 points