CIS351-Java Basic Lab

Submission Instructions

Submit a zip folder via Blackboard. It should contain the following-

1. the final version of the program **Hello.java**
2. the completed worksheet **Lab01.txt**

Download Materials

- Download the worksheet **Lab01.txt** from Blackboard.

Background

Welcome to your first lab! Today's emphasis is on the development cycle of edit, save, compile, and run. We will experiment with different errors, learn to recognize when an error occurs, what information is conveyed in the error message, where the error occurred, and how to resolve it.

Objectives

- Use an IDE (Integrated Development Environment).
- Edit, save, compile, and run a simple Java program.
- Recognize and correct syntax errors in a Java program.

Key Terms

- **java/source file**: the Java program as written by the programmer
- **class file**:the executable program produced by the compiler
- **compile**: process of checking syntax and making a class file
- **syntax error**: mistake in the source code that violates the language
- **logic error**: mistake in the program that causes incorrect behavior
- **execute/run**: the process of running a program on a computer

Before You Begin: Create folder for this course

1. if you haven't installed JDK and and IDE (JGrast, Eclipse or such) yet, please setup your environment first.

2. Throughout the semester, you will download and edit a lot of files. We recommend that you create a separate folder for each lab, homework, and exam.

3. If you haven't already, you should create a top-level CIS351 folder (in your home directory) to organize all your labs and other assignments for the semester. Then create a Lab01 folder for today's lab. Make sure you spell these names as shown; it's important that you don't add spaces.

Part 1: Java Development Cycle

For today's lab, you are recommended to work in JGrasp Editor. JGrasp is a text editor much like TextEdit (macOS) or Notepad (Windows). But it does a lot more! First, it highlights key words as you type them. It also supports indenting of sections of the

code. When you want to compile your code, there is a tool within JGrasp to do so. And when you want to execute and test your program, it will do that too.

0. The first thing you should do when beginning a lab is read *all* the instructions at the top of the page. Don't just skip down to the first question.
1. Open JGrasp and click "File -> New Java Class" from the menu. Name the class Hello **(note the Uppercase H)** and check the box to include a main method. Save the file in your cis351/Lab01 folder.
2. Type in the code below into the editor window. Change the @author to your name and @version to today's date. Pay attention to all spelling, punctuation, and indentation.

```
/**
 * Hello and welcome program.
 *
 * @author Farzana Rahman
 * @version 1/10/2019
 */
    public class Hello {

      public static void main(String[] args) {
           System.out.println("Welcome to CIS 351!");
      }

}
```

   NOTE: Java is case sensitive ("System" is different than "system"). Using the wrong case anywhere will cause an error. Also, Indentation doesn't matter to the Java compiler, but proper indentation makes code much easier for humans to read.

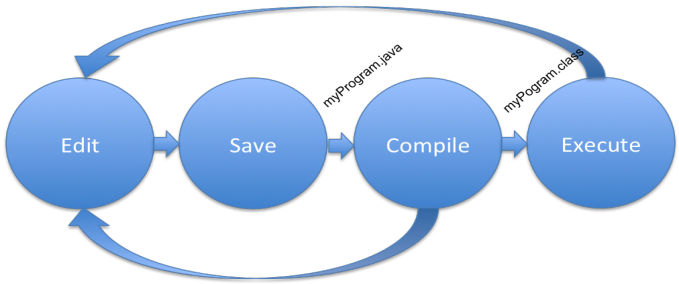
3. Compile your Java program (click the compile button on the toolbar).

   - If it compiles successfully you will get a message: Compilation completed. If you have other messages indicating errors, check your typing very carefully. Your error message will give you the line number of the first place the compiler was confused by what you typed.

   - Upon successful compilation, examine the directory (use the file browser) in which you placed your Hello.java file and you should see a Hello.class file.

4. Execute your program from JGrasp. (Click the run button on the toolbar.) Under the Interactions tab, you should see the message "Welcome to CIS 351!" appear in output panel.

You have just completed the "edit, save, compile, execute" cycle. Each time you change and save your program, you will need to recompile the source file to see the changes reflected in the executed program.



Part 2: Syntax Errors

Byt this time, you should have downloaded **Lab01.txt**. Open the file **using a text editor** (e.g., gedit on Linux, TextWrangler on macOS, Notepad++ on or Notepad Windows).

This part of the lab will give you some practice in reading and interpreting syntax errors. As you make each error, pay particular attention to the message produced, and in some cases, a single error will cascade several other errors. Record the answers to the following questions in your lab worksheet.

5. **Class name different from file name.**

   Delete the beginning 'H' from the name of the class (so the first non-comment line is `public class ello`) and save the program.
   (Question 1) What happens when you try to save it?

   Now compile your program. Keep the `public class ello` mistake in the code.
   (Question 2) What error message do you get during the compile?

6. **Interpreting the error message.**

   All compiler messages will begin with the name of the source file (Hello.java) and the line number in that file that contains the error.
   (Question 3) What line number was your error on?

7. **Misspelling inside a string literal.**

   Correct the mistake above, save, and compile. Next, delete one letter 'l' from the Welcome in the message to be printed (inside the quotation marks). Save the program and recompile it.
   (Question 4) Why is there no error message?

   Now run the program, and review the "key terms" at the top of this lab.
   (Question 5) What type of error just occurred?

8. **No ending quote mark for a string literal.**

   Correct the spelling in the string, then delete the ending quotation mark enclosing the "Welcome to CIS 351!" Save the program and recompile it.
   (Question 6) What error message(s) do you get?

9. **No beginning quote mark for a string literal.**

   Put the ending quotation mark back, then take out the beginning one. Save and recompile.
   (Question 7) What was different about the errors this time?

10. **No semicolon after a statement.**

    Put the missing quote back, and remove the semicolon at the end of the line that prints the message. Save the program and recompile it.
    (Question 8) What error message(s) do you get?

A good practice to follow when you have multiple errors is to focus on the first error, correct it, then recompile. Do not try to figure out all of the errors at once!

Part 3: Declaration and Assignment

Now we will make variables using the Java language. All variables must be declared prior to their first use in the program. A *declaration* is an abstract data type, followed by an identifier, followed by a semi-colon. For example `int sum;` declares the variable sum to be an int (short for integer).Take your previous java file Hello.java. Add the following line to the same program, prior to the `System.out.println` statement.

```
String message;
```

This statement declares message to be a variable container that can hold Strings. Note that String must be capitalized. Skip one line (make one line of white space) and add an *assignment* statement. This statement will put the String literal "Welcome to CIS 351!" into the container called message.

```
message = "Welcome to CIS 351!";
```

You may use a different string if you prefer for your message. Finally, make this change to print the contents of the message variable: meaning, replace `System.out.println("Welcome to CIS 351!");` to `System.out.println(message);`

```
System.out.println(message);
```

Save and recompile your program. Then run it to make sure that it prints Welcome to CIS 351!

(Question 9) Why does `message` not have quotes around it?

Part 4: Manipulating Output

System.out.println sends a string to standard output and adds a newline character at the end. What happens if we use System.out.print instead? This section will have you experiment with the output.

11. In your program, right after where you declared `message,` add in a second String variable named `message2.` Assign to it the value "I'm happy to be a programmer."

12. Change your println to use print instead: `System.out.print(message);` At this point, there is only one print statement in the program. Now, add another println statement on the next line: `System.out.println(message2);` Compile and run your program.
    (Question 10) How many lines of output do you get?

13. Now, after the word "351!" in message, put in the code `\n` . Meaning do this: `message = "Welcome to CIS 351!\n";` This,"\n", is one of the *escape characters* in Java. Recompile and run your program.
    (Question 11) How many lines of output do you get?

14. In Java, `\n` is the newline character and can be used to force a new line wherever we want it. In this case, it is doing the job that the println did before, adding a newline after the last character.

15. Finally, remove the second println command which is printing the second line of the message. Change the other print command to read: `System.out.println(message + message2);` Recompile and run.
    (Question 12) What output do you get?

Grading Criteria

Total points: 100 points

Q1 - Q4: Each worth 5 pt
Q5 - Q12: Each worth 10 pt

Farzana Rahman / frahman@syr.edu