CSE 174 – Fall 2021
Lab1: 20 points – Due Saturday, Aug 28, 2021 by 11:59pm

This first lab assignment is different from future assignments in two very important ways:

1. You will not normally be given Java code and told to copy it.
2. You are NOT expected to understand all of this code. You will probably figure out what some of this code does as you type it, but do not worry if you do not understand all of it.

Typical assignments (in the future) in CSE 174 will ask you to figure out code on your own, and will focus on programming concepts that have already been taught.

Outcomes:

- Use a contemporary programming language and programming environment
- Write, compile, edit, and debug simple Java programs
- Format and comment source code that adheres to a given set of formatting guidelines
- Use the course website

Scoring:

At a bare minimum, the program you submit must have the assigned source code, and your source code must compile and run without crashing.

- If you submit source code on canvas, but not through the Code Plugin, you score will be zero.
- If you submit source code that roughly resembles the requirements and it compiles, but it crashes under normal operating conditions (nice input from the user), your score for this assignment will be 5 points.

	Full credit	**No credit or Partial credit**
Enter and run given code (10 points)	You entered the code as given, and it runs as expected (user input/output, game, triangle).	You entered the code, and it compiles and runs, but it contains multiple errors.
Format and comment source code (5 points)	You followed all of the given formatting requirements (indentation, comments, upper/lowercase, etc.).	You did not follow some or all of the formatting requirements as specified in the requirements.
Make program modifications (5 points)	You successfully made all of the required modifications to the assignment.	You did not make one or more of the required modifications to your program.

Part 1: Get jGRASP set up

- If you already installed the jGRASP you can discard this one. Download and install jGRASP from http://jgrasp.org. Choose the version for your computer that is “bundled” with OpenJDK.
- In jGRASP's Settings menu, choose CSD Window Settings, and make sure the box is checked next to “Line Numbers”. Enabling this option helps you with seeing the line numbers as you write your code.

Part 2: Type your source code

On the next several pages is the source code for a Java program. Using JGRASP, type it in exactly as shown, including indentation, comments, blank lines, upper/lowercase, etc. Use **TAB** to indent, and **Shift-TAB** to outdent.

```java
 1 // Name:
 2 // CSE 174, Section ___
 3 // Date:
 4 // Description: Practice with writing, saving,
 5 //              and compiling code
 6 
 7 import java.util.Scanner;  // Needed for user input
 8 
 9 public class FirstProgram {
10 
11    public static void main(String[] args) {
12 
13    }
14 
15    public static void welcome(String name) {
16 
17    }
18 
19    public static void gameRules(String name) {
20 
21    }
22 
23    public static void drawBorder(int length) {
24 
25    }
26 
27 }
```

Modify this section with your name, section, and date.

Finished? **Save** your work. The file must be named FirstProgram.java

Things to notice:

- **Indentation:** Control the indentation by using the TAB key to indent by one level. To move back out one level, use Shift-TAB.
- **Curly braces:** Lines 9, 11, 15, 19, and 23 contain opening curly braces, and lines 13, 17, 21, 25, and 27 contain closing curly braces. Curly braces are important because the Java compiler looks for them to know when a block of code begins and ends.
- **Comments:** Beginning a line with a double slash, //, indicates that the text is a comment for humans (the compiler ignores it).
- **Color coding:** JGRASP will automatically color code certain words. Your colors may be different depending on the color scheme you choose. These colors help humans read your code.
- **Upper/lowercase:** Note that only a few words begin with an uppercase letter (`Scanner`, `FirstProgram`, and `String`). Java is case sensitive, meaning that the Java compiler considers Scanner, scanner, SCANNER, and ScAnNeR to be different. (You will see later that choosing uppercase vs. lowercase helps humans as well.)

Get to know the basic "structure" of a Java program:

```
 1 // Name:
 2 // CSE 174, Section ___
 3 // Date:
 4 // Description: Practice with writing, saving,
 5 //              and compiling code
 6
 7 import java.util.Scanner;  // Needed for user input
 8
 9 public class FirstProgram {
10
11    public static void main(String[] args) {
12
13    }
14
15    public static void welcome(String name) {
16
17    }
18
19    public static void gameRules(String name) {
20
21    }
22
23    public static void drawBorder(int length) {
24
25    }
26
27 }
```

Basic structure:

- **One class:** Represented by all the code within the yellow rectangle. Notice that the curly brace at the end of line 9 marks the beginning of that block of code, and the curly brace at the end of line 27 ends that block of code.
- **Four methods:** Our class contains four methods, highlighted in orange. The names of these methods are: `main`, `welcome`, `gameRules`, and `drawBorder`. Each method has its own set of curly braces.
- **One of those methods is named main:** When you tell the Java Virtual Machine to run a program, the JVM will look for the method named `main`. Without it, you would get an error message indicating that no `main` method was found.

In CSE 174, most of the programs we write will consist of one class with multiple methods.

Make it easier to spot your curly braces:

One technique that programmers often use to make their code easier to read is to put a comment after the closing brace of each class and method to indicate which code has just ended. Modify your code to include the comments shown below:

```
 9 public class FirstProgram {
10
11    public static void main(String[] args) {
12
13    } // end main
14
15    public static void welcome(String name) {
16
17    } // end welcome
18
19    public static void gameRules(String name) {
20
21    } // end gameRules
22
23    public static void drawBorder(int length) {
24
25    } // end drawBorder
26
27 } // end class
```

Save your work now. Make it a habit to save regularly.

Explain the purpose of your program:

Before writing the code for a class or any of its methods, write comments to summarize the purpose of that code:

- The comments before a class should summarize the purpose of the class as a whole. We have already written some of that, but we will add a little more.
- The comments before a method should summarize the purpose of that specific method. Typically we do not comment the method named main() because its purpose is really the same as the entire class.

Later, we may put comments within a method to explain what part of that method does. For now, add the following comments to your code.

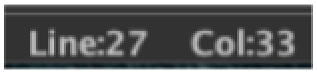
```java
// Name:
// CSE 174, Section ___
// Date:
// Description: Practice with writing, saving,
//              and compiling code. Plays a game of
//              "Guess my number" with the user.

import java.util.Scanner;  // Needed for user input

public class FirstProgram {

   public static void main(String[] args) {

   } // end main

   // Prints a personalized welcome message to the user.
   public static void welcome(String name) {

   } // end welcome

   // Explains the rules of the game.
   public static void gameRules(String name) {

   } // end gameRules

   // Draws a border made of asterisks.
   public static void drawBorder(int length) {

   } // end drawBorder

} // end class
```

Use the Enter key!

Unlike when using a word processor, you need to press the "Enter" key at the end of each line.

As a bonus: when you press enter at the end of a line, JGRASP should automatically indent your code to the same level as the previous line of code.

As a general rule we will look for a way to keep every line of our program at 70 characters or fewer. Notice that JGRASP indicates the current line number and column in the lower right hand corner . For example, this is what would be displayed if the cursor were on line number 27, column 33:


Line:27 Col:33

Can you guess what you should do now? (Hint: It rhymes with "**Dave**")

A key to writing programs is to save, compile, and test frequently.

- Write a little, save it, compile it, and test it.
- Write a little more, save it, compile it, and test it.
- Write a little more, save it, compile it, and test it.

Now is a good time to compile and test your work:

- jGRASP's keyboard shortcut for compiling is Ctrl-B (or Command-B on a Mac). ("B" for "Build").
- If you get a compiler error: Note the line number where the error occurred (often, the actual mistake comes *before* that line number).
- Try to figure out how to fix the problem on your own. If you are stuck, ask your instructor or one of the TAs. It's okay if you don't know/understand something at this point. Ask your instructor or TAs, you don't want to sit there confused, not knowing what is going on.

Don't move on until the above compiles correctly. After compiling successfully, you will have a new file, FirstProgram.class. This is your byte code file that can be run by the Java Virtual Machine. You can run your compiled byte code (Ctrl-R or Command-R), but won't see anything happen yet (because the main method contains no code).

Write the code for the `welcome()` method:

Put your cursor inside the `welcome` method, and type the yellow highlighted code below. We call this the *body* of the welcome method. Notice that the non-highlighted code is what you've already typed. Notice also that `println` stands for "print line". The character between the `t` and the `n` is a lowercase letter `L`.

```
16   // Prints a personalized welcome message to the user.
17   public static void welcome(String name) {
18      System.out.println("Welcome, " + name + ".");
19      System.out.println("This is my first CSE 174 programming assignment.");
20      System.out.println("Let's play \"Guess My Number\"");
21   } // end welcome
```

Did you notice `\"` on line 20? That pair of characters is how you make a quotation mark appear in your printed text. The backslash character is known as an "escape" character, and it announces to Java that the next character is to be treated as a special character.

Save and compile your code (Ctrl-B or Command-B), fixing any errors.
Run your code (Ctrl-R or Command-R), but you still won't see anything happen yet.

Why does nothing happen when you run your code?

The reason is that when you *run* a program, the Java Virtual Machine (JVM) runs the code that you've written in the body of the `main()` method. Right now, you have a `main()` method, but there is no code in the body of that method. Once you start putting code in the body of the `main()` method, *then* you will find that your program actually does something when you click "run".

Fixing indentation: press f2, followed by Shift-f2

JGRASP will automatically handle indentation for you as long as you press "enter" at the end of each line of text that you type. If at any time, you notice that your indentation looks incorrect, do the following:

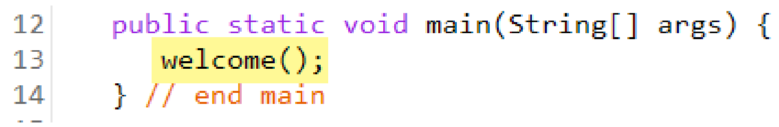
1. You can select everything by pressing Ctrl + A (Command + A).
2. Press F2 (fn + F2) on your keyboard. This turns on jGRASP's "CSD" view of your code, and fixes the indentation.
3. Press Shift-F2. This turns off the CSD view, but now your code will be properly indented.

If JGRASP is still not indenting correctly, there is a good chance that you've made an error in your code (such as missing parentheses, misspelling a Java command, or something similar).

The jGRASP uses 3 spaces for indenting lines. The standard style that we will be using in this course is 4 spaces. But, don't worry, you will learn later how to change it to 4 spaces. Let's move on with 3 spaces for now.

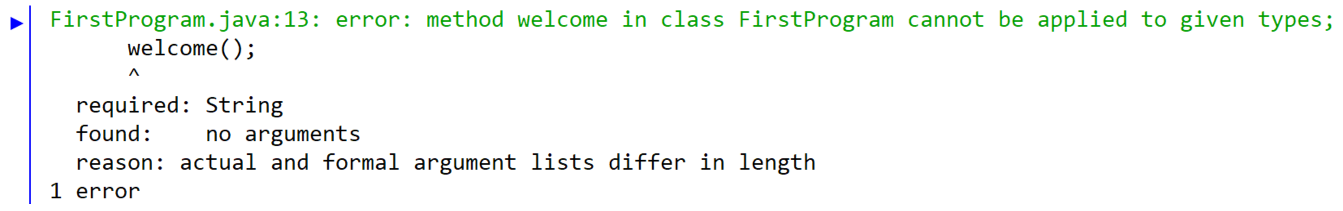
Let's test your welcome() method:

In the main() method, add this highlighted line of code:



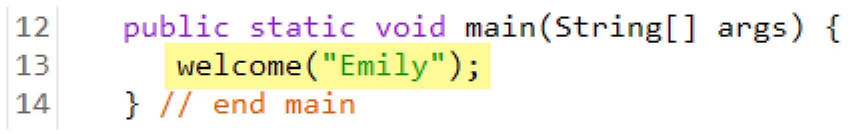
```
12    public static void main(String[] args) {
13       welcome();
14    } // end main
```

Compile. You should get a compiler error:



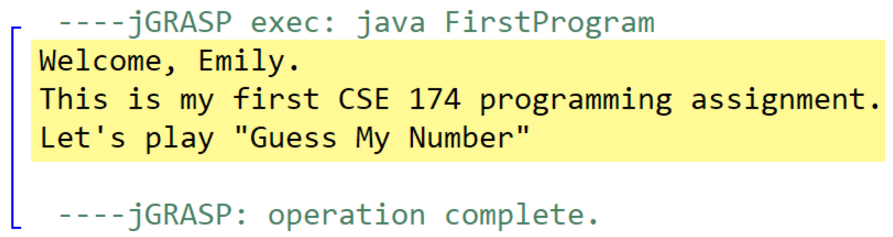
```
FirstProgram.java:13: error: method welcome in class FirstProgram cannot be applied to given types;
      welcome();
      ^
  required: String
  found:    no arguments
  reason: actual and formal argument lists differ in length
1 error
```

This error tells us that welcome() <u>expects a String</u> but we forgot to include a string. So, change the main() method as follows:



```
12    public static void main(String[] args) {
13       welcome("Emily");
14    } // end main
```

Compile again. This time, the compiling should be successful. So, run your program. You should see the following:



```
 ----jGRASP exec: java FirstProgram
Welcome, Emily.
This is my first CSE 174 programming assignment.
Let's play "Guess My Number"

 ----jGRASP: operation complete.
```

You can experiment a little now. Try changing "Emily" to another name (try shorter names and longer names). Each time you make a change to your source code, you will need to compile it again before running.

Type the highlighted code for the body of the `explainGame()` method:

```java
// Explains the rules of the game.
public static void gameRules(String name) {
   System.out.println("Are you ready to play a game, " + name + "?");
   System.out.println("I will think of a number between 1 and 50.");
   System.out.println("Try to guess it in 5 or fewer tries.");
} // end gameRules
```

Save and compile your code, fixing any errors.
To test this method, try putting just this line of code in the main() method, then compile and run:

```java
public static void main(String[] args) {
   gameRules("Mary");
} // end main
```

Type the highlighted code for the body of the `printBorder()` method:

```java
// Draws a border made of asterisks.
public static void drawBorder(int length) {
   for (int i = 0; i < length; i++) {
      System.out.print("*");
   }
   System.out.println(); // moves to the next line
} // end drawBorder
```

Note that one of the lines above contains a `print()` statement, and the other contains a `println()` statement. What's the difference?

- When you print with `println()`, the cursor moves to the next line after it is done printing.
- When you print with `print()`, the cursor stays on the same line when it is done printing.

Save and compile your code, fixing any errors.
To test this method, try putting just this line of code in the main() method, then compile and run:

```java
public static void main(String[] args) {
   drawBorder(20);
} // end main
```

Looking back:
Recall that this class contains 4 methods. So far, you have written code for three of those methods, and written some test code in the `main()` method.
We will now write the part of the program that plays "Guess my Number" by writing code in the main() method. You can delete any other code you wrote in the main() method when you were testing. Notice as you write your code that you will "call" the `welcome()`, `gameRules()` and `drawBorder()` from the `main()` method.

Write the *body* of the `main()` method.

Put your cursor inside the body of the `main()` method. Since this is a longer method, you should frequently compile and run your code. Throughout the code, you will see several stars ★ to indicate that it would be a good idea to save, compile, and run. Don't move on until the program compiles and runs correctly at each star.

(At this point, if you run your program, it should prompt the user for her first and last name, display a greeting, introduce the game, and ask for her first guess.)

```
   public static void main(String[] args) {
      // Declare local variables
      String first, last;
      int correctNumber, guess, guessCount;
      int triangleSize;
      Scanner keyboardReader = new Scanner(System.in); ★

      // Get user's first and last name
      System.out.print("What is your first and last name? ");
      first = keyboardReader.next();
      last = keyboardReader.next(); ★

      // Display border and greeting
      drawBorder(50);
      welcome(first);
      drawBorder(50); ★

      // Explain the rules
      gameRules(first);

      // Start the game with a random number
      correctNumber = (int)(1 + 50 * Math.random());
      guessCount = 0; ★

      // Get first guess
      guessCount++;
      System.out.print("Enter guess #" + guessCount + ": ");
      guess = keyboardReader.nextInt(); ★
```

Continued on the next page…

```java
      // Loop until guess is correct
      while (guess != correctNumber) {

         // Higher or lower?
         if (guess < correctNumber) {
            System.out.println("Guess higher.");
         }
         else {
            System.out.println("Guess lower.");
         }

         // Get next guess
         guessCount++;
         System.out.print("Enter guess #" + guessCount + ": ");
         guess = keyboardReader.nextInt();
      } // end loop

      // By the time we get here, the user has guessed the correct
      // number. Print the results.
      System.out.println("Congratulations, " + first + ".");
      System.out.println("You got it in " + guessCount + " guesses.");

      if (guessCount <= 5) {
         System.out.println("You are an excellent guesser. :)");
      }
      else {
         System.out.println("Try harder next time. :(");
      }

      // Display some art:
      System.out.println();
      System.out.println("And now, some stars to make you happy!");

      triangleSize = 7;
      for (int length = triangleSize; length >= 1; length--) {
         drawBorder(length);
      }

      System.out.println("Goodbye!");

   } // end main
```

Here's what a sample run of the program should look like if you run it. (Notice that Mary Smith is a very lucky guesser.)

```
What is your first and last name? Mary Smith
*********************************************
Welcome, Mary.
This is my first CSE 174 programming assignment.
Let's play "Guess My Number"
*********************************************
Are you ready to play a game, Mary?
I will think of a number between 1 and 50.
Try to guess it in 5 or fewer tries.
Enter guess #1: 15
Guess higher.
Enter guess #2: 22
Congratulations, Mary.
You got it in 2 guesses.
You are an excellent guesser. :)

And now, some stars to make you happy!
*******
******
*****
****
***
**
*
Goodbye!
```

Part 3: Submit the current version of your program

You will be submitting your work more than once for this assignment.
If your program is running as expected submit your current working version. This is a very important step because soon you will be making several changes to your program. So, you will have a version submitted *before* the changes, and then another version submitted *after* those changes. So, it will be as a way of getting backups from your work in different stages.

So, prior to making any of the modifications below, go to **Lab1** on the course website at and submit your source code (FirstProgram.java) for this assignment. When you do, the website automatically changes the name of the file (it might add a number to your file name). Don't worry about that. It's fine. **Do not change the name of your file on your computer. It should be FirstProgram.java at all times, even when you make changes below.**

Part 4: Modify the program

Now it is time to "play around" with your program. Note that you do not need to do any external "research" to solve the following. Instead, look carefully at the program you already typed, and learn some techniques from the code you already typed.

Once the program is working correctly, make all three of these modifications:

1. Even though the program prompts the user for her first and last name, it only displays her first name in the introduction. Don't change the part of the program that asks the user for her name, but modify the program so that the <u>greeting</u> displays the user's first and last name, separated with a space, rather than just the first name. For example, “Welcome, Mary Smith.” rather than "Welcome, Mary."). The game introduction should still only display the user's first name, but the greeting should display the first and last names.

2. The mystery number should be a random number from 1 to 100, rather than 1 to 50, and in order to "win" the game, the number must be guessed in fewer than 10 guesses (rather than 6).

3. Modify the part of the program that prints the triangle by choosing one of the following options. Either:
   - *Reverse* the triangle so that the smallest row is on the top, and the largest is on the bottom.
   - *Ask* the user how many rows should be in the triangle, and display that many rows. For example, if the user says 10, then the triangle would display 10 rows of asterisks, beginning with 10, then 9, then 8, then 7, then 6, and so on.
   - (you can try to do both of these if you want)

What if something goes wrong with your file?

Whenever you upload your work to the course website, you are creating your own personal "backup copy" of your work. If something should go wrong as you make your modifications, remember that you can go back to the course website, locate the file you submitted, and download it to your computer.

Part 5: Submit the modified version of your source code

Once you have made the required modifications to your code, and you find that your code works as expected, it's time to re-upload your <u>source</u> code for the last time. DO NOT CHANGE THE NAME OF YOUR FILE. The course website automatically manages multiple uploads. When you have more than one submission, canvas might add a number to the file name, DON’T WORRY, IT'S OKAY. **On the course website, go to Lab1 and upload your modified <u>source code</u> file.**