ITSC 1212 Module 4 Lab

In this lab, you'll be practicing working with Strings and String methods. You'll also be using the Math class to build a functioning Rock-Paper-Scissors game.

Concepts covered in this lab:

- Return values
- Strings and String methods
- String operators
- Math class

Required files or links

- ITSC1212Lab4.java
- RPSGame.java

Part A: Strings and String methods

1. Start by downloading the ITSC1212Lab4.java file found on the Canvas page for this lab. We'll use this as a starting point for our discussion about Strings.
2. Take a look at the code in this file. Hopefully this should look familiar to you, but if not we have a Scanner object we'll be using to collect user input. Before, we used this to collect numbers, but now we'll use the Scanner to collect String inputs. Add the following lines to your file below the Scanner creation line.

```
String name = scnr.nextLine();
System.out.println("Welcome to ITSC1212 " + name);
```

3. Now, run your code. Remember that Scanners will stop your program and wait for input, so be sure to enter your name in the Terminal window and press enter. Once you do that, you should hopefully see a message that looks like this

```
9a4a1204b0c51e1ede7524b/redhat.java/jdt_ws/ITSC1212_9f3e5672/bin" ITSC1212Lab4
****************************  START  ********************************

Nadia
Welcome to ITSC1212 Nadia

****************************   END   ********************************

(base) CCI183LVDRAWS:ITSC1212 nanajjar$
```

4. Previously when we used the + symbol in our code, it was used with numbers for addition. Notice here the usage is different. When a + is used with Strings, it is for *concatenation* instead of addition. This change in operation applies whenever at least one of the elements on either side of the operator is a String. For example, change your print statement to look like this, and rerun your code.

```
System.out.println("Welcome to ITSC" +1212 + " " + name);
```

5. Notice how the resulting print statement did not change, even though the print statement now includes a combination of String literals, integers, and String variables. You will do this kind of printing often throughout the next two semesters, and getting used to how to format these print statements is vital.
6. For now, let's continue exploring Strings. Another of the things you hopefully remember from the previous section and your prepwork is that String objects, just like Turtle objects, have behaviors we can use by calling methods. Let's quickly review some of these from the prepwork and introduce one new one.

length()	Returns the length of the String object as an integer
indexOf(String *str*)	Returns the index where the String *str* starts in the current String, or returns -1 if it is not found
toUpperCase()	Returns the current String in all caps
equals(String *str*)	Tests if the current String is the same as *str*

7. Most of these seem pretty straightforward, so let's test it. Change your code so it looks like this, and run it.

```
String name = scnr.nextLine();
name.toUpperCase();
System.out.println("Welcome to ITSC1212 " + name);
```

8. Uh oh… What happened? Why do you think it didn't work?

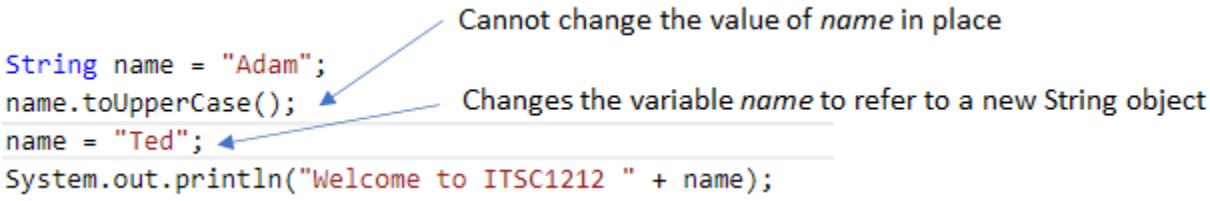
_____

_____

_____

9. You might have expected your name to have appeared in all capital letters, but that is not what happened. To fully understand why that is, we have to introduce two important concepts.

10. The first thing you need to understand is that String objects are immutable. What this means is that a String object's value cannot change. String variables can be changed to refer to different String objects, but the objects themselves cannot be changed. Although that sentence has a lot of technical jargon that might not make complete sense right now, in practice it's actually quite simple.




```
String name = "Adam";
name.toUpperCase();
name = "Ted";
System.out.println("Welcome to ITSC1212 " + name);
```

11. Now that we've established that String objects are immutable, then you might be asking yourself what is the point of toUpperCase() and other methods like it. And the answer is in the word "return" that appears in all those method descriptions. A return is the output or result of a method. By using a return, a method can effectively spit out a value. So although we cannot change the value of a String object, we can return a new String object that contains the desired changes and use that new String object instead.
12. Let's change the code one more time so it looks like this:

```
Scanner scnr = new Scanner(System.in);

String name = scnr.nextLine();

System.out.println("Welcome to ITSC1212 " + name.toUpperCase());
```

13. Run your program, and examine the output. The reason we now see the name in uppercase is that we are no longer printing the value held by the *name* variable, but rather the output (or return value) of the toUpperCase() method.
14. Try the following:

```
String name = scnr.nextLine();
name = name.toUpperCase();
System.out.println("Welcome to ITSC1212 " + name);
```

15. Does this work? Why?

_____

_____

_____

16. Finally, let's test your ability to use String methods and concatenation. Comment out the lines from the first portion of this section including the Scanner object, and add the following:

```
String welcomeMessage = "Welcome to ITSC1212";
String name = "adam"; //Replace with your own name in lowercase
```

17. Using these two Strings as a starting point, plus any others you want to create*, add print statements so that your program outputs the following:

```
WELCOME TO ITSC1212 ADAM
The length of this message is: 24
The word 'to' is located at index: -1
```

*You can create additional String variables (and this will be helpful for getting the length of the message) but you must use the String methods introduced in this section rather than hard-coding the result.

18. When you are satisfied with your code, show it to your instructor or a TA to receive credit for this section.

Part B: Math class

1. The Math class contains a number of useful methods which were highlighted in your prepwork. Let's start by recalling some of those. Complete the following table by describing the function of the following Math methods

Math.random()	
Math.abs(int)	
Math.pow(double, double)	
Math.sqrt(double)	

2. Although all of these can be incredibly useful in the right context, chances are you will end up using random() far more than any others. This is because randomness can be combined with decision making to make programs that branch off in unpredictable ways, which you will see in a moment.
3. For now, let's practice the usage of Math.random(). Fill in the following chart with the correct range of numbers for each usage of Math.random().
   Order of operations is extremely important here.

(Math.random())	
(int) (Math.random())	
(int) (Math.random() * 5)	
(int) (Math.random() * 2)	
(int) (Math.random() * 9 + 1)	

4. Show your completed charts to the instructor or a TA to receive credit for this section.

Part C: Using Math.Random() in a game

1. Now let's put your newfound skill using Math.Random() to good use. Download RPSGame.java, place it into your ITSC1212 folder, and open the file in VSCode.

2. Take a look at the code in this file. This Java class, when run, executes 1 round of a game of Rock-Paper-Scissors. Or at least.. it will once you complete it.

3. If you look through this code, you'll notice a fair number of programming techniques you aren't familiar with yet. The function of the lines that start with "if" and "else" are to examine the state of the game (the choices of rock/paper/scissors) and determine who won based on those choices.

4. There's one other thing to point out about the way this game is structured. It would be entirely possible to use String objects instead of integers to represent the choices between rock/paper/scissors. You might even be tempted to think that it would be easier to use Strings. But by using integers, we can use Math.Random() to simulate the choices of a second, computer-controlled player. There are ways to do the same with Strings, but they aren't necessarily as straightforward. The TL;DR version of this is that sometimes it is worth considering how more simplistic data types can be used to represent complex choices.

5. There are four sections of this code that you are required to complete in order to make this game functional. Examine the comment prompts inside the main method, and write the lines of code that would fulfill those prompts (HINT: If you get stuck, think back to previous sections of this lab and others to when you did similar things). When you think you have the code completed, run the program a few times to play a few rounds. Does everything work the way it should?

```
9a4a1204b0c51e1ede7524b/redhat.java/jdt_ws/ITSC1212_9f3e5672/bin" RPSGame
**************************  START  ********************************

Welcome to the RPS game! Are you ready?
Enter a number between 1 and 3. 1 is for Rock, 2 for Paper and 3 for Scissors
3
User chose Scissors
Computer chose Scissors.
It's a tie!

**************************   END   ********************************
```

6. Show your finished Rock-Paper-Scissors game to the instructor or a TA to receive credit for this section.