Lab 7: More Conditionals and Loops!

Let's get started!

Today we'd like you to:

1. **Open Eclipse**. Remember to keep all your projects in the same workspace. Think of a workspace as a shelf, and each project is a binder of stuff on the shelf. If you need some help organizing your workspace, we'd be happy to help!
2. **Create a new Java project**. Call it something that will help keep your code organized, i.e., COMP 1510 Lab 7.
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 7 project to quickly create a new Java class.
4. When you have completed the exercises, **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. **Remember that for full marks your code must be properly indented, fully commented, and free of Checkstyle complaints**. Remember to activate Checkstyle by right-clicking your project in the Package Explorer pane and selecting Checkstyle > Activate Checkstyle.

What will you DO in this lab?

In this lab, you will:
1. Repeat blocks of code using while, do-while, and the for loop
2. Make decisions using if, if-else, and switch
3. Create a guessing game and a rock, paper, scissors game
4. Create a responsive GUI that validates a password.

Table of Contents

1. Games

Now that we know how to create loops and make decisions we can start making games (pause for applause). Let's start with a pair of easy, fun games that use the Random class and luck:

1. Create a new class called **Games** inside package ca.bcit.comp1510.lab7:
   i. Do not include a **main method** inside the class definition. Remember the main method is what gets executed first when we start our program. In this case, the Games class is not a complete program.
   ii. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
      1. The name of the class and a (very) short description
      2. An @author tag followed by your name
      3. An @version tag followed by the version number.

2. The games we play will need the following elements. Create private instance variables to store each of them:
   i. user's score (an int)
   ii. a Scanner object to get input from the user
   iii. a Random object to generate random numbers

3. Create a constructor that:
   i. initializes the user score to 0
   ii. instantiates the Scanner object
   iii. instantiate the Random object

4. There's no main method. How do we play? Create a **Driver** class that contains a main method inside the same package. Don't forget to give it comments. In the main method, declare and instantiate a Games object, like this:

```
/**
 * Drives the program.
 * @param args unused
 */
public static void main(String[] args) {
    Games myGame = new Games();
    myGame.play();

}
```

5. **The Games class needs a public play method. Let's create one now. Create a public method called public void play() inside Games:**
   i. This method must continuously print the following menu:

```
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
>
```

   ii. After printing the menu, use the Scanner to get the user's choice:
      1. If the user chooses 1, print their score

2. If the user chooses 2 or 3, do nothing right now
3. If the user chooses 4, end the game (just return from the play method)

iii. If the user chooses anything else, tell them to make another choice.

iv. The following output is how your play() method should behave:

```
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 2
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 3
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 1
Your score is 0
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 5
That's not a valid choice!
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> -1
That's not a valid choice!
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 4
```

3 of 9

6. Let's make our first game. Modify the play() method so that when the user selects
   2, a method called **public void guessANumber()** is called. The guessANumber game
   is easy. The users must guess a number between 0 and 100 (inclusive). The game
   tells the user if their guess is too high or too low. The user can keep guessing until
   they get the right number. If the user can guess in fewer than five guesses, add five
   points to their score.

   i. Here is some sample output for the number guessing game. Remember: when
      the user finally guesses the number, if they have made the guess in five
      guesses or fewer, add five points to their score, otherwise there is no
      penalty:

   ```
   Welcome to the Games program!
   Make your choice (enter a number):
   1. See your score
   2. Guess a number
   3. Play Rock, Paper, Scissors
   4. Quit
   > 2
   I've picked a random number between 0 and 100
   Can you guess it?
   Guess the number!
   50
   Too high, guess again!
   Guess the number!
   25
   Too low, guess again!
   Guess the number!
   38
   Too high, guess again!
   Guess the number!
   31
   Too high, guess again!
   Guess the number!
   28
   RIGHT!
   Five points!
   Welcome to the Games program!
   Make your choice (enter a number):
   1. See your score
   2. Guess a number
   3. Play Rock, Paper, Scissors
   4. Quit
   >
   ```

7. Now it's time for our second (and final game): Rock, Paper, Scissors, a traditional
   North American child's game. Modify your play() method so that if the user selects
   3, a method called **public void rockPaperScissors()** is called. This method plays one
   round of Rock, Paper, Scissors:

   i. Use the random number generator to generate a 0, 1, or a 2. The zero will
      represent a Rock, 1 will represent Paper, and 2 will represent Scissors.

    ii. Ask the user to enter their choice: Rock, Paper, or Scissors. Wrap this in a loop
        so that if the user enters something else, they are prompted to try again.
        Your program should behave like this:

```
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 3
I've picked one of ROCK, PAPER, and SCISSORS
Which one do you choose?
> tomato
That's not a valid choice! Try again!
pencil
That's not a valid choice! Try again!
scissor
That's not a valid choice! Try again!
scissors
```

    iii. Once the user has made a valid choice, compare it to the computer's choice.
       The rules are:

        1. A rock smashes scissors, ties with another rock, but loses to paper.
        2. Paper wraps rock, ties with other paper, but loses to scissors.
        3. Scissors cut paper, tie with other scissors, but lose to rock.

    iv. If the user wins, tell them and add 5 points to their score
    v. If the user loses, tell them and take 3 points off their score

8. When we put it all together, we end up with a Driver program that instantiates a
Games object. The Games object contains a play method that keeps asking the user
to play a game or display their score until the user quits. The user can select a
number guessing game or a round of Rock, Paper, Scissors. Your game should
behave like this:

```
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 1
Your score is 0
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 2
I've picked a random number between 0 and 100
Can you guess it?
```

```
Guess the number!
50
Too low, guess again!
Guess the number!
75
Too high, guess again!
Guess the number!
58
Too high, guess again!
Guess the number!
54
Too low, guess again!
Guess the number!
56
Too high, guess again!
Guess the number!
55
RIGHT!
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 1
Your score is 0
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 3
I've picked one of ROCK, PAPER, and SCISSORS
Which one do you choose?
> sleep
That's not a valid choice! Try again!
#imsotired
That's not a valid choice! Try again!
rocks
That's not a valid choice! Try again!
rock
Nope, I picked Paper
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 1
Your score is -3
```

Bruce Link, Keith Tang, Christopher Thompson

```
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 3
I've picked one of ROCK, PAPER, and SCISSORS
Which one do you choose?
> paper
Nope, I picked Scissors
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 1
Your score is -6
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 3
I've picked one of ROCK, PAPER, and SCISSORS
Which one do you choose?
> iamawfulatthis
That's not a valid choice! Try again!
paper
Yes!  Paper wraps rock
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 1
Your score is -1
Welcome to the Games program!
Make your choice (enter a number):
1. See your score
2. Guess a number
3. Play Rock, Paper, Scissors
4. Quit
> 4
```

3. Finish with some GUI fun

Let's create a very simply GUI that validates a user's password. We will start with a simple version, and enhance it in future labs.

1. Create a new class called **PasswordValidator** inside package ca.bcit.comp1510.lab7:
   a. **REMEMBER!** In the new Java Class dialog box, enter javafx.application.Application as the superclass, or enter Application and then press the Browse button to select javafx.application.Application from the options provided.
   b. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
      i. The name of the class and a (very) short description
      ii. An @author tag followed by your name
      iii. An @version tag followed by the version number.
   c. Include a **main method** inside the class definition. Remember the main method is what gets executed first when we start our program. Include a Javadoc comment for the main method. The Javadoc comment should contain:
      i. A description of the method. In this case, "Drives the program." is a good idea.
      ii. An @param tag followed by args, which are unused.
2. After Eclipse generates your class, note its contents. We declared the 'superclass' to be the Application class from the JavaFX package. Eclipse automatically added an extra method called start (NEATO!).
3. Inside your main method you only need one line of code:
        launch(args);
4. Your PasswordValidator should work like this:
   a. Generate a GUI that contains two labeled TextFields for the user. The user will enter their password into each TextField.
   b. The GUI should include a button called Submit. When the button is pressed, an event handling method called public void validatePassword(ActionEvent event) should determine if the two versions of the password are the same.
   c. The GUI should print a message to the user that tells them if the two versions of the password are the same. That's it! Here's an example:
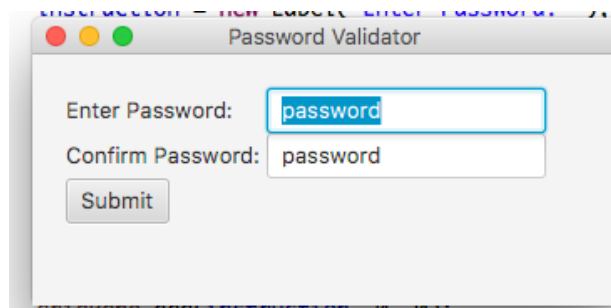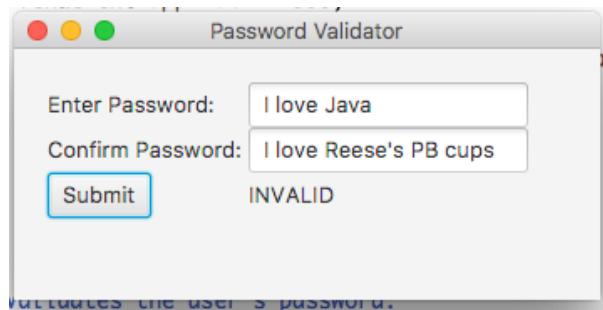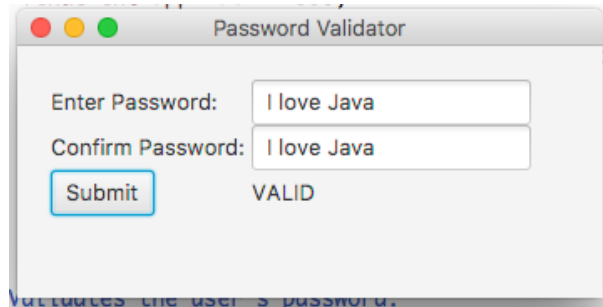

Figure 1 Starts like this

Figure 2 Invalid (not the same!)

Figure 3 Valid (the same!)

4. You're done! Show your lab instructor your work.