Objectives of this lab:

- Understand the importance of incremental development and testing your code often
- Utilize lightweight debugging techniques such as printing variable values and code tracing.
- Identify and correct syntax, runtime, and logical errors
- Understand random number generation

Pre-lab Reading Material:

You should review the material from your textbook below before coming to lab. If you have not done so, you will need to read it while the TA is explaining the lab, and you can read them after the lab to reinforce what you have learned.

- Types of errors: Section 1.11 (pp 26-28), 9th ed or Section 2.17 (pp 53-55), 8th ed.
- Random number generation: Section 3.8 (pp 96-97), 9th ed. or Section 3.9 (pp 82-83), 8th ed.

Exercise Preparation:

- Start a terminal and follow the next steps by typing the appropriate commands:
  - Create the lab7 directory inside your cs170 directory
    `mkdir ~/cs170/lab7`
  - Copy the files you will need
    `cp ~/cs170001/share/lab7/* ~/cs170/lab7`
  - Move into the lab7 directory (ie make it the working directory)
    `cd ~/cs170/lab7`
  - List the contents of the current directory
    `ls`
  - You should see 2 files: `SumPractice.java` and `Guess.java`. If you do not see these files, ask the TA for help.

Task 1: Find three types of errors in `SumPractice.java`.

- The `SumPractice.java` Java contains several errors. Specifically, it contains 2 syntax errors, a runtime error and a logic error. Your task is to find and correct these errors.

- Open the terminal, change into your lab7 directory, and open the `SumPractice.java` in gedit by typing:
  `cd ~/lab7`
  `gedit SumPractice.java &`
- Compile the Convert.java program, execute this command in a terminal window:
  `javac SumPractice.java`
- The file will not compile due to syntax errors.
- Fix the syntax errors in the file by looking at the line numbers in your program that the compiler specifies. Remember to always begin with the first error the compiler gives. (You may have to scroll upwards in the terminal window to see the first error.)
• Once your program compiles, run it. Now you will see a runtime error. That is an error which only occurs when the program is run and can't be identified by the compiler (like the syntax errors you found above).

• Carefully read the error message and look at the code. Try to identify the problem in the code. Remember that small details such as capitalization and punctuation matter a great deal in code. If you can't identify the source of the runtime error, ask your TA for help.

• After you fix the runtime error, recompile the program and try running it again.

• Success! Or is it? The program SHOULD compute the sum of the first n odd integers. Does the program give you the correct answer? Double check since you should NEVER assume the program is giving you the correct answer just because it runs. Indeed, you should know what answer you expect the program to compute before you run it. Otherwise, how will you know if your program is correct?

• Try running the program with different inputs. If you notice a pattern, this will give you a clue to the logic error.

• Debugging would be a lot easier if we had some intermediate information to work with and not just the final answer. So let's add a statement to print out some of those intermediate results, every time the loop executes. Add the statement

\[
\text{System.out.println("Adding } + \text{ odd } + \text{ to my current sum } + \text{ sum");}
\]

as the first line of code inside the while loop.

• Compile and run your program again. This statement prints out some very useful information. Printing out intermediate results inside of a loop can be a very useful strategy for finding logic errors in your code. You can add more println statements if you think other information would be useful.

**Task 2: A more complicated debugging exercise in Guess.java**

• This exercise is a great example of why you should never write a lot of code before you test and debug. It's much harder to debug this program now that it's “finished” then it would be if you practiced incremental development and fixed the errors as you went along.

• As above, open the Guess.java file in gedit. Then compile it with javac. This file contains only logic errors.

• This program emulates a guessing game. Here's a description of how the game is SUPPOSED to work:

The game plays five rounds and during each round the user is given three chances to guess a number between 1 and 6 inclusive. (You can think of this as rolling a 6 sided die.) During each round, the program generates a random number in the same way we did above. (The equivalent of rolling the die once.) If the user guesses correctly, then s/he wins a point, otherwise the computer wins a point. At the end of five rounds, whoever has the most points wins the game. However, the program has bugs, and it doesn't follow the rules currently.

• Your job is to make the game (i.e. program) follow the above rules by finding and correcting bugs.

• Before looking at the code, run the Guess program to see what it does. It is vital when
debugging a program, to collect evidence before making ANY changes to the code. This helps you build a mental model of how the code works and gives you a starting point to look for errors. Sometimes, you can identify errors without much effort by carefully tracing the code with the input sequence that causes an error.

• **An important aside:** For your prelab reading, you read about using the `Math.random()` function to generate random numbers. The code in this program is similar, but more powerful and is capable of generating random integers easily. However, because the code generates a different number each time (what fun would it be to play a game that picked the same number every time?), it's hard to debug. To make the program's behavior repeatable (i.e. give us the same random number result each time, we have to use something called a **seed** when we create our random number generator. To make the code's behavior repeatable, change the code that constructs the generator to include a number as shown below:

```
Random ranGen = new Random(11);  // 11 is a seed value; you could use other numbers if you want.
```

• After you have identified one (or more) incorrect behaviors, take a look at the **Guess** class code. You should put in println statements like we did above to help you identify issues as you begin to try to fix the code.

  ◦ It is often useful to identify a loop or if-statement that isn't working the way you think it should. Read the code and identify which loop or if statement you think is incorrect. (Wouldn't it be nice if this code were commented? This is why we make you comment the code you write.)

  ◦ Print out the variables that control the loop or if-statement (i.e., print the variable in its condition) immediately before the loop/if-statement executes (or doesn't execute).

  ◦ This gives you an idea of exactly why your loop/if-statement is executing or not. You can then alter the variable to make the behavior correct. After making a correction or adding println statements, run the program again and observe the behavior.

  ◦ Once your loop or if statement is working correctly, comment the println statements out.

  ◦ Are there other problems? **When debugging it is important to focus on one problem at a time.** Once you are satisfied you've corrected that problem, then run the program again to identify the next problem.

  ◦ Continue in this way until you are satisfied that the program plays by the rules. As you correct the code, it is a good idea to add a comment saying what you've changed (and why). This helps you keep track of the changes, which is particularly useful if you need to undo a change later if you discover that it wasn't the cause of the problem.

• Are you sure you have found **ALL** of the errors in the program? You need to run the program multiple times and try different scenarios. Think of all the different ways a player could play the game and emulate those. For example, does your program run correctly if the player loses all the rounds? What about if they win all the rounds? If they win the first round but lose subsequent rounds? If they lose the first round and win subsequent rounds? Your program should handle **ALL** of these scenarios correctly. It is NOT enough to simply test your program with one configuration and declare it correct.

• You should be able to identify 3 errors. (Note that failure to follow programming conventions isn't an error. It's bad form, but not an error.) However, you will have to add more than just 3
lines of code to fix these 3 errors. However, all the code changes required are minor. (In other words, you will not have to make major changes like changing levels of code.)

- Debugging is often a slow process which can be prone to bugs itself. Sometimes you will make a change but later have to undo it. Go slow, work on one problem at a time, and add println statements to help you identify the issues. Debugging is methodical and much easier when you spend more time THINKING about the code rather than randomly changing variables hoping that you will get the correct behavior.

**Turning in your work:**
- When you are done, turn-in lab 7 using these commands:
  
  ```
  /home/cs170xxx/turnin SumPractice.java lab7a
  /home/cs170xxx/turnin Guess.java lab7b
  ```

  
  ◦ *Note: you will need to replace ‘xxx’ with your section number.*