ITSC 1212 Module 8 Lab

In this lab, we will be combining two of the concepts from your prepwork (number finding and Turtle drawing) to create a sort of game. If you haven't completed that work yet, it would be a good idea to take a few minutes and review those tasks before starting the lab.

Concepts covered in this lab:

- For and While loops
- Method calling
- Booleans

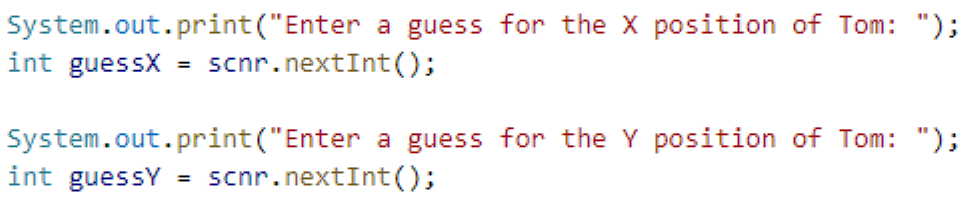
Required files or links

- ITSC1212Lab8.java
- colorGuess.txt

Part A: Designing the base game

1. The goal of the game we're going to create in the lab is to create a World and drop an invisible Turtle somewhere in that World. The player's goal is to guess the Turtle's X/Y coordinates until they have successfully located the invisible Turtle.
2. Start by downloading the ITSC1212Lab8.java document. Copy it into your VSCode project folder and open it in VSCode. Take a moment and review the contents of this file. Notice that above the main method we're creating two fields: randomX and randomY that will eventually hold random numbers, and a World object. We're creating those variables outside of the main method so that we can access those variables in additional methods later in the lab. That topic will be covered in more detail in a later unit.
3. Notice that both of our random number variables are being initialized to 0 in the code that you have. These are the variables that will determine the random position of the Turtle and will be different each time the game is played. It's important to consider that the play space of this World is 500x500, meaning there are 250,000 possible positions for our invisible Turtle. To make things easier, let's limit these possible positions to only multiples of 5. Take some time to think about how you might use Math.random() as you have in past assignments to generate a random multiple of 5 from 0-500. If you get stuck here for more than a few minutes, skip to the bottom of this document for the solution (please do actually attempt this though, it's really good practice). Use that formula to change the randomX and randomY initial values to something other than zero.
4. Now that we have our World and our random positions, we can advance to the main method and actually create our game.

5. The first thing we need to do here is actually create the Turtle we're going to be working with. Start by creating a Turtle, as well as a Scanner object with which we can take user input.
6. Use the Turtle method .setVisible() to set your Turtle's visibility to **false** so it's invisible.
7. We now have an invisible Turtle at some unknown location inside our World. The next step is going to be to collect the first guess from the user. Add the following lines:



```
System.out.print("Enter a guess for the X position of Tom: ");
int guessX = scnr.nextInt();

System.out.print("Enter a guess for the Y position of Tom: ");
int guessY = scnr.nextInt();
```

8. Let's take a moment and evaluate all the different pieces we have. We have a World, a Turtle at an unknown location inside that world, two variables that contain the X/Y position of the Turtle, and the user's guesses for that X/Y.
9. The next bit is up to you and will be a good challenge of your ability to write both a while loop, as well as your understanding of boolean expressions.
10. Translate the following pseudo code into Java (the indentation of this is a hint about how to structure your code).

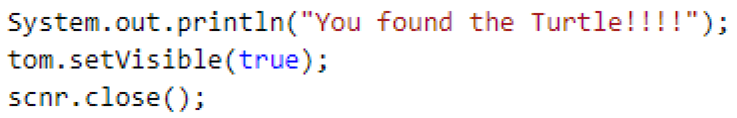
```
While guessX does not equal randomX OR guessY does not equal randomY
    If guessX was too high
        Inform the player to guess lower next time
    Alternatively, if it was too low
            Inform the player to guess higher next time
    If guessY was too high
        Inform the player to guess lower next time
    Alternatively, if it was too low
            Inform the player to guess higher next time
    If guessX was wrong
        Prompt the user for a new guess of the X value
        Save that to guessX
    If guessY was wrong
        Prompt the user for a new guess of the Y value
        Save that to guessY
```

When you are satisfied with how your loop is written (we will test it later), continue on.

11. The beauty of structuring our code this way is that it will continue to ask the user for guesses until both the X and Y are correct, and once one is correct it will only ask for new input for the value that is still incorrect. This while loop will ONLY break once the Turtle has been found.
12. Be prepared to answer the following questions:
    a. Why is a while loop more appropriate here than a for loop?

    b. Why do we use OR in the while loop instead of AND?

13. Because we know that once the while loop has broken that the Turtle has been found, we can add in our "win" conditions. Specifically, what we want to have happen is the User be notified of their correct guesses, and the invisible Turtle should appear. Add the following lines to do this.



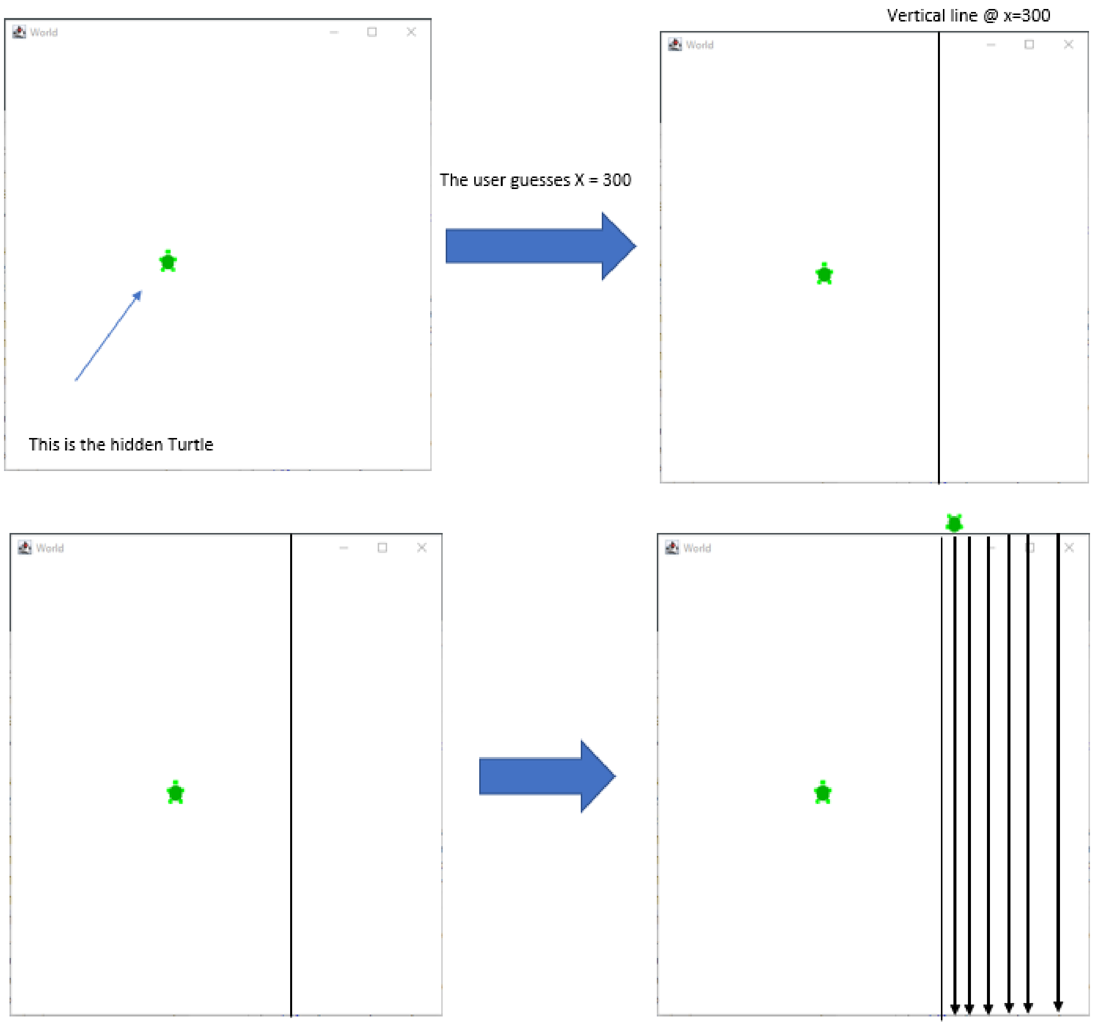
```
System.out.println("You found the Turtle!!!!");
tom.setVisible(true);
scnr.close();
```

    Feel free to add any kind of formatting or additional print statements to really jazz up this win message.
14. Now that the basic pieces of this code are all in place, we need to test it so we can tell if your while loop and the statements within it were written correctly.
15. Before the code starts guessing Tom's coordinates, print the correct coordinates of the Turtle. This must happen before your while loop and NOT inside or after it. It's important to know the correct location of the Turtle during this testing phase so we can test different scenarios.
16. Test the following scenarios:
    a. If the guess is too low, is the user instructed to guess higher next time?
    b. If the guess too high, is the user instructed to guess lower next time?
    c. If one guess is correct, is asking for that value skipped in the next round?
    d. If both guesses are correct, is the win message displayed?
17. When you are satisfied that this game is working correctly, show your code to a TA or instructor to receive credit for this section.

Part B: Add in color blocks

ITSC 1212 Module 8 Lab

1. One of the issues with this game is that it doesn't feel very responsive. Part of the problem is that we have this World object but we're not really doing anything with it. What we could do in this situation is use the World to visually "block off" areas where we know the Turtle can't be anymore based on previous guesses.
2. Open the file **colorGuesses.txt** and add the contents BELOW (but not inside) your main method. Most of this code is already written for you, but you have four FOR loops that must be corrected. Before you jump into that, take a moment, and make sure you understand what this code is attempting to do: it blocks off areas where the hidden Turtle cannot be by using a second invisible Turtle to draw a series of black lines in those areas. Here's a visual breakdown of this process describing what happens when an X guess is too high.



X=300 was too high, meaning we know anything from 300-500 is a wrong guess

Our second invisible Turtle draws a series of vertical lines, starting at x=300, moving one pixel over each time, until x=499. The lines are 1 pixel thick, so the Turtle must draw a line for each x position to the right of the initial incorrect guess.

3. What this does is black out the area of the World from x = 300 to the right side of the World. The same logic must be applied to the Y coordinate. The code you've been given will accomplish this IF you write the FOR loops correctly: one draws lines at all coordinates greater than the user's guess if the user guessed too high, the other draws lines at all coordinates smaller than the user's guess if the user guessed too low. Your job is to fill in the code between the parentheses in each of these four FOR loops. Note that you will need to add the import for the Color class.

```
import java.awt.Color;
```

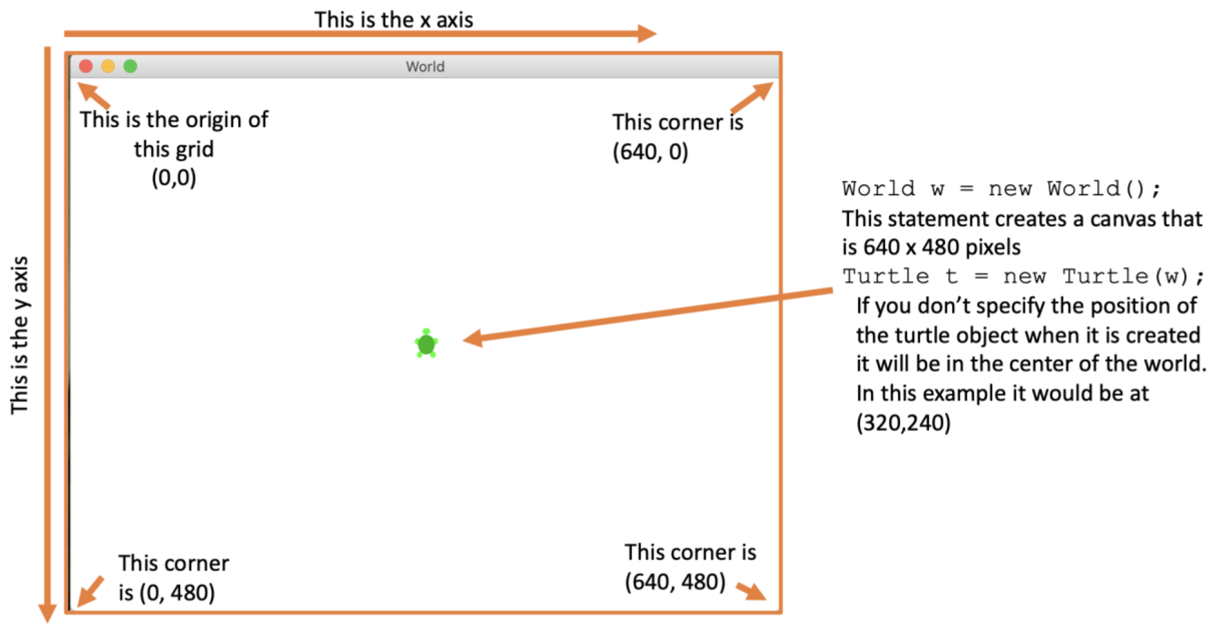
4. The last thing to add after you've written the logic for the for loops is to call these methods. You will want to add these methods calls below or above the print statements telling the user of their incorrect guess, like so:

```
System.out.println("Your guess for X was too low! Guess higher!");
colorGuessX(guessX);
```

5. Writing the correct for loops to make this process work will likely take some trial and error. Test your code frequently, making sure that the correct areas are blocked off each time.

****Important note****

The point 0,0 in the World is the top left corner of the box. Intuitively, we want to place that at the bottom left, because we're used to that from math classes. Also remember that a "box" 500 pixels wide would have X-coordinates in the range of 0 to 499, not 1 to 500.

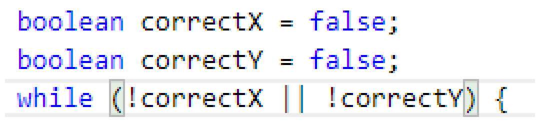


```
World w = new World();
```
This statement creates a canvas that is 640 x 480 pixels

```
Turtle t = new Turtle(w);
```
If you don't specify the position of the turtle object when it is created it will be in the center of the world. In this example it would be at (320,240)

Part C: Add a new game mode

1. In this section, what you will do is give the player the option to play in a “Close-enough/Easy” mode. The goal of this mode is:
   a. Allow the user to select their desired game mode
   b. If the user has selected the Easy mode, any guesses within 20 pixels of the correct X or Y position will be counted as correct
2. Think about the flow of the game, and what, if anything, would need to change at each step if the Easy mode were to be selected. We want you to think about how these two modes could live side by side, rather than just copy and pasting the first version and altering it.
3. Here are some helpful hints we think will be helpful for how to make this work:
   a. Add boolean variables to track whether or not a guess is correct.
   The reason you may find this helpful is because the two game modes have different criteria for what a “correct” guess is. Your code, at this stage, is likely using a guessX == randomX to test correctness, and obviously this doesn’t work for the Easy mode. By using boolean variables, you can have two distinct ways of establishing what a correct answer is and store that result based on which game mode is being played. You can also use these booleans to control your while loop, which vastly simplifies the way that loop looks.



```
boolean correctX = false;
boolean correctY = false;
while (!correctX || !correctY) {
```

   This also allows us to move the code that controls the first guess INSIDE the while loop, since we start the loop with the assumption that the correct answer is not given.
   b. Leave the code that says a guess was too high or too low alone.
   Altering the logic of this part of the loop is … a lot. Focus on the aspects that determine if an answer was right or not.
   c. Think about how you can alter the logic where a correct guess was established.
   Towards the end of your loop, you have a section that was written based on this pseudocode:
   Consider this partial section of the pseudocode from Part A:

```
If guessY does not equal randomY
    Prompt the user for a new guess of the Y value
    Save that to guessY
    If guessY is correct
        Inform the player of that
Or, If guessY was too high
    Inform the player of that.
Alternatively, if it was too low
    Inform the player of that.
Prompt the user for a new guess and save it
```

Alter this so that the logic is now:

```
    If guessY does not equal randomY
        Prompt the user for a new guess of the Y value
        Save that to guessY
        If guessY is correct Or if Easy Mode is on and guessY is within 20 pixels
        of the correct location
            Inform the player of that
            Update the value of the boolean correctY
        Or, If guessY was too high
            Inform the player of that.
        Alternatively, if it was too low
            Inform the player of that.
```

d. Once all this works, consider other rearrangements you could make.

4. Once you're satisfied with your Easy mode, show your work to an instructor or TA to receive credit.

Solution to Random Number Multiples of 5

```
((int)(Math.random() * 100)) * 5;
```