ITSC 1212 Module 9 Lab - TurtleBoards

In this lab, we'll be exploring the uses and formatting of nested loops. Nested loops are incredibly important and extremely useful, but need to be used carefully.

Concepts covered in this lab:

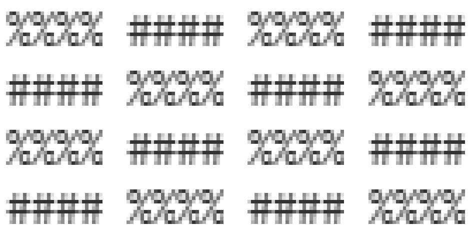
- Nested loops

Required files or links

- drawSquare.txt
- TurtleCheckerboard.java

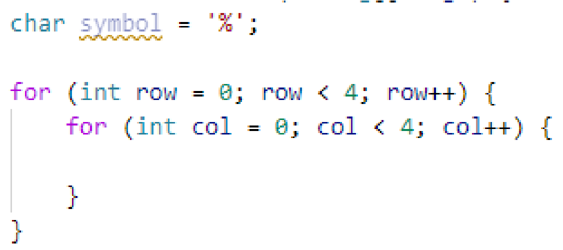
Part A: Building a text-based checkerboard

1. A nested loop is, generally speaking, used for a data or problem that is multidimensional. Imagine for instance that we wanted to scan through a photo and manipulate the pixels in that photo. To scan through each pixel, we need to look at every pixel in a row. That task is two dimensional, and a great example of why nested loops can be so powerful: it allows us to repeat a task over a large problem set with minimal lines of code.
2. To start to demonstrate this, let's think about how one might build a text-based checkerboard or grid. Something that looks like this:



```
%%%% #### %%%% ####
#### %%%% #### %%%%
%%%% #### %%%% ####
#### %%%% #### %%%%
```

3. To get started, let's assess what we know about this board
   a. It has 4 rows, each row has 16 characters in four groups of four characters each
   b. Each row alternates characters every four instances of a character
   c. Will likely use a combination of print and println
4. It's fairly safe to assume we're going to be using some nested loops to design this, but we also need some sort of mechanism to to switch the character after each block.
5. Now that we've defined the problem, it's time to get started. In your VSCode 1212 project folder create a new Java file, **ITSC1212Lab9.java**, and add a main method. Inside the main method, add the following code:

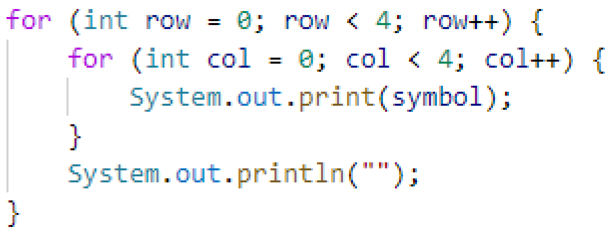


```
char symbol = '%';

for (int row = 0; row < 4; row++) {
    for (int col = 0; col < 4; col++) {

    }
}
```

6. Take a look at this code and evaluate what we have. Specifically, we have a char variable to hold our symbol, and a nested for loop that will do much of the work for us. Now, the important thing to remember is that with these loops, for every single iteration of the “row” loop, we get the full set of iterations of the “col” (column) loop. If you’re doing the math in your head, you might have realized that 4x4 is only sixteen, which is not enough characters to fill out our board. That’s okay for now. It’s often a good idea to solve a simpler version of a problem first, then alter it to suit the larger objective.

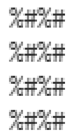
This is an example of using variable names that help the reader understand how the code is being used. We could use any legal variable name – “i” and “j” are typical – but a variable name that indicates what’s going on is sometimes better. “row” will refer to the row of our output and “col” will refer to the column.

7. We have two things to consider at this point: first, how do we format our print statements so that we can print one symbol at a time, but move to a new line after a row is completed; and secondly, how do we switch characters?
8. The good news is that both of these are relatively straightforward, but let’s tackle the first in this step. In order to print without moving to a new line each time, we use “System.out.print”, and to move to a new line, we use “System.out.println”. So, in terms of our needs, we would want to use the plain “print” while we’re creating a row, and move to a new line after that row is completed. We can do that like so:



```java
for (int row = 0; row < 4; row++) {
    for (int col = 0; col < 4; col++) {
        System.out.print(symbol);
    }
    System.out.println("");
}
```

9. If you “read” this code, it would say, “For each row, and for each column in that row, do ___”. These two loops give us a way of accessing each element of our grid, and placing a symbol in that element. Now we need to establish the symbol switching.
10. What we mean by “symbol switching” is that after we’ve printed one four-character group of symbols, we need to switch to the other symbol: from % to # or vice versa. You should know that we will only need a simple if/else, so go ahead and write that. If you’ve done that correctly, when you run your code you should see this:



```
%#%#
%#%#
%#%#
%#%#
```

11. Now obviously this is not our desired result, but it does help us establish our pattern. In fact, if you think about it, the ONLY real difference here between what we have and what we want is that this only prints the symbol once, whereas we want to print it four

times. Now if only we had just spent the last week practicing a simple programming concept that would allow us to repeat a certain action a given number of times…

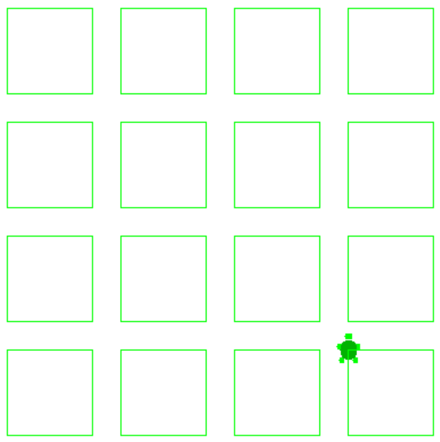
12. Add in the last remaining loop. That's right, we're going to have a three-nested loop structure to create our final board. It's important to understand the logic of this structure. The "row" (or outer) loop controls how many rows are printed – four. The "col" (or inner) loop controls what gets printed in each row. But we need something else because we don't want to print just one symbol at a time, we want to print four at a time. That sounds like another loop, right? Add that new loop now.
13. Run your code. You might notice two differences between what you have and the example, repeated below this step. Make those two final additions, and when your output looks like the example below, show an instructor or TA to receive credit for this section.

```
%%%% #### %%%% ####
#### %%%% #### %%%%
%%%% #### %%%% ####
#### %%%% #### %%%%
```

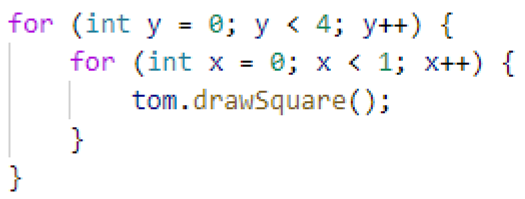
The purpose of this section is for you to get used to using nested loops and controlling output with them.

Part B: Making a checkerboard with Turtles

1. Start by downloading **TurtleCheckboard.java** and adding it to your VSCode 1212 project folder. Open this file and take a close look at the contents, as there are hints for what you need to do later in this section.
2. Our goal with this code is to use a Turtle object, tom, to produce a set of squares in a checkerboard pattern in a World using nested loops similar to what we did in Part A. The end result should look something like this:



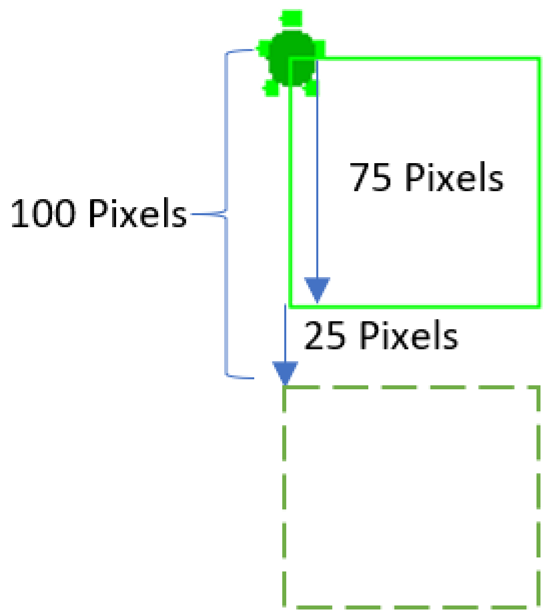
3. Copy the text from drawSquare.txt into your **Turtle.java** file. (Remember that file is in the Java Swing Turtle folder. Also, be sure to copy the text above the curly brace that ends the class but below the curly brace that ends the last method in the class.) This is a method we can call to have our Turtle object tom draw a square. The main method code in TurtleCheckerboard has tom positioned at (50, 50) in the World, is facing up (towards the top of the World), and has the pen down. Add a call to the drawSquare method. Your Turtle tom starts at its current position of (50, 50) which is the upper left of the square, draws the square, and ends in the place where it started, facing up. Remember that every time you call this method your Turtle object will perform the actions described in the method. With this knowledge, to draw multiple squares all you need to do is put your turtle object in the right location, call the method, and a square will be drawn there.
4. As we did in Part A, we'll have an easier time if we focus on a smaller part of the problem first, and then expand from there. What we can do is focus on building properly spaced squares in one dimension using one loop, then expand that into a second dimension using a different loop. To start, let's focus on doing one square per row to start. Before we do that, delete the call to drawSquare. We'll add it inside the loops we're going to build.
5. Below the code that you already have in the main method of your TurtleCheckerboard class, add the following code. We're going to use variables named x and y to indicate the x-direction (horizontally) and y-direction (vertically of the World:



```
for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 1; x++) {
        tom.drawSquare();
    }
}
```

6. Before you run this, let's also establish some goals and rules:
   a. The square we will draw will have sides that are 75 pixels long.
   b. The squares should be spaced 25 pixels apart both horizontally and vertically.
   c. We want 16 total squares.
7. Run your code at this point. Notice that although we are telling our Turtle object to draw a square 4 total times (once per row), we only get one. That's because each square is just being drawn on top of the previous one.
8. What this means for us is that after one row is drawn, our Turtle object must be moved to the start of the next row. Taking into account the size of the square and the required gap between squares, it means Tom must move down a total of 100 pixels:

ITSC 1212 Module 9 Lab - TurtleBoards



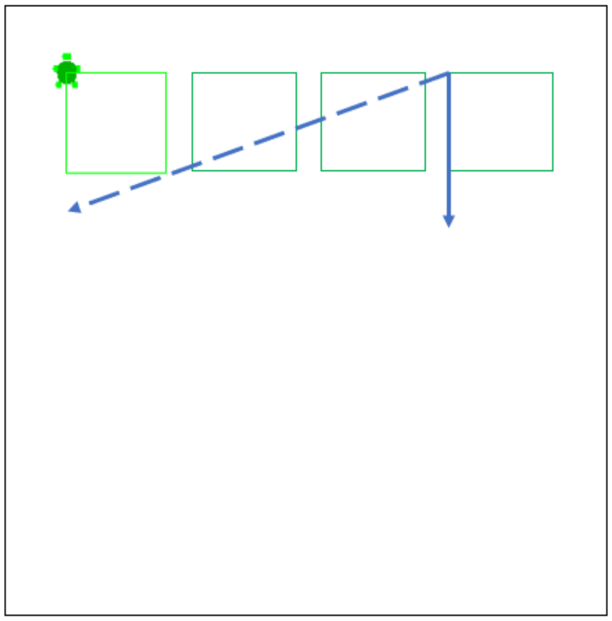
(The same is will be true of the horizonal spacing: columns of squares are separated from each other by gaps of 25 pixels.)

9. There are several ways to move Tom down 100 pixels, but let's start by demonstrating a "wrong" way (for what it's worth, this isn't so much wrong as it will introduce other problems later). Modify your code so it looks like this:

```
for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 1; x++) {
        harry.drawSquare();
    }
    harry.turn(180);
    harry.penUp();
    harry.forward(100);
    harry.turn(180);
    harry.penDown();
}
```

10. Run your code. What is happening is that after the row is completed, our Turtle object is moved down to where the next row should start. You should correctly see a set of nicely spaced squares along the let's consider what's going to happen when we start filling out the rows so more than one square is drawn in each row:

11. If our Turtle object is moved straight down after the last square of the row is drawn, the



place we'll eventually WANT that object to be is indicated by the dashed line above, but it will actually go where the solid line is pointing; directly below the last square in a row. Obviously this is a bad situation, as it means each subsequent row will get further and further out of alignment.
12. You could add another couple of lines of code to move Tom back to the left 300 pixels before or after moving him down, but at this point I would argue that our solution is quickly becoming a complete mess and that we should explore other options.
13. One solution is to use the moveTo(x,y) method of Turtles to move the Turtle to the specified X/Y position without having to use any combination of turns and forwards. All we need to do is come up with some way of identifying the X/Y position where the next row would need to start.

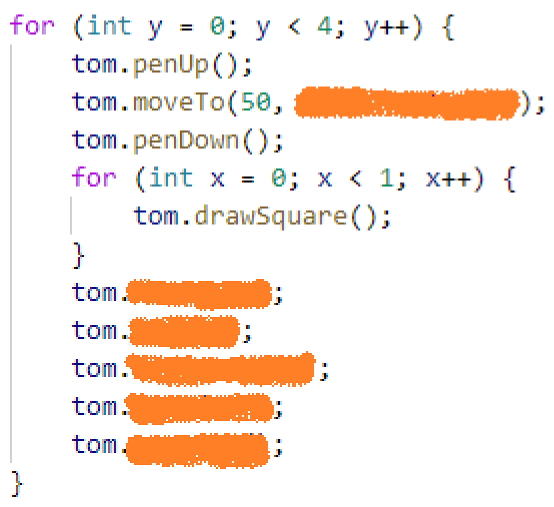
A very important concept for you to start thinking about involves relationships that emerge as you study a problem and start to devise a solution. Often, a significant part of finding a solution to a coding problem involves recognizing patterns in the data. If you can recognize a pattern then you can use it to help craft your code.

14. When you think about the starting coordinates for each row you'll realize that half of this information is always going to be the same: the X half of this position for the beginning of each row is always going to be the same. Since we started our first square at (50, 50), the X coordinate of each row will always be 50. What is going to change for each new

row is just the Y position. Keeping in mind that we were moving 100 pixels down each time (increasing the Y value by 100), fill in the following chart:

Square number	Value of y in the loop that controls drawing rows	X/Y position @ start of row
1	0	(50 , 50)
2	1	(50 ,)
3	2	(50 ,)
4	3	(50 ,)

15. Take a look at these values, and the value of the y variable in the loop. What is the pattern that determines where tom needs to be moved at each step of the "y" loop? In other words, if I tell you the value of the y variable, can you tell me the value of the Y coordinate of the moveTo method call? It might be helpful to draw a picture of this in your sketchbook and label the upper left corner of each square so you know the X/Y coordinates. You already know how big a square is and how far apart horizontally and vertically each square should be separated from the others. Remember: you're looking for a pattern in the relationship between some of the variables.
16. Modify your y loop to use the moveTo method so that tom starts at the correct position BEFORE the squares in a row are drawn. Then, modify the "x" loop to draw four squares per row instead of one. Finally, make the Turtle invisible so all you see is the World with a checkierboard of squares in it. Here's a hint about how your code might look at one point:



```
for (int y = 0; y < 4; y++) {
    tom.penUp();
    tom.moveTo(50, [redacted]);
    tom.penDown();
    for (int x = 0; x < 1; x++) {
        tom.drawSquare();
    }
    tom.[redacted];
    tom.[redacted];
    tom.[redacted];
    tom.[redacted];
    tom.[redacted];
}
```

17. When you are satisfied with your solution, show your checkerboard and the code to an instructor or TA to receive credit for this section.

ITSC 1212 Module 9 Lab - TurtleBoards

Part C: Bonus #1

1. In your ITSC1212Lab9 class, copy the Part A code and paste it below the Part B code. Add a comment that identifies it as Part C. Modify the code such that:
   a. There are three new int variables: numRows, numCols, and numSymbols
   b. These three ints determine the number of rows, number of columns, and number of symbols in each “block”, respectively. This should make your code highly customizable. Continue to use the ‘#’ and ‘%’ symbols in your “blocks.”
   c. Make sure you consider the implications of having odd numbered dimensions such that symbols alternate correctly on both rows and columns.

Part D: Bonus #2

1. In your TurtleCheckerboard class, copy your solution to Part B and paste it at the bottom of your main method. Add a comment that identifies it as Part D. In a new World, modify that code such that:
   a. The only line that moves your Turtle in any way (except .drawSquare) is a SINGLE usage of moveTo().
   b. This means no using .turn() or .forward(), but you may still move the pen up and down.