CMPINF0401 Recitation

TUESDAYS 11:00-12:50

MICHAEL BARTLETT

Overview

- Finalized Office Hours Location
- Loops
  - While Loop
  - Do-While Loop
  - For Loop
- Lab 4

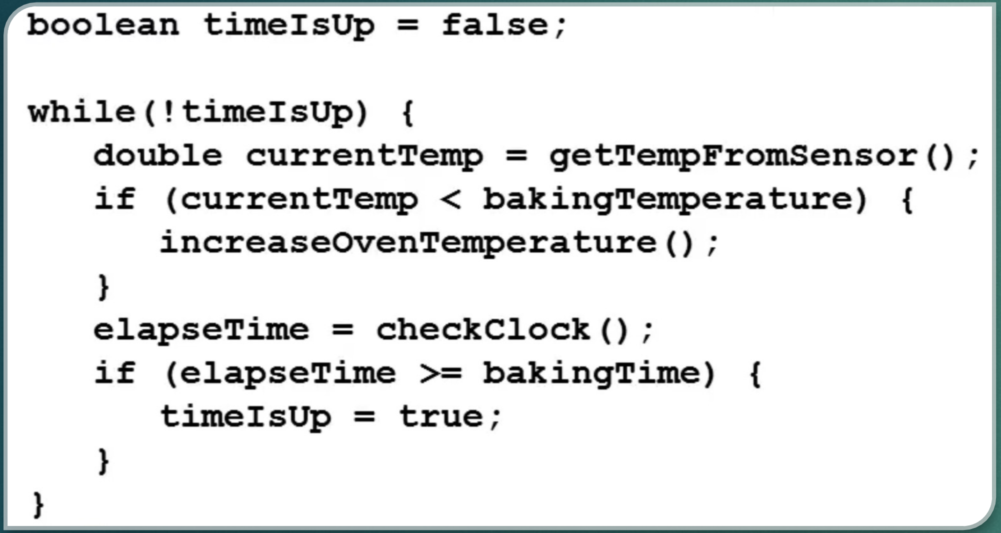
Finalized Office Hours Location

- 5806 Sennot Sq.
  - 11:30-1 on Wednesday
  - 12:30-2 on Thursday

Loops

- Exactly what it sounds like!
- Avoids the need to write the same block of code over and over again
- Loops will check to see if a condition is true and execute the code inside of it any number of times depending on whether the condition stays true
  - Almost like an if statement if it ran over and over…
- Three types
  - While loop
  - Do-while loop
  - For loop

While Loops

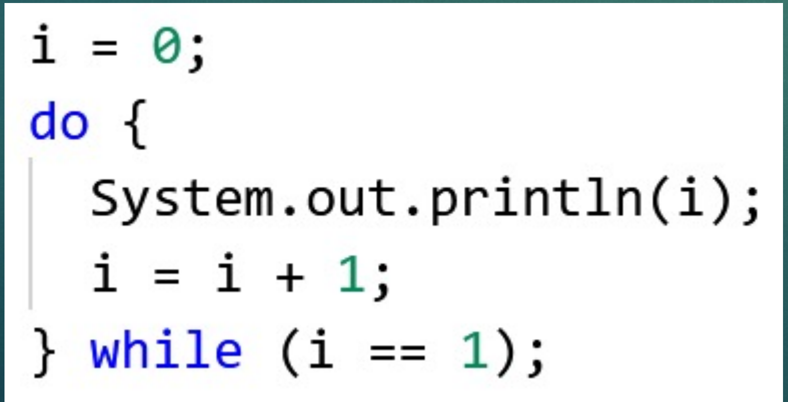


```
boolean timeIsUp = false;

while(!timeIsUp) {
   double currentTemp = getTempFromSensor();
   if (currentTemp < bakingTemperature) {
      increaseOvenTemperature();
   }
   elapseTime = checkClock();
   if (elapseTime >= bakingTime) {
      timeIsUp = true;
   }
}
```

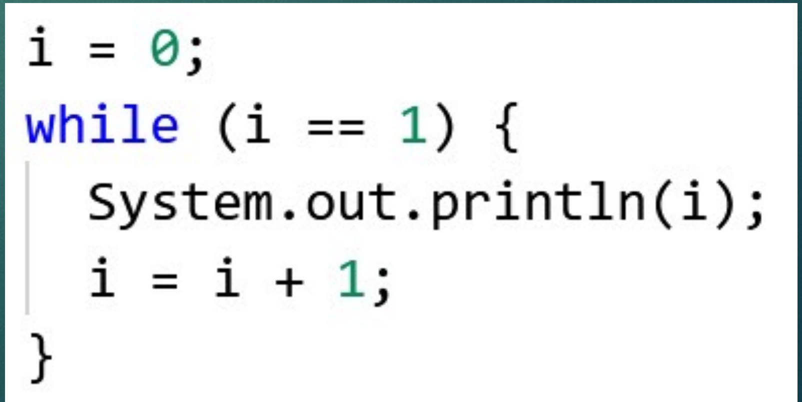
- In plain English: “While this condition is true, do this action”
- To unpack this example
  - Our condition is checking whether the time on the oven is up or not
  - **While** it isn’t, we elapse time and check to see if the elapsed time is equal to the baking time
    - If they’re equal, then we set timeIsUp to true.
    - The loop will stop when it goes back to check whether the while clause is true; since timeIsUp is true, !timeIsUp will be false, so we exit the loop

Do-While Loops

- In plain English: "Do this action, and while the condition is true, do it again"
- Essentially… the same as a while loop, except the code inside of the loop always runs **at least once**
- The loop on the left will print the numbers 0 and 1 (because it will always run at least once, and the second time it runs, the condition at the end is true), but the loop on the right will print nothing because the condition at the beginning is false

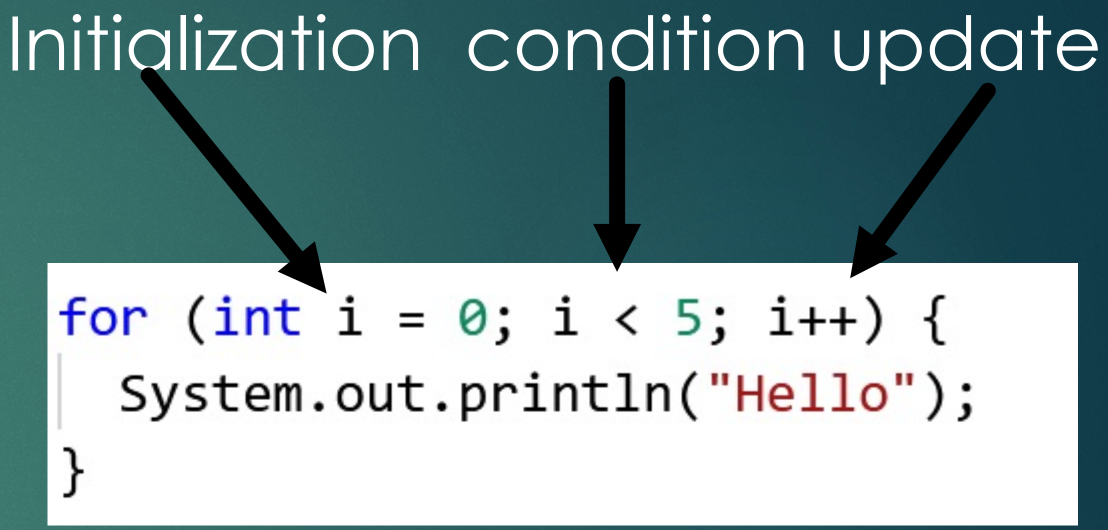


```
i = 0;
do {
  System.out.println(i);
  i = i + 1;
} while (i == 1);
```



```
i = 0;
while (i == 1) {
  System.out.println(i);
  i = i + 1;
}
```

For Loops

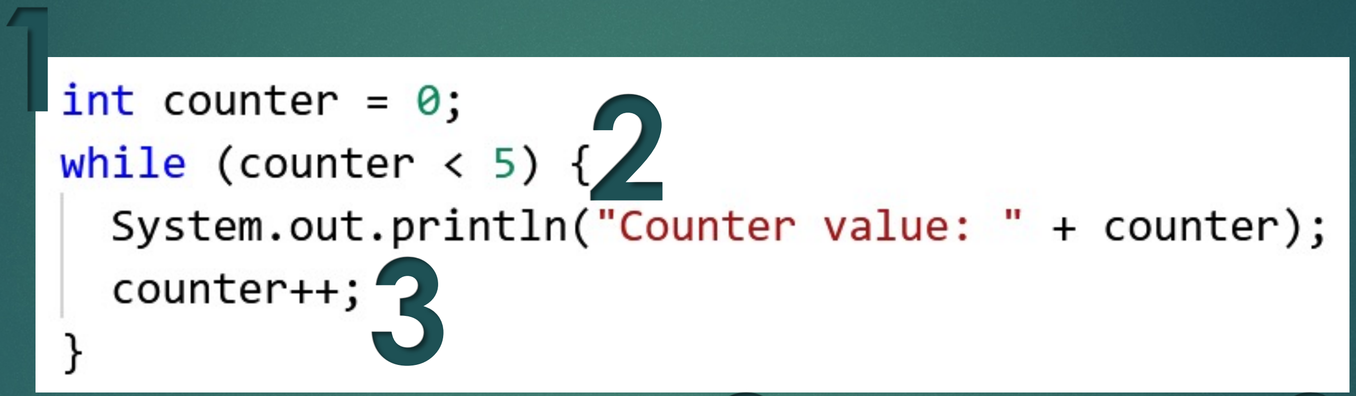


```
for (int i = 0; i < 5; i++) {
  System.out.println("Hello");
}
```

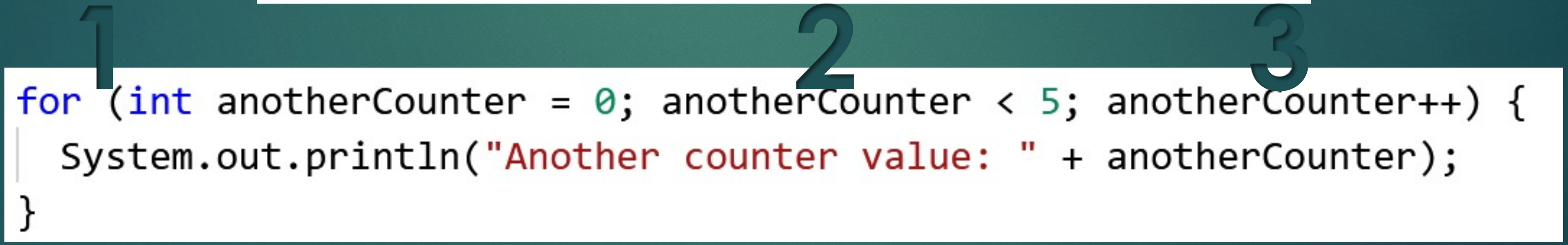
- There are three clauses in a for loop:
  - Initialization
  - Condition
  - Update
- The convention when initializing a variable is to use the letter `i` as your variable name – it's essentially the only time you should be using single-letter variables!

For Loops

- A bit harder to translate into plain English, so let’s compare to a while loop



```
int counter = 0;
while (counter < 5) {
  System.out.println("Counter value: " + counter);
  counter++;
}
```



```
for (int anotherCounter = 0; anotherCounter < 5; anotherCounter++) {
  System.out.println("Another counter value: " + anotherCounter);
}
```

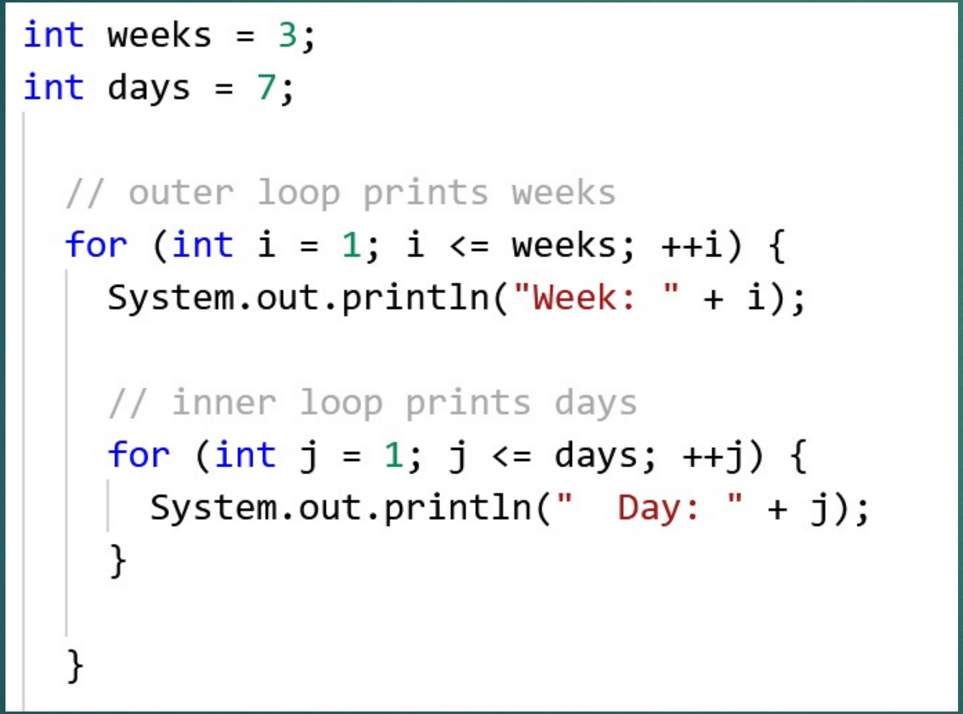
- Initialize a variable (1), create a Boolean condition (2), increment the variable (3)

Nested Loops

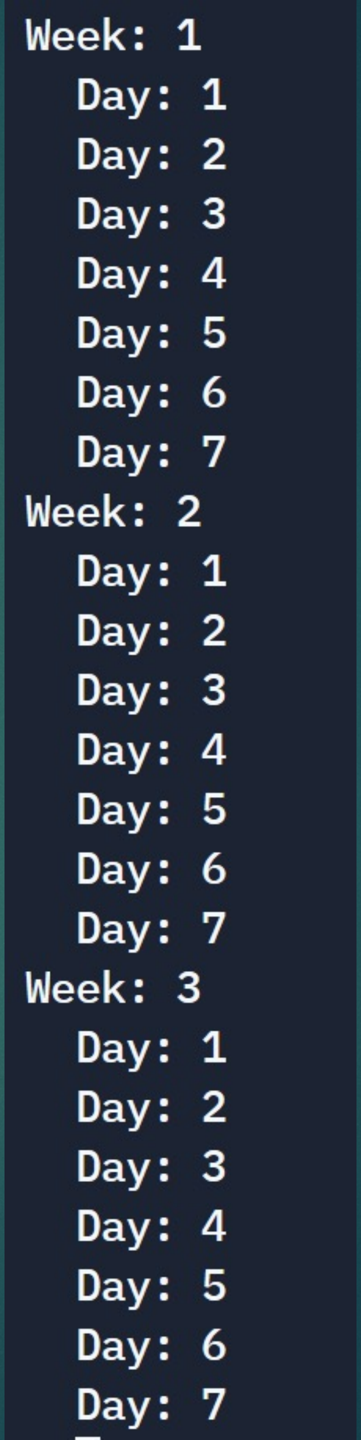
- Think nested if statements, but with loops!
  - A loop within a loop (within a loop, within a loop…)
- These tend to be harder to debug since it's two loops that can potentially go haywire and never stop… more on infinite loops in a bit
- The convention for nested for loops specifically is to have your outer for loop's variable be `i` and your inner for loop's variable be `j`

Nested Loops

- An example and its output…



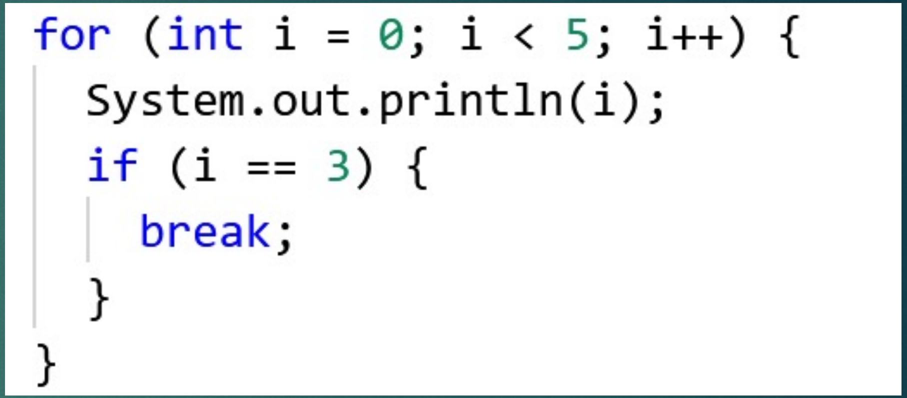
```java
int weeks = 3;
int days = 7;

  // outer loop prints weeks
  for (int i = 1; i <= weeks; ++i) {
    System.out.println("Week: " + i);

    // inner loop prints days
    for (int j = 1; j <= days; ++j) {
      System.out.println("  Day: " + j);
    }

  }
```



```
Week: 1
  Day: 1
  Day: 2
  Day: 3
  Day: 4
  Day: 5
  Day: 6
  Day: 7
Week: 2
  Day: 1
  Day: 2
  Day: 3
  Day: 4
  Day: 5
  Day: 6
  Day: 7
Week: 3
  Day: 1
  Day: 2
  Day: 3
  Day: 4
  Day: 5
  Day: 6
  Day: 7
```

The Break Keyword

- Remember this from switch cases?
- The break keyword essentially "breaks" the loop
  - It stops the loop exactly where it is in its tracks; code will continue to run, but outside of the for loop
  - Very helpful in debugging infinite loops!

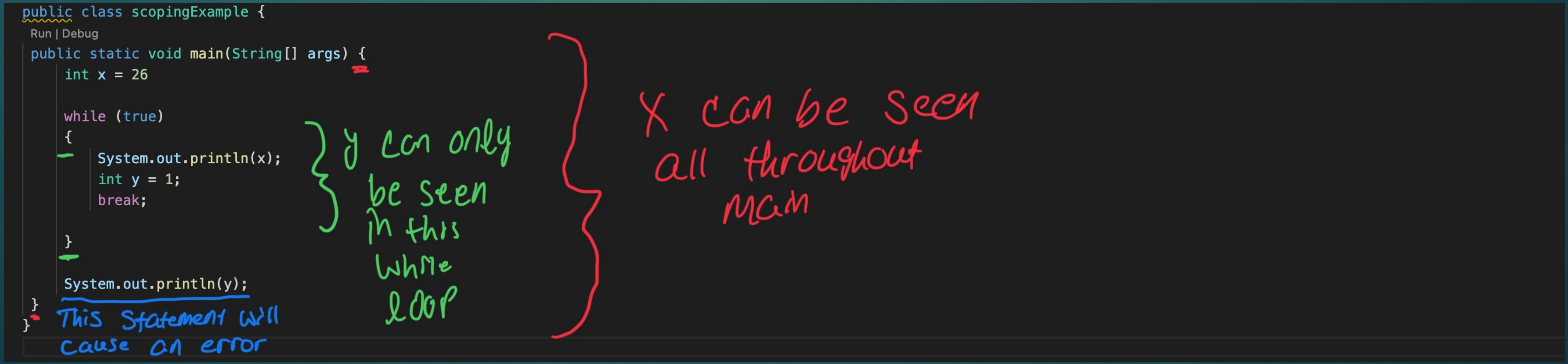


```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
  if (i == 3) {
    break;
  }
}
```

Outputs 0 1 2 3 AND THEN STOPS!

Scoping Blocks

- Variables declared in main can be seen anywhere in the main method.
- Variables declared in loops and if statements can only be seen in those loops and if statements.



```
public class scopingExample {
 Run | Debug
 public static void main(String[] args) {
     int x = 26

     while (true)
     {
         System.out.println(x);
         int y = 1;
         break;

     }

     System.out.println(y);
 }
}
```

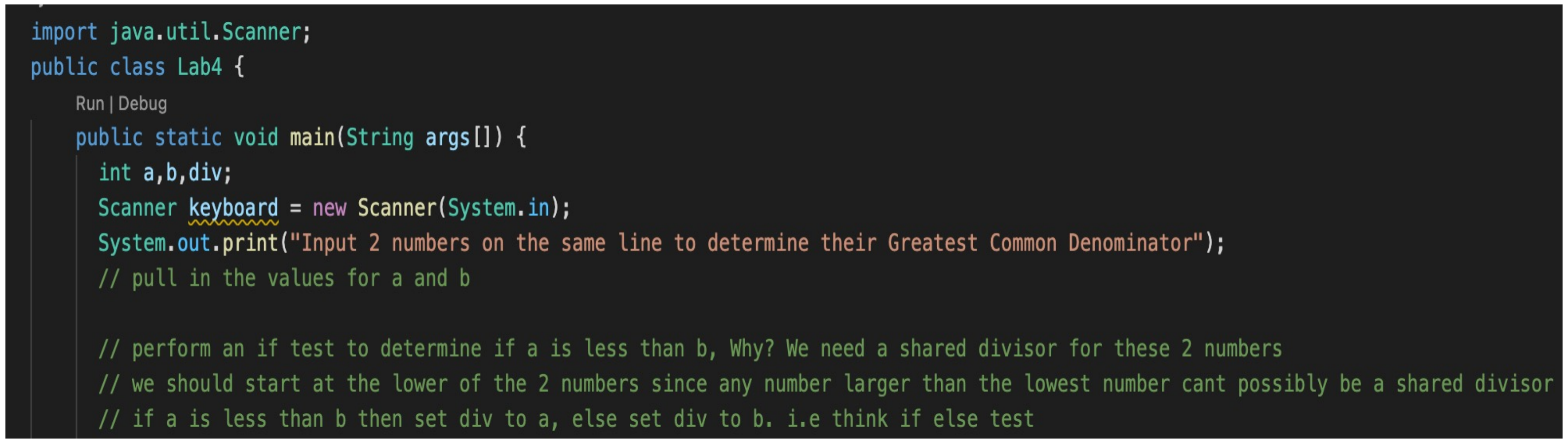
y can only be seen in this while loop
x can be seen all throughout main
This statement will cause an error

Lab 4

- Assignment:
  - https://canvas.pitt.edu/courses/127916/files/8050393?module_item_id=2735263

Lab 4

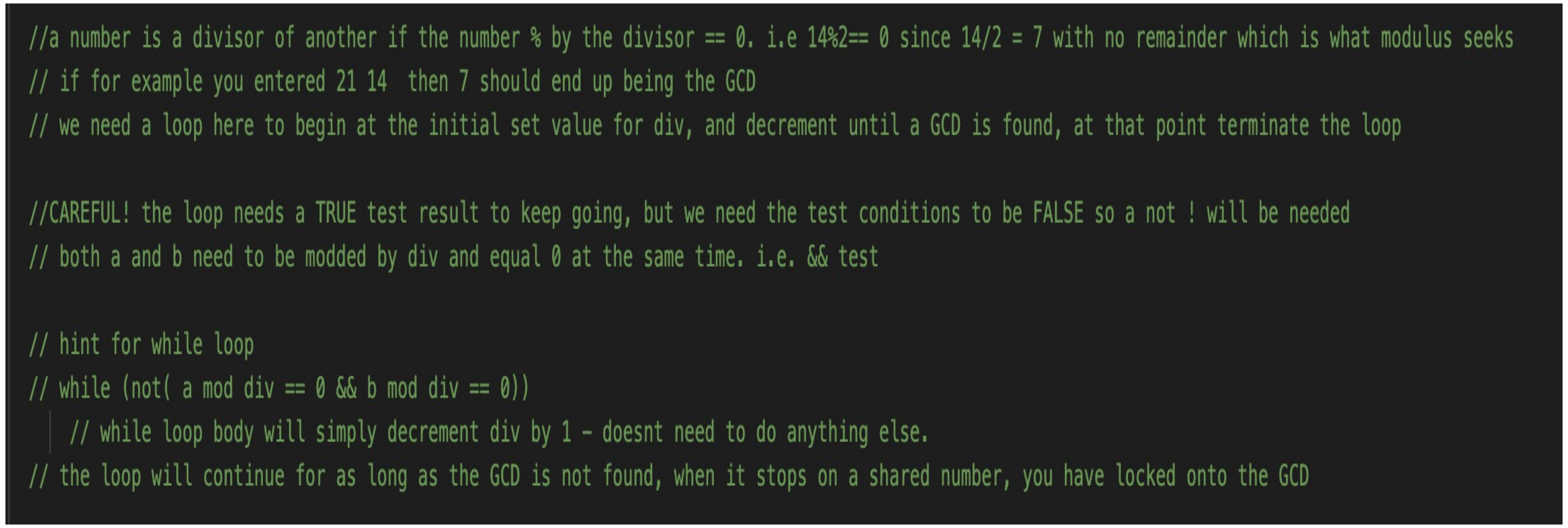
- You need to start by getting two values from one prompt on the command line.
  - Remember we do this with two keyboard.nextInt() calls in a row after the prompt
- Then, you need to perform a test to see if a < b
  - Can be done with if else with div being set equal to the proper variable.



```
import java.util.Scanner;
public class Lab4 {
    Run | Debug
    public static void main(String args[]) {
      int a,b,div;
      Scanner keyboard = new Scanner(System.in);
      System.out.print("Input 2 numbers on the same line to determine their Greatest Common Denominator");
      // pull in the values for a and b

      // perform an if test to determine if a is less than b, Why? We need a shared divisor for these 2 numbers
      // we should start at the lower of the 2 numbers since any number larger than the lowest number cant possibly be a shared divisor
      // if a is less than b then set div to a, else set div to b. i.e think if else test
```

Lab 4

- After you write your if-else test and set div equal to the right variable, all you need to do is use the while loop skeleton to write a while loop the decrements div by one every time the condition is in parenthesis is true.
  - Remember to use the right symbol (!) for not.
  - And the right symbol (%) for the modulus operator.



```
//a number is a divisor of another if the number % by the divisor == 0. i.e 14%2== 0 since 14/2 = 7 with no remainder which is what modulus seeks
// if for example you entered 21 14  then 7 should end up being the GCD
// we need a loop here to begin at the initial set value for div, and decrement until a GCD is found, at that point terminate the loop

//CAREFUL! the loop needs a TRUE test result to keep going, but we need the test conditions to be FALSE so a not ! will be needed
// both a and b need to be modded by div and equal 0 at the same time. i.e. && test

// hint for while loop
// while (not( a mod div == 0 && b mod div == 0))
    // while loop body will simply decrement div by 1 - doesnt need to do anything else.
// the loop will continue for as long as the GCD is not found, when it stops on a shared number, you have locked onto the GCD
```