CMPINF0401 Recitation

TUESDAYS 11:00-12:50

MICHAEL BARTLETT

Overview

- Midterm Review
- Assignment Reminder

Variables and Datatypes

- Declare a variable like so:
  - datatype variableName = value;
- Datatypes:
  - Primitive: boolean, char, int, short, byte, long, float, and double
- Objects:
  - String, Array, etc.

Differences in Datatypes

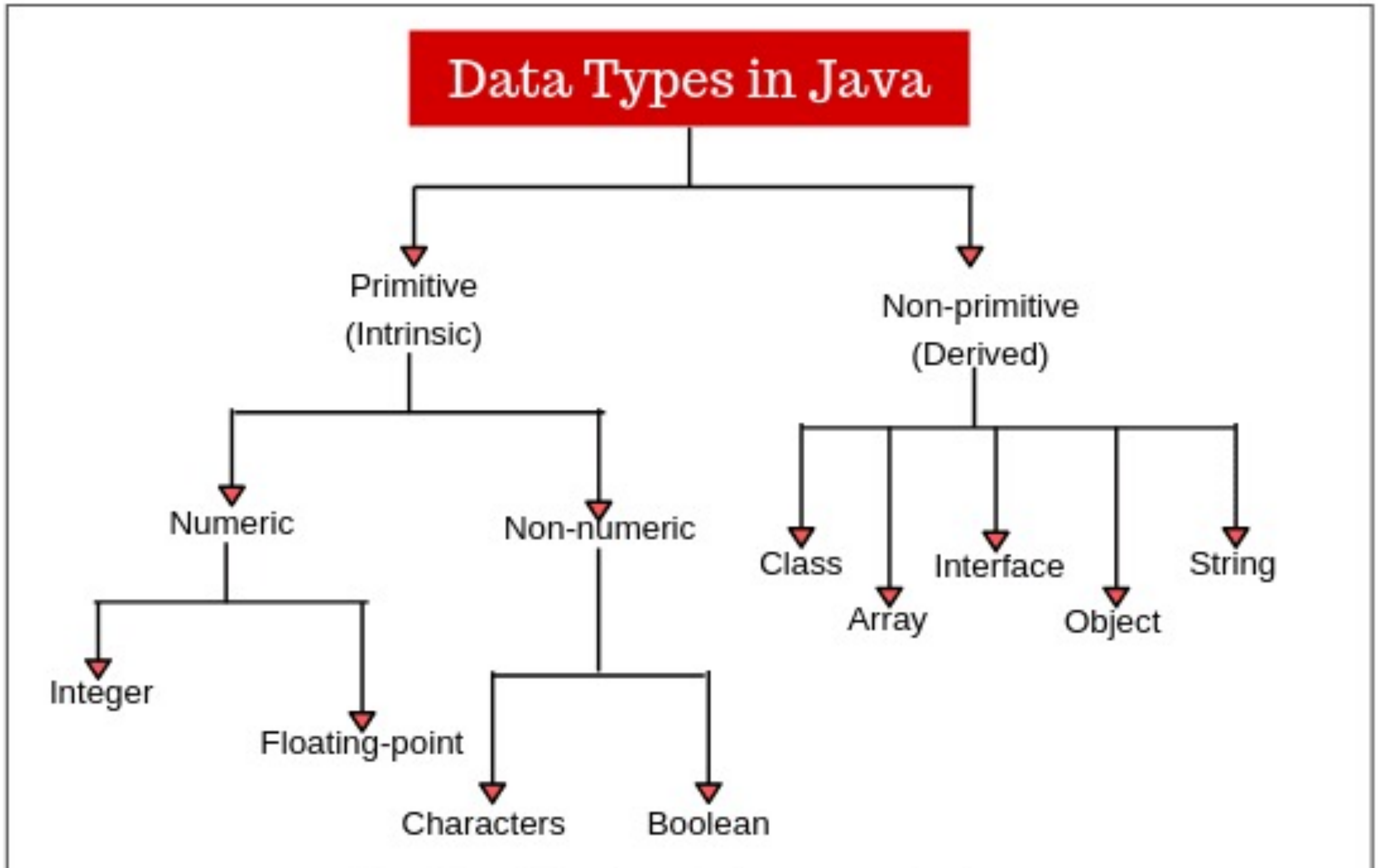
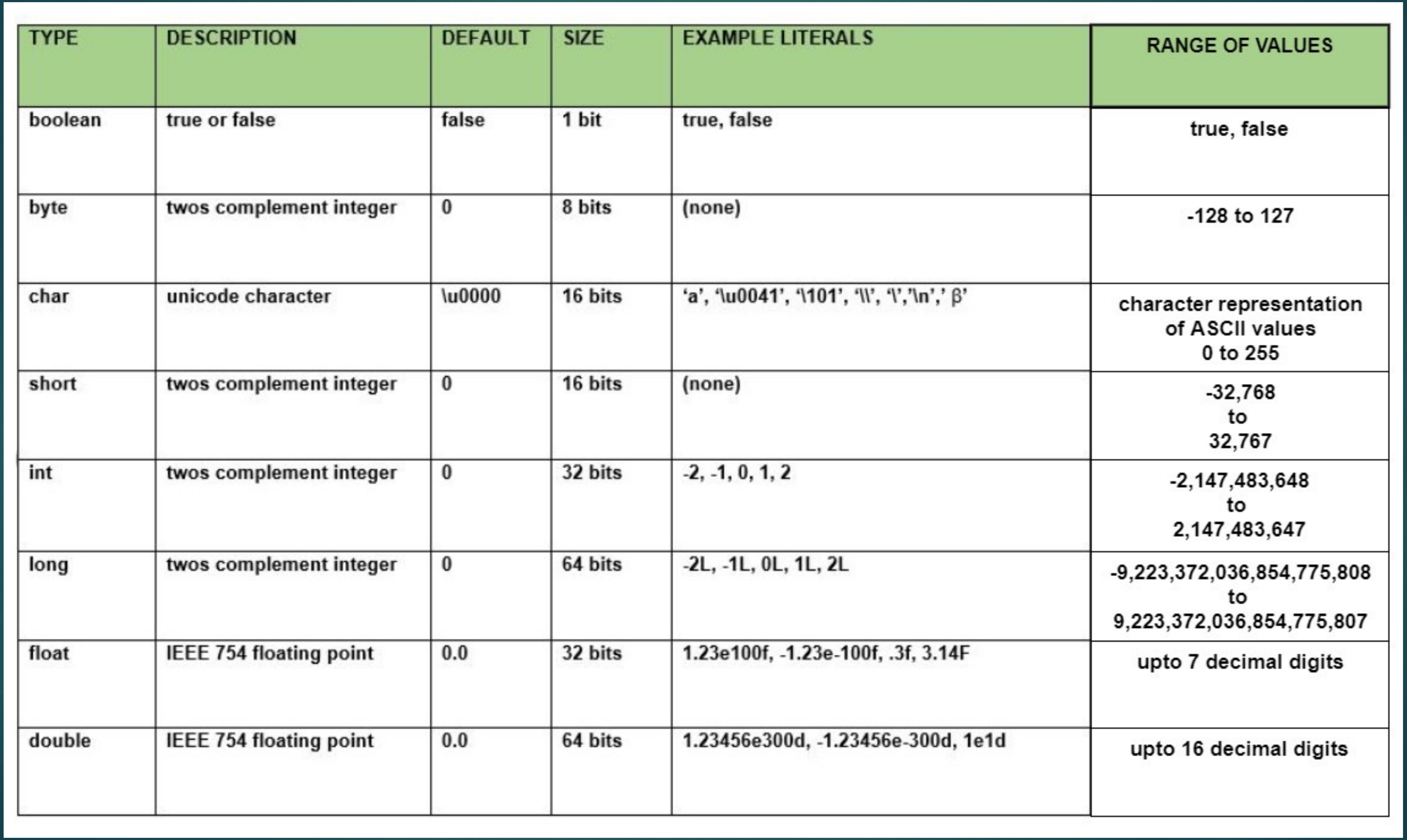


Fig: Classification of data types in java

Differences in Datatypes: Primitives



TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	‘a’, ‘\u0041’, ‘\101’, ‘\\’, ‘\’,’\n’,’ β’	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

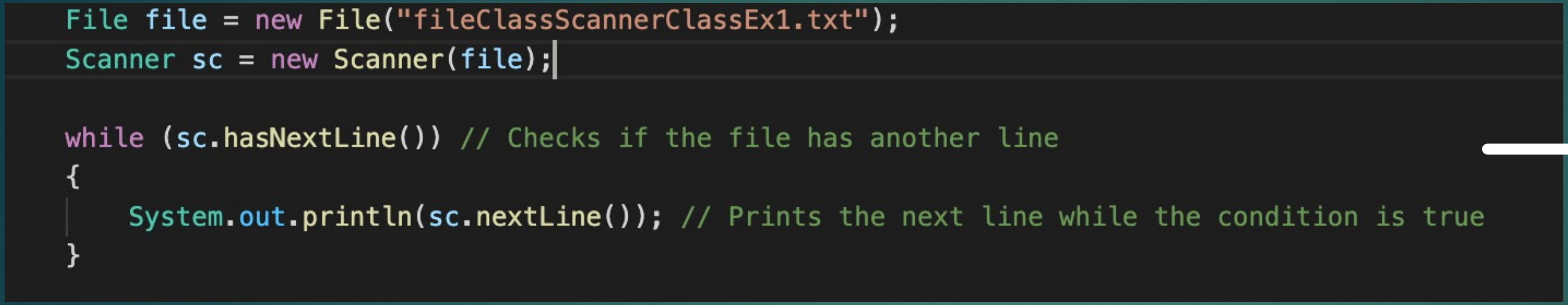
Using Scanner to Read From Keyboad

- To get user input we need to use a Scanner.
  - To initialize a Scanner, we make a variable to "hold" our scanner and then make a new Scanner using System.in
    - Scanner keyboard = new Scanner(System.in)
  - Once you have a Scanner, you can get input from the user and set the input equal to a variable:
    - String name = keyboard.next() // .next takes the next token as a String
    - int num = keyboard.nextInt() // Takes the user input as an int
      - If an int isn't entered the program will crash since it's expecting an int. This exception could be handled though.
    - More methods: https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

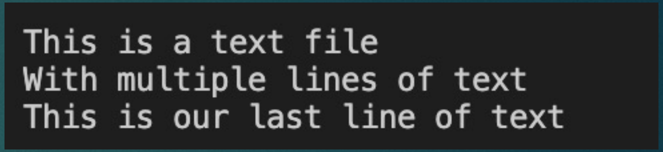
File I/O: Using the Scanner

- First, we need to use the File Class to get a reference to the file that we want to read.
- Then we use a Scanner to read the file and do something with the data.
- Scanner has multiple methods that we can utilize to read the file.
  - Check out the Java API docs for Scanner

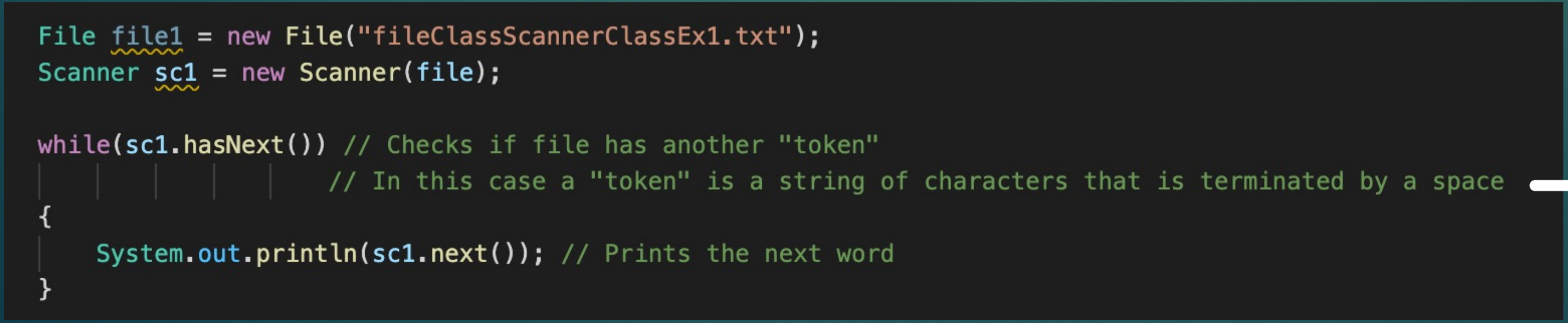
Reading Text Files Using Scanner



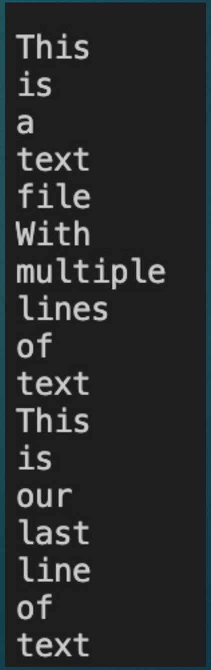
```
File file = new File("fileClassScannerClassEx1.txt");
Scanner sc = new Scanner(file);

while (sc.hasNextLine()) // Checks if the file has another line
{
    System.out.println(sc.nextLine()); // Prints the next line while the condition is true
}
```



```
This is a text file
With multiple lines of text
This is our last line of text
```



```
File file1 = new File("fileClassScannerClassEx1.txt");
Scanner sc1 = new Scanner(file);

while(sc1.hasNext()) // Checks if file has another "token"
                     // In this case a "token" is a string of characters that is terminated by a space
{
    System.out.println(sc1.next()); // Prints the next word
}
```

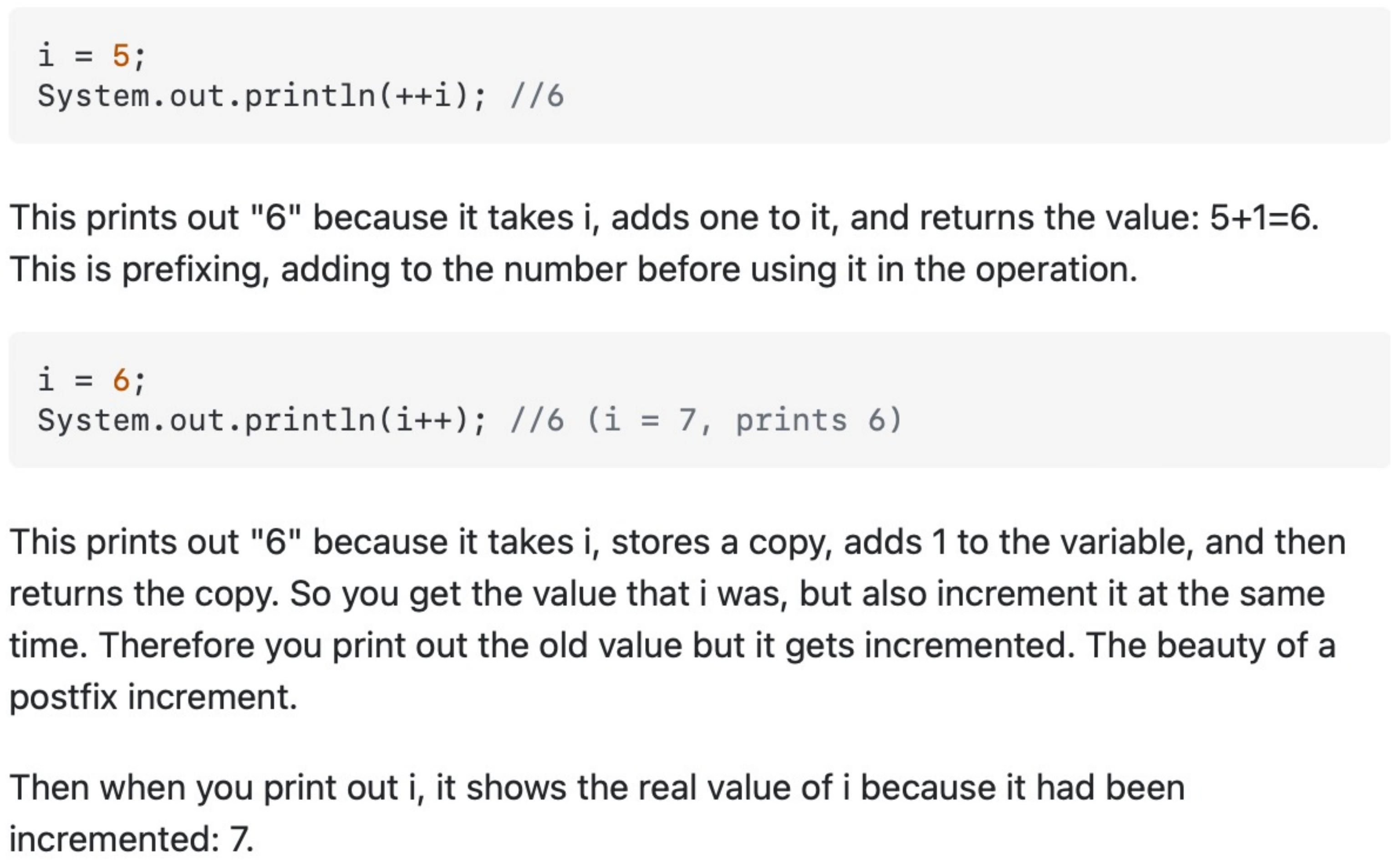


```
This
is
a
text
file
With
multiple
lines
of
text
This
is
our
last
line
of
text
```

Scanner's Methods

Return Type	Method and Description
boolean	hasNext() Returns true if this scanner has another token in its input. Useful when looping through a file to find its endpoint. There are multiple hasNext() methods such as: hasNextInt(), hasNextLine() and so on.
String	next() Finds and returns the next complete token from this scanner.
String	nextLine() Advances this scanner past the current line and returns the input that was skipped.
int	nextInt() Scans the next token of the input as an int.
double	nextDouble() Scans the next token of the input as a double.
boolean	nextBoolean() Scans the next token of the input into a boolean value and returns that value.

Prefix vs. Postfix Incrementation



```
i = 5;
System.out.println(++i); //6
```

This prints out "6" because it takes i, adds one to it, and returns the value: 5+1=6. This is prefixing, adding to the number before using it in the operation.

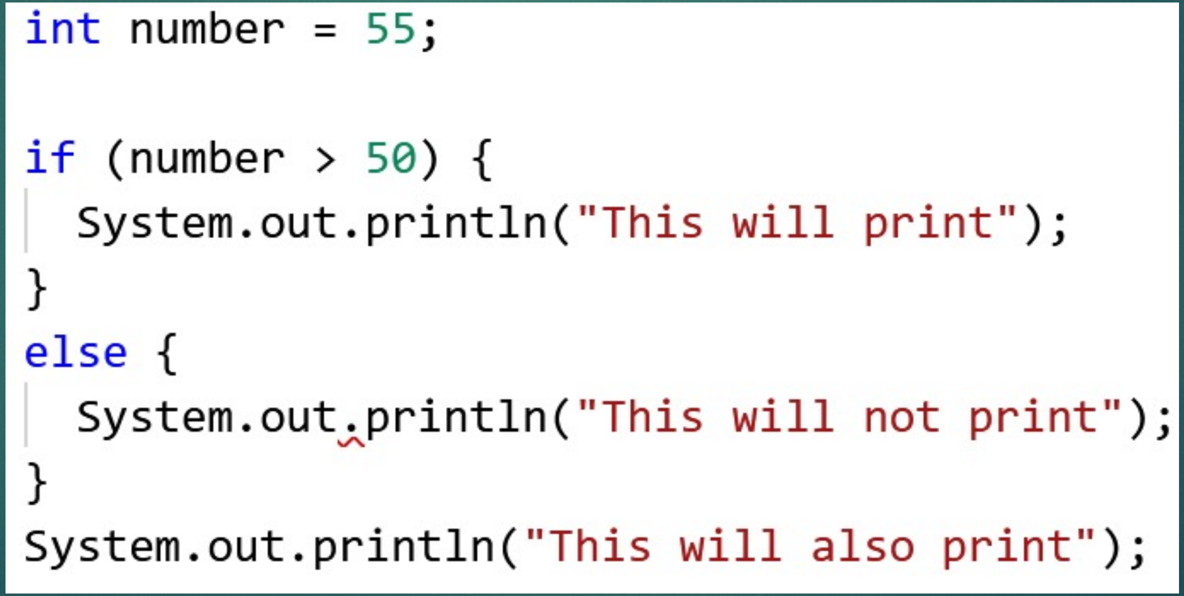
```
i = 6;
System.out.println(i++); //6 (i = 7, prints 6)
```

This prints out "6" because it takes i, stores a copy, adds 1 to the variable, and then returns the copy. So you get the value that i was, but also increment it at the same time. Therefore you print out the old value but it gets incremented. The beauty of a postfix increment.

Then when you print out i, it shows the real value of i because it had been incremented: 7.

If structures

- Simpler than you might think…
- If this is true, do this. Otherwise, do that.

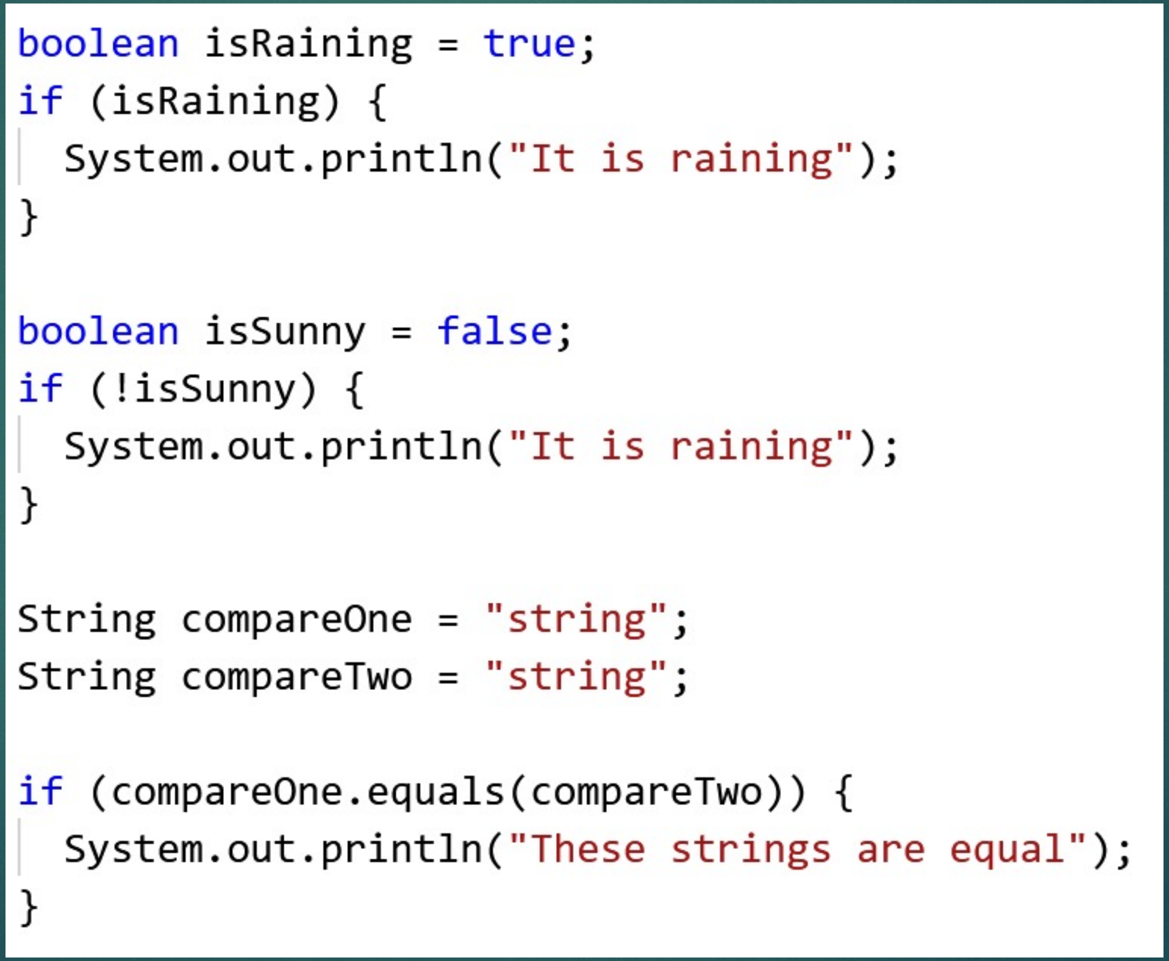


```
int number = 55;

if (number > 50) {
    System.out.println("This will print");
}
else {
    System.out.println("This will not print");
}
System.out.println("This will also print");
```

What goes in an if statement?

Condition	Meaning
a == b	Equal to
a > b	Greater than
a < b	Less than
a >= b	Greater than or equal to
a <= b	Less than or equal to
a != b	Not equal to
str1.equals(str2)	Seeing if strings are equal
str1.equalsIgnoreCase(str2)	Seeing if strings are equal while ignoring case sensitivity

What goes in an if statement?

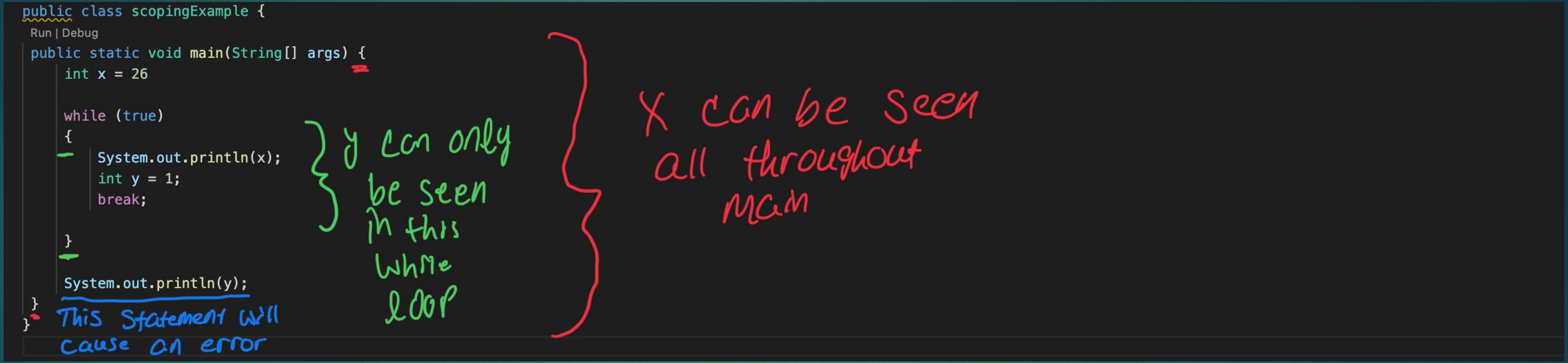


```
boolean isRaining = true;
if (isRaining) {
  System.out.println("It is raining");
}

boolean isSunny = false;
if (!isSunny) {
  System.out.println("It is raining");
}

String compareOne = "string";
String compareTwo = "string";

if (compareOne.equals(compareTwo)) {
  System.out.println("These strings are equal");
}
```

Some Notes about If Statements

- An If statement doesn’t need an else, you can have just an if.
- You can also have multiple else cases: else ifs
- You can have as many else ifs as the stack allows (a large number)
- Remember that the curly braces are for code that you only want to perform if the if statement runs.
  - Also remember that anything declared in these curly braces is local to the statement.

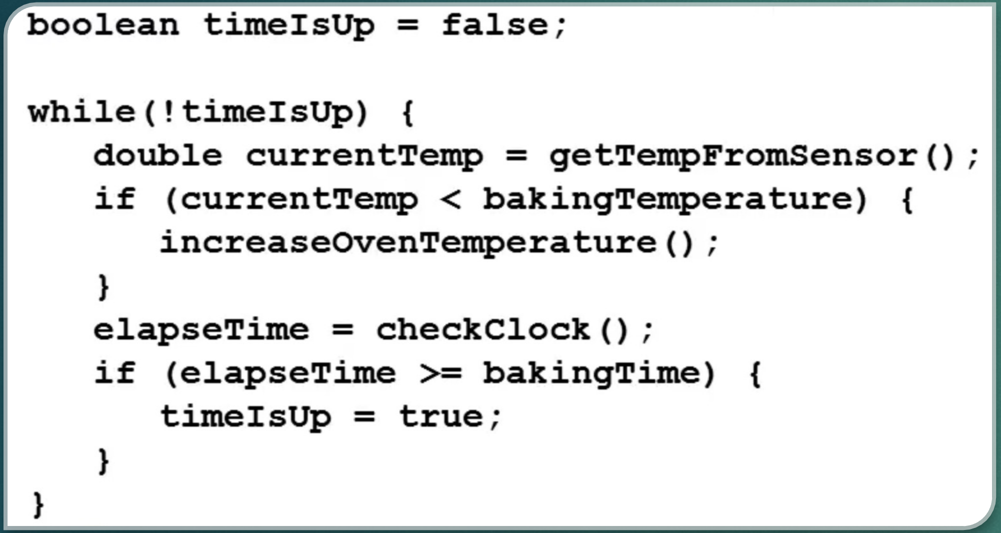
Code Blocks



```
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

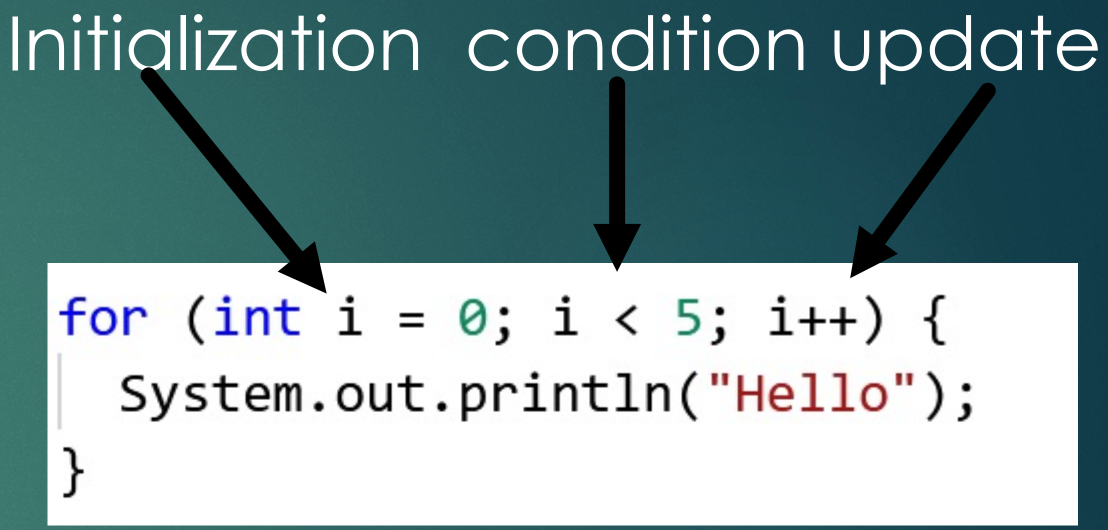
While Loops



```
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

For Loops

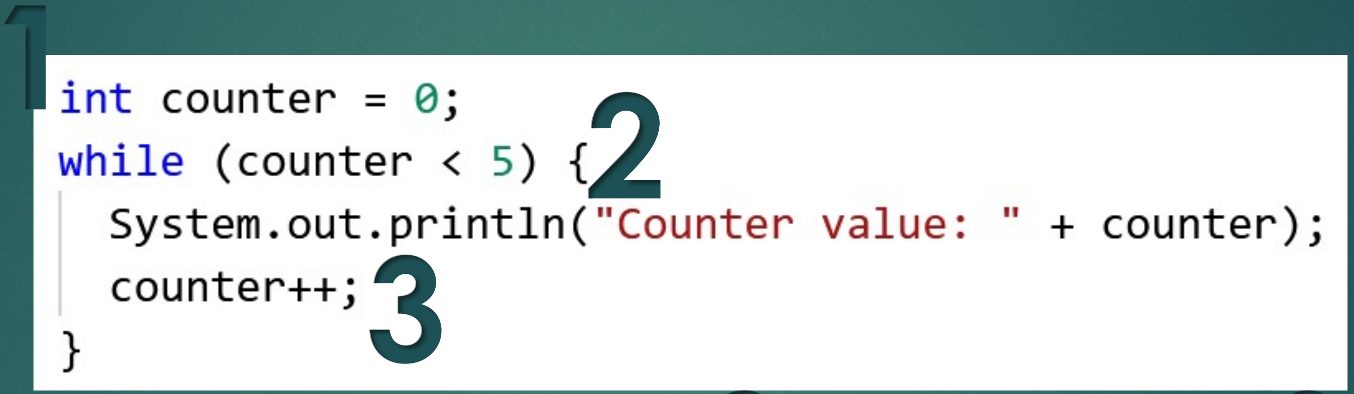


```
for (int i = 0; i < 5; i++) {
  System.out.println("Hello");
}
```

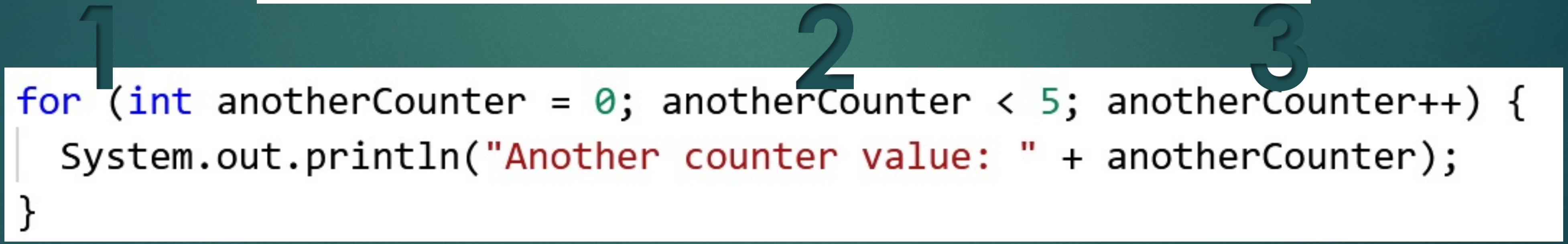
- There are three clauses in a for loop:
  - Initialization
  - Condition
  - Update
- The convention when initializing a variable is to use the letter `i` as your variable name – it's essentially the only time you should be using single-letter variables!

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);
}
```

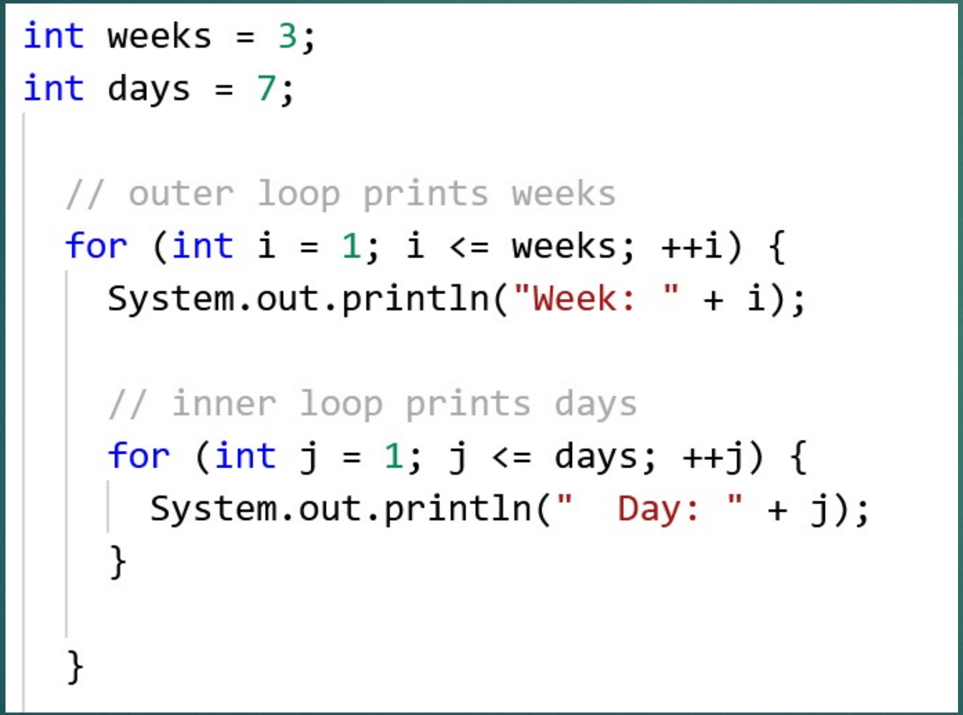
- Initialize a variable (1), create a Boolean condition (2), increment the variable (3)

Nested Loops

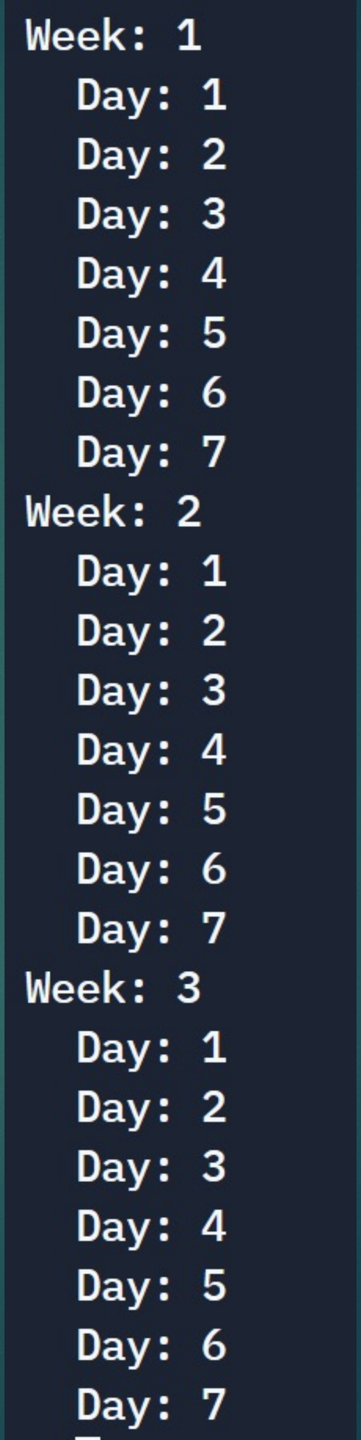
- Think nested if statements, but with loops!
  - A loop within a loop (within a loop, within a loop…)
- These tend to be harder to debug since it’s two loops that can potentially go haywire and never stop… more on infinite loops in a bit
- The convention for nested for loops specifically is to have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

Nested Loops

- An example and its output…



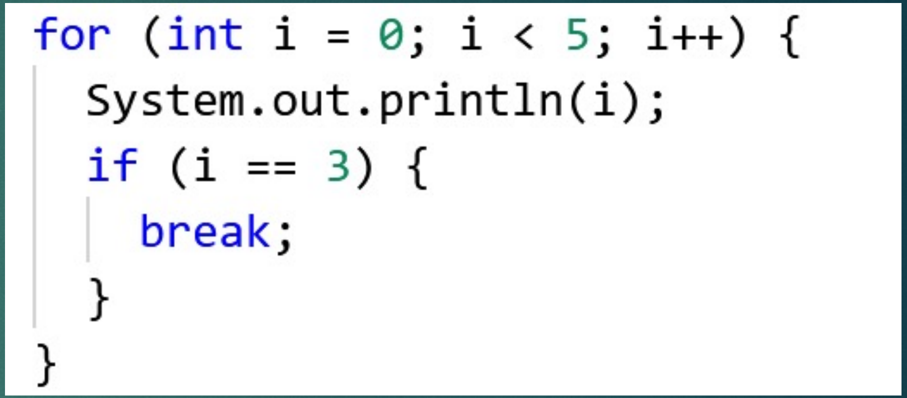
```java
int weeks = 3;
int days = 7;

  // 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!



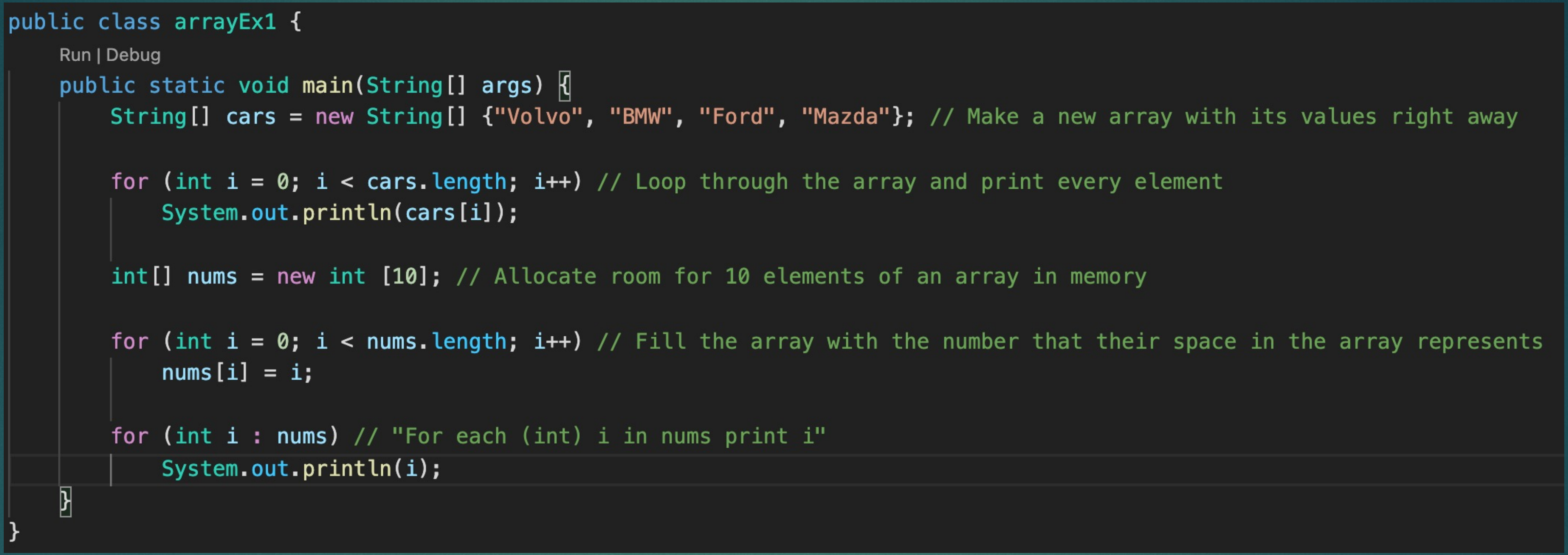
```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
  if (i == 3) {
    break;
  }
}
```

Outputs 0 1 2 3 AND THEN STOPS!

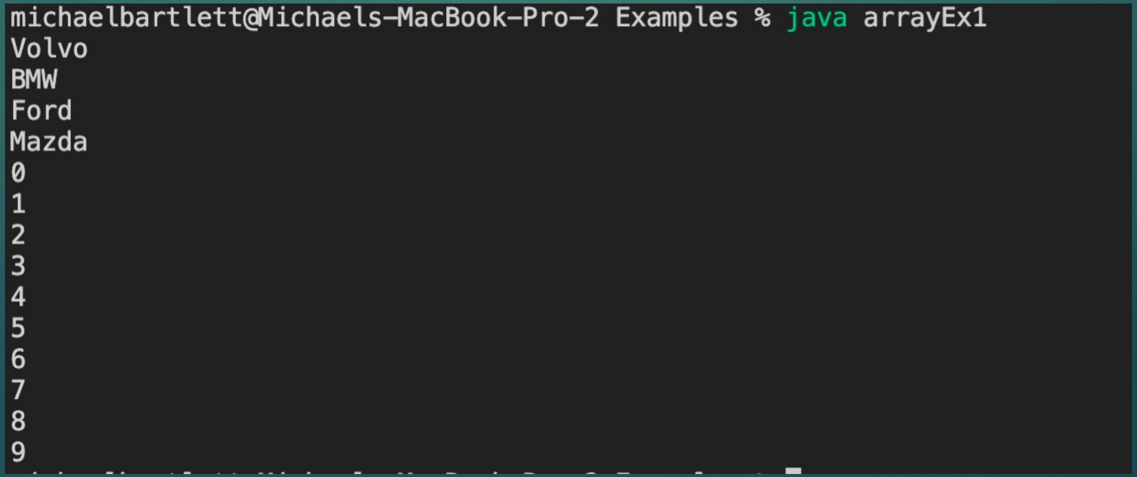
What's an Array?

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
  - This variable is a "reference variable." This means it "points" to the first object in the array and then all the objects in the array are stored next to it in memory.
  - Therefore, we uses indices when accessing an array. Let's say we have an array named arr and we want the first item, arr[0] will give us that item, arr[1] gives us the second item and so on…
- When making an array, declare the type that's going to be stored in it and name it. Then set the values that you want in the array.
- Or you can declare an array that's not equal to anything and just has its size in the square brackets.

Creating/Reading Arrays



```java
public class arrayEx1 {
    Run | Debug
    public static void main(String[] args) {
        String[] cars = new String[] {"Volvo", "BMW", "Ford", "Mazda"}; // Make a new array with its values right away

        for (int i = 0; i < cars.length; i++) // Loop through the array and print every element
            System.out.println(cars[i]);

        int[] nums = new int [10]; // Allocate room for 10 elements of an array in memory

        for (int i = 0; i < nums.length; i++) // Fill the array with the number that their space in the array represents
            nums[i] = i;

        for (int i : nums) // "For each (int) i in nums print i"
            System.out.println(i);
    }
}
```



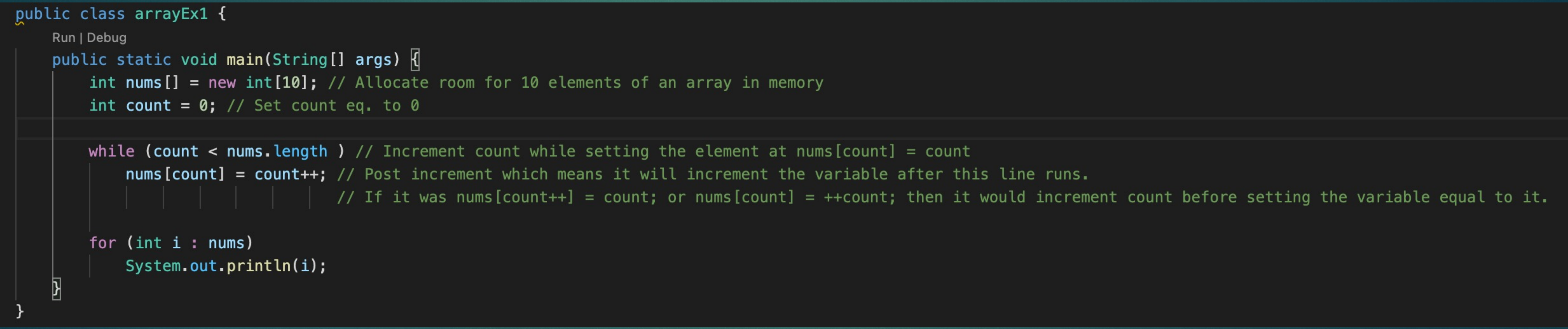
```
michaelbartlett@Michaels-MacBook-Pro-2 Examples % java arrayEx1
Volvo
BMW
Ford
Mazda
0
1
2
3
4
5
6
7
8
9
```

The Array Discipline

- When you declare an array you should declare an int named count or such to track how many values you have put into the array
- Initialize count to 0 and then use count to represent two things:
  - The number of values you have put into the array so far
  - The index position of where the next value should be stored

Credit: Prof. M. Devine

Let's look at the example again using Array Discipline



```
public class arrayEx1 {
    Run | Debug
    public static void main(String[] args) {
        int nums[] = new int[10]; // Allocate room for 10 elements of an array in memory
        int count = 0; // Set count eq. to 0

        while (count < nums.length ) // Increment count while setting the element at nums[count] = count
            nums[count] = count++; // Post increment which means it will increment the variable after this line runs.
                                   // If it was nums[count++] = count; or nums[count] = ++count; then it would increment count before setting the variable equal to it.

        for (int i : nums)
            System.out.println(i);
    }
}
```

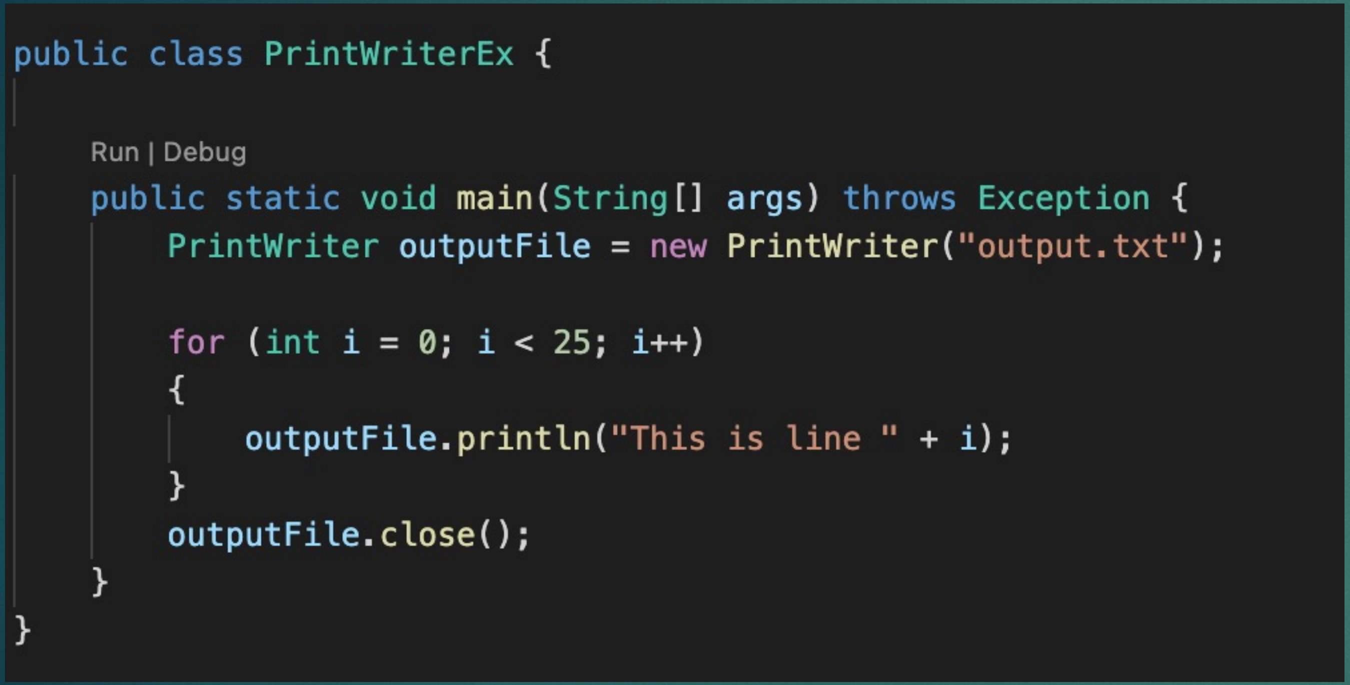
Notes About Arrays

- Array length vs the upperbound of an array
  - The length returns the number of items that can be stored in the array whereas the upperbound is 1 – arr.length.
    - This is because arrays are 0 indexed.
  - I.e., an array of length 5 has an upper bounds of 4
- If you try to access an index that is out of bounds, you get an: **ArrayIndexOutOfBounds exception**

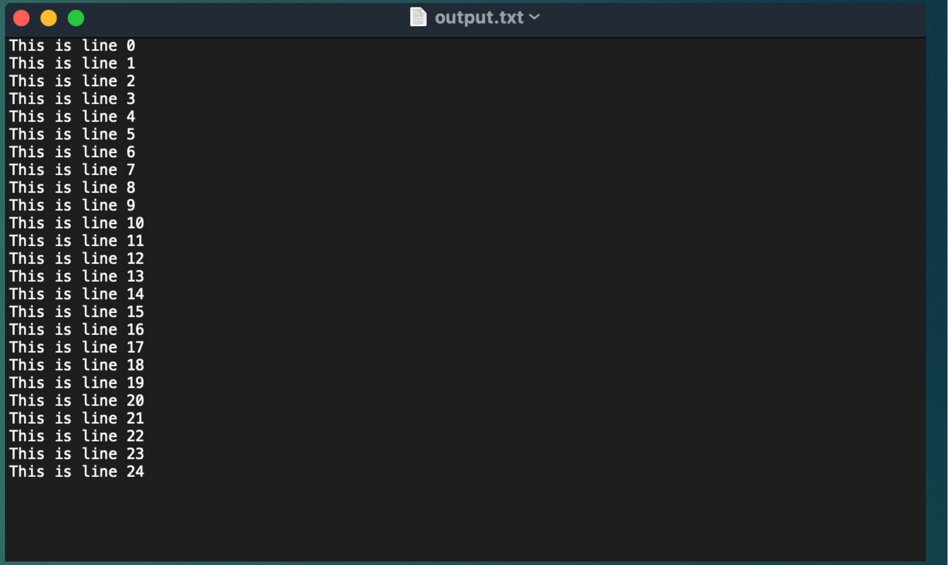
Writing to files Using PrintWriter

- First, make a new PrintWriter object and set the file name.
- Then, write your data.
- Make sure to close it using writer.close();
  - Your data won't write if you don't do this.

Writing Text Files Using PrintWriter



```
public class PrintWriterEx {

    Run | Debug
    public static void main(String[] args) throws Exception {
        PrintWriter outputFile = new PrintWriter("output.txt");

        for (int i = 0; i < 25; i++)
        {
            outputFile.println("This is line " + i);
        }
        outputFile.close();
    }
}
```

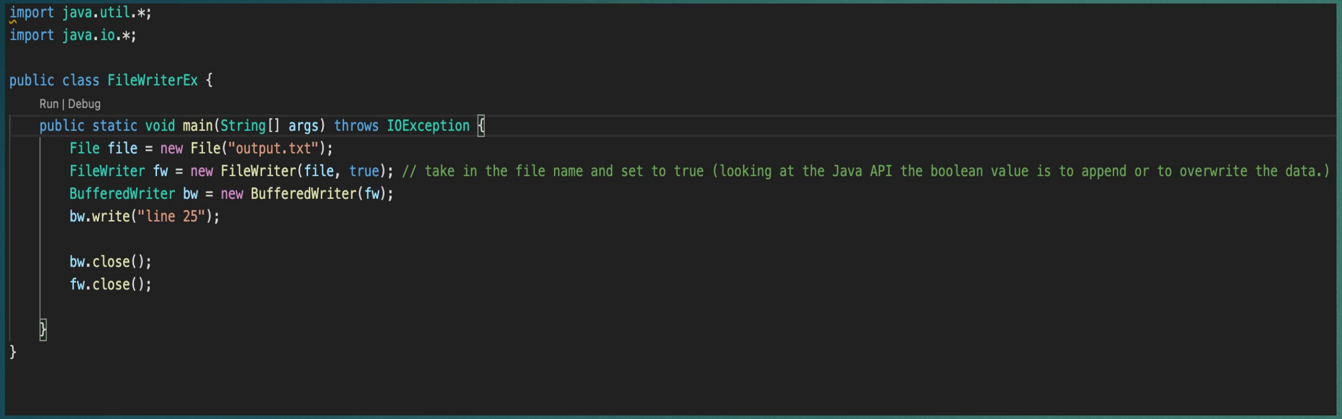


```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
This is line 11
This is line 12
This is line 13
This is line 14
This is line 15
This is line 16
This is line 17
This is line 18
This is line 19
This is line 20
This is line 21
This is line 22
This is line 23
This is line 24
```

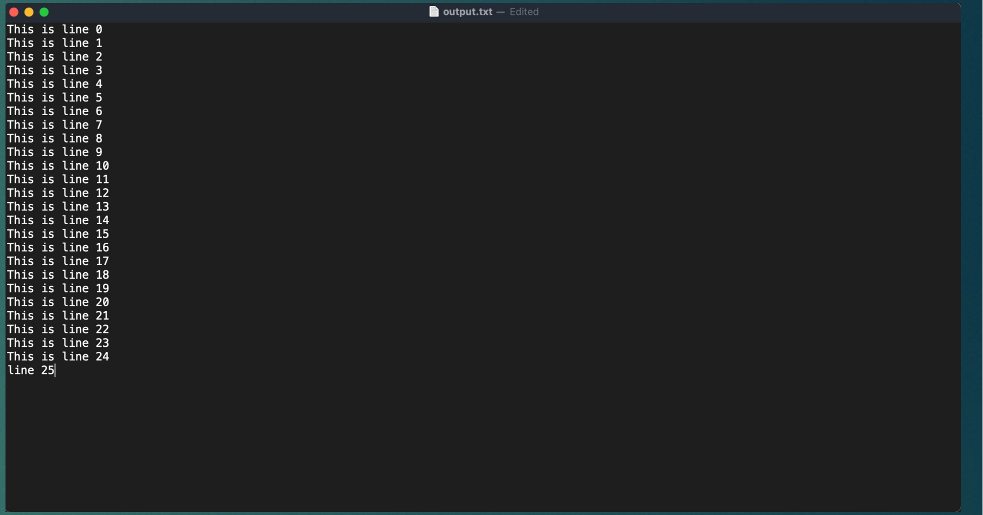
Appending Data To Existing File Using FileWriter

- First, make a new File object
- Then, make a new FileWriter object and set the file name.
- Then, make a BufferedWriter object out of your FileWriter object
- Write your data and then close the BufferedWriter and FileWriter objects.

Appending Data To Existing File Using FileWriter



```
import java.util.*;
import java.io.*;

public class FileWriterEx {
    Run | Debug
    public static void main(String[] args) throws IOException {
        File file = new File("output.txt");
        FileWriter fw = new FileWriter(file, true); // take in the file name and set to true (looking at the Java API the boolean value is to append or to overwrite the data.)
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write("line 25");

        bw.close();
        fw.close();

    }
}
```



output.txt — Edited

```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
This is line 11
This is line 12
This is line 13
This is line 14
This is line 15
This is line 16
This is line 17
This is line 18
This is line 19
This is line 20
This is line 21
This is line 22
This is line 23
This is line 24
line 25
```

Assignment 3

- Due 2/28
- Check out last week's slides if you need some hints