Java Memory and Intro to OOP
What is Java?

- Programming language
- Standard libraries
  - ex. Math.random()
- Tools: ‘compiler’, ‘runtime’, others…
What is the compiler?

- AKA javac

- A compiler is a program that translates a source program written in some high-level programming language (such as Java) into a lower level language that a machine can understand.

- Turns .java files into .class files.

- The compiler serves many purposes, but an important one is that it is a way to do error checking before running your program.

- You know those red lines in Eclipse? They indicate a compile error.
What is the runtime?

- AKA The Jvm
  - Stands for is a ‘Java Virtual Machine’

- Is the ‘runtime’ i.e. the thing that runs your program once it is compiled and you execute it.

- Sits between your program and the operating system and abstracts away complexity
  - including managing the memory of your program
  - and lots, lots, lots more.
Errors in Java programs

- **Syntax errors** - These are errors the compiler can catch. Moreover, the code you wrote is not correct Java.
  - Example: `int[] numbers = new float[10];`
  - Those pesky red squiggly lines in Eclipse indicate a syntax error.
  - Compiler is our buddy! Finds errors early for us.

- **Runtime errors** - An error that occurs when your program is running. May crash your program.
  - Example: division by zero, array index out of bounds
  - Can be hard to debug. Use the debugger!

- **Logic errors** - Your code is correct syntactically and does not have runtime errors, but it just does the wrong thing.
Java Program Lifecycle
What is memory?

- Every time a program is run, some memory is allocated for your program.
- You can think of memory as the computer’s work area for executing a program.
- Every bit of memory has a unique *address* that is used for storing and accessing data (variables) for your program.
- How much data your program is working with informs how much memory a given program requires in order to run properly.
JVM Memory Management

- When the JVM is started it asks for a specific amount of memory from the operating system.

- You can control this yourself, but it defaults to some number of megabytes, probably something like 128MB (depends on the system).

- If your program exceeds that amount, the JVM will handle asking for more from the operating system up to some maximum size.
Types of Memory in Java

- When you declare a variable in a program, Java allocates space for that variable from one of several memory regions.

- One region of memory is reserved for variables that are never created or destroyed as the program runs, such as constants. This information is called static data.

- Whenever you create a new variable of a reference type (using the new keyword), Java allocates space from a pool of memory called the heap.

- Each time you call a method, Java allocates a new block of memory called a stack frame to hold its variables. These stack frames come from a region of memory called the stack.
public class TypesOfMemory() {

    public static final int STATIC_ALLOCATED = 0;

    public static void main(String[] args) {
        String referenceToObjectOnHeap = new String("On the heap");

        int stackAllocatedArgument = 0;
        method(stackAllocatedArgument);
    }

    public void method(int stackAllocatedParameter) {
        int stackAllocatedVariable = stackAllocatedParameter + 1;
        System.out.println(stackAllocatedVariable);
    }
}

With me so far?

- JVM manages memory for your variables
- Depending on the characteristics of the variable it stores the data in different ‘regions’
- The regions are:
  - Stack
  - Heap
  - Static
- We are not going to cover static in detail today. Much easier after we understand classes.
The Stack
What is a Stack?

- The ‘stack’ we have been talking about so far is a memory region. However, it gets its name from the *data structure* it uses.

- By data structure, we mean something that is meant to hold data and provides certain operations on that data.

- An array is an example of simple data structure. Its ‘operations’ are using offsets with brackets.

  - ex. someArray[2];
What is a Stack?

- A stack is a bit more sophisticated than an array, but we don’t need to understand it deeply for this class.

- The two important things..
  - Its “first-in, last-out”
  - Its operations are “push” and “pop”
  - Its easy to understand with an analogy…. 
What is a Stack?

- Imagine a stack of plates...
  - You can “push” a new plate onto the stack of plates by sticking a plate on top
  - You can “pop” a plate off the stack by taking the top one
The Call Stack

- The stack memory region works like the stack data structure
- What gets pushed and popped from it are “stack frames”
Stack Frame

- Every time a method is called a “stack frame” gets created.

- You can think of the “stack frame” as the set of all the variables needed for that method.
Pushing and Popping Stack Frames

- Every time a method is invoked a “stack frame” for that method is pushed onto the stack memory region.

- When the method returns, the “stack frame” gets popped, and all those variables are deleted!

- Once a stack frame is popped, that region of memory becomes available for other stack variables.
The main method is invoked.
j is declared and initialized

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+i+" and "+j+" is "+k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}

The main method is invoked.
Tracking the Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}
```

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Declare k

Space required for the main method

- k: j: 2
- i: 5

The main method is invoked.
Tracking the Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between \"+ i + \" and \"+ j + \" is \"+ k);
}
```

Space required for the main method
- k: 
- j: 2
- i: 5

The main method is invoked.

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```
pass the values of i and j to num1 and num2

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Space required for the main method
- num1: 5
- num2: 2

The max method is invoked.

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println(
        "The maximum between " + i + 
        " and " + j + " is " + k);
}
```
Tracking the Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i + ", " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

- Declare result
- Space required for the main method
  - num1: 5
  - num2: 2
  - result: 5
- The max method is invoked.
  - k: 5
  - j: 2
  - i: 5
(num1 > num2) is true

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i + 
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Space required for the main method:
- num1: 5
- num2: 2
- k: 5
- j: 2

The max method is invoked.
Tracking the Stack

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2)
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Assign num1 to result

<table>
<thead>
<tr>
<th>Space required for the max method</th>
</tr>
</thead>
<tbody>
<tr>
<td>result: 5</td>
</tr>
<tr>
<td>num2: 2</td>
</tr>
<tr>
<td>num1: 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Space required for the main method</th>
</tr>
</thead>
<tbody>
<tr>
<td>k:</td>
</tr>
<tr>
<td>j: 2</td>
</tr>
<tr>
<td>i: 5</td>
</tr>
</tbody>
</table>

The max method is invoked.
Tracking the Stack

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}
```

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```
Tracking the Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+i+" and "+j+" is "+k);
}
```

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Space required for the main method

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>5</td>
</tr>
<tr>
<td>j</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>5</td>
</tr>
</tbody>
</table>

The main method is invoked.
Tracking the Stack

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Complete the method

The main method is complete.
With me so far?

- For every method call (even the main method!) a stack frame is created.

- *All* variables declared in the method are in the stack frame.

- When the method exits, those variables go bye-bye!
Heap Memory
‘Reference’ Types

- In Java we have two types of variables, *primitives* and *references*

- Reference types are *classes*

- The class can be a class that’s provided as part of the Java API class library..
  - ex String, Scanner, arrays, etc.
  - ..or a class that you write yourself. (We’ll be doing that soon).
Variables on the Heap

- When you create a variable of a reference type, Java allocates the amount of memory the object requires to store the object on the heap.
  - (You know this happened when you see `new`)

- Then, the variable is actually assigned a reference to the object, not the object itself.

```java
String referenceToObjectOnHeap = new String("On the heap");
```

  ^^ reference (aka pointer)  ^^ creates variable on heap
Key Idea

- Again, when working with reference types, a variable of a particular type *doesn’t actually contain an object of that type*.

- Instead, it contains a *reference to an object* of that type that is resident on the heap!

- This is the *key intuition* and you must understand this in order to understand why object oriented constructs behave the way they do. (when we get there)
We’ve seen this before

- Yet again a variable of a reference type doesn’t actually contain an object of that type.

- An important side effect is that two variables can refer to the same object.

- Remember this?

```java
1  char[] a1 = new char[];
2
3  // Doing this does not copy the array!
4  // a1 and a2 are now literally the same array.
5  char[] a2 = a1;
```
Why?

- Consider this code...
- We call a method on line 4
- What happens?
- New stack frame!

```java
public class WhyHeap {

    public static void main(String[] args) {
        int[] a = createArray();
        for(int i = 0 ; i < a.length() ; ++i) {
            System.out.println(a[i]);
        }
    }

    public static int[] createArray() {
        int[] arrayOfInts = new int[2];
        arrayOfInts[0] = 1;
        arrayOfInts[1] = 2;
        return arrayOfInts;
    }
}
```
Why?

- We create an array on line 9.
- What happens?
- Reference is on stack frame, array itself is on heap
Why?

- We return the array at line 12.

- What happens?

- We copy the reference into main’s stack frame, array itself is on heap
Why?

- We return the array at line 12.
- What happens?
- We copy the reference into main’s stack frame.
Why?

- If it were not for the heap this code would not work as we expect or be horrifically inefficient.

```java
class WhyHeap {
    public static void main(String[] args) {
        int[] a = createArray();
        for(int i = 0 ; i < a.length() ; ++i) {
            System.out.println(a[i]);
        }
    }

    public static int[] createArray() {
        int[] arrayOfInts = new int[2];
        arrayOfInts[0] = 1;
        arrayOfInts[1] = 2;
        return arrayOfInts;
    }
}
```
Intro to Object Oriented Programming
What is OOP

- The most popular programming languages developed in the last 30 years are, for the most part, Object-Oriented languages, but this wasn't always the way.

- The prevailing paradigm before that was “Procedural Programming”

- It had functions, but the source files for large programs would be many thousands of lines long!

- This led to software that was extremely difficult to understand and change.
What is OOP

- OOP was a direct response to that condition

- In an OOP language, this one large program will instead be split apart into self contained *objects*, almost like having several mini-programs.

- Each *object* represents a different part of the application. Now each object contains its own data and its own logic, and they communicate and collaborate to execute your program.
What is OOP

- Furthermore, OOP is…
  
  - a programming method that is used to help organize your code when you have complex programs that require a lot of code.
  
  - a set of language constructs that help you organize your code according to this method.

- *That’s it.*
Benefits of OOP

- Allows us to organize our code in a way that allows us to isolate responsibility to a single entity, therefore reducing complexity
  - “Do one thing and do it well”
- Allows us to have code that is reusable
  - “Don’t repeat yourself”
- Allows us to model the “primitives” in our system as types.
Nouns & Verbs

- So how do we make these mini-programs?
- Identify the entities in the program. Our nouns.
- Identify the actions in the program. Our verbs.
- For example, let’s say we had wanted to write the software for an online shopping application.
  - What types of entities would be in the system? ex ‘Shopping Cart’
  - What kinds of actions would be performed by the actors? ex. ‘checkout’
Classes

- A class represents a ‘noun’ in our program
- It has two attributes..
  - *State* - the variables on the class.
  - *Methods* - the ways in which the object can interact with its data, the ‘verbs’.
If we think of a real-world object, such as a television, it will have several features and properties:

- We do not have to open the case to use it.
- We have some controls to use it (buttons on the box, or a remote control).
- We understand the concept of a television without necessarily understanding how it is built and functions.
- It is complete when we purchase it, with any external requirements well documented.

Very much how you might describe a class!
Similarly, a class should…

- Provide a well-defined purpose
- Represent a clear concept
- Be complete and well-documented
- Do one thing and do it well!
Objects

- An object is an instance of a class. What we get when we call `new`.
- You could think of a class as the description of a concept, and an object as the realization of this description.
Classes & Object Example

```java
public class Rectangle {
    final int length;
    final int width;

    // Constructor.. special kind of function.
    public Rectangle(int w, int l) {
        width = w;
        length = l;
    }

    public int getArea() {
        return width * length;
    }
}

public class SomeOtherClass {
    public static void main(String[] args) {
        Rectangle r = new Rectangle(100, 50);
        System.out.println(r.getArea());
    }
}
```