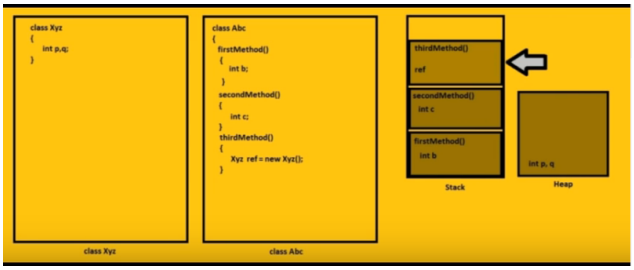
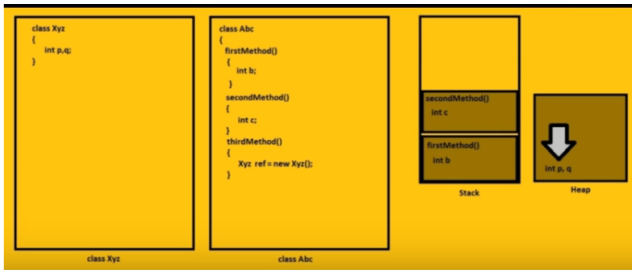
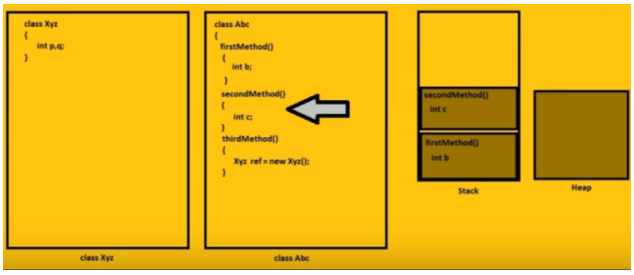
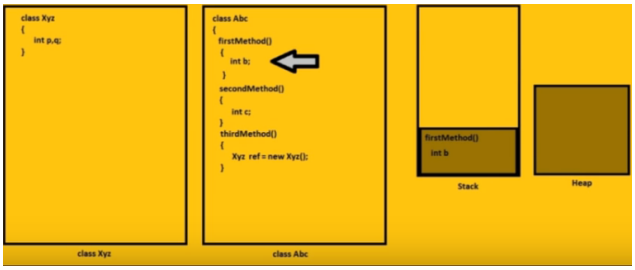


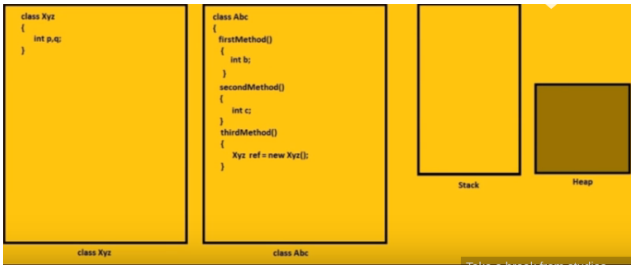
Memory management in Java is separated into two parts.

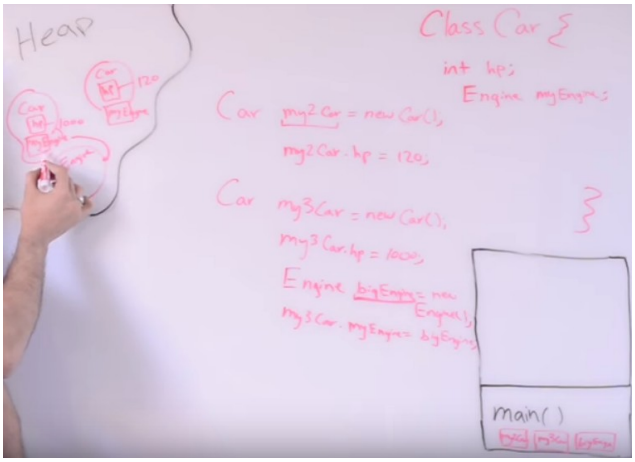
1. Stack Memory that's where local variables and Method calls are allocated.
2. Heap Memory that's where instances of classes are allocated.

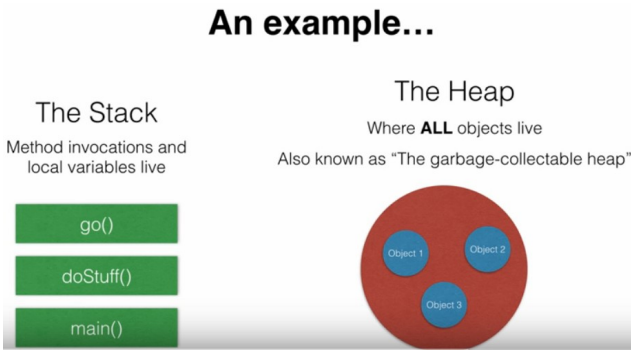












Instance Variables

- Instance variables are declared inside a class but not inside a method. They represent each individual field that an object has.
- Every Car has a wheels instance variable.

```
public class Car {
    int wheels;
}
```

Local Variables

- Local variables are declared inside a method, including method parameters. They're temporary, therefore they only live as long as the method is on the stack—in other words, as long as the method has not reached the closing curly brace.

```
public void foo(int x) {
    int i = x + 3;
    boolean b = true;
}
```

Methods are stacked

Call stack

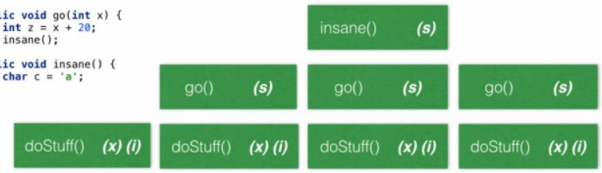
This holds two methods

When you call a method, the method lands on top of the stack. The stack frame holds the state of the method, including which line of code is executing, and the values of all local variables.

The method on the top of the stack is the method that is currently executing.

An example of a stack

```
public void doStuff() {
    boolean b = true;
    go(3);
}
public void go(int x) {
    int z = x + 20;
    insane();
}
public void insane() {
    char c = 'a';
}
```

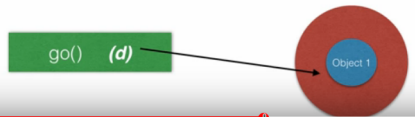


Local variables that store objects?

- Firstly, let's remember that variables are nothing but reserved memory locations to store values. This means when you create a variable you are reserving space in memory.
- Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.
- There are two data types available in Java: **Reference/Object Data Types**

Referencing an object

- A non-primitive variable holds a *reference* to an object, not the object itself. Remember that objects live on the heap!
- The *reference* to the object is stored on the stack!



- A non-primitive variable holds a *reference* to an object, not the object itself. Remember that objects live on the heap!
- The *reference* to the object is stored on the stack!

Creating objects

```
Car myCar = new Car();
```

- Make a new reference variable of a class or interface type.
- Assign the object to the reference!
- Are we calling the Car() method? No. We're calling the Car constructor.
- Looks like a method but isn't. It runs when you instantiate an object.

I didn't write any constructor?

```
public Car() {

}
```

- We generally will write a constructor (the same name as the class).
- If we don't write a constructor, the compiler will create a default constructor that's empty.
- Where's the return type? If this was a method we would have to put the return type between public and Car()!
- Constructors can be overloaded!

The Heap (more important things) - P1

- When a Java program starts, Java Virtual Machine gets some memory from Operating System.
- Java Virtual Machine or JVM uses this memory for all its need and part of this memory is call java heap memory.
- Whenever we create object using new operator or by any another means object is allocated memory from Heap and When object dies or garbage collected, memory goes back to Heap space in Java.

The Heap (more important things) - P2

JVM Option	Meaning
-Xms	Initial Java Heap size
-Xmx	Maximum Java Heap size
-Xmn	The size of the heap

- Good practise for big product projects to set the minimum -Xms and maxium -Xmx value.
- Heap size does not determine the amount of memory your process uses.
- For efficient garbage collection, the -Xmn value should be lower than the -Xmx value. Heap size does not determine the amount of memory your process uses.

Let's talk about memory - P1

- If there is no memory left in stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space.
- Garbage collection aims to collect unreachable objects in heap.
- Everything that reachable from every stack of every thread should be considered as live.
- There is also some other root set references like Thread objects and some class objects.

Let's talk about memory - P2

- If you are using Recursion, on which method calls itself, You can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is lot lesser than size of heap memory in Java.
- Variables stored in stacks are only visible to the owner Thread, while objects created in heap are visible to all thread. In other words stack memory is kind of private memory of Java Threads, while heap memory is shared among all threads.

Most important things to remember

- We only care about two types of memory: the Stack and the Heap.
- Instance variables are declared inside a class, local variables are declared inside a method or parameter.
- All local variables live on the stack.
- Object reference variables live on the stack but as a reference to the object!
- All objects live on the heap!

https://www.youtube.com/watch?v=g67agnEPmnA&t=4s

https://www.youtube.com/watch?v=UcPuWY0wn3w

https://www.youtube.com/watch?v=clOUdVDDzIM&t=84s