Java Stack and Heap: Java Memory Allocation Tutorial

What is Stack Memory?

Stack in java is a section of memory which contains methods, local variables, and reference variables. Stack memory is always referenced in Last-In-First-Out order. Local variables are created in the stack.

What is Heap Memory?

Heap is a section of memory which contains Objects and may also contain reference variables. Instance variables are created in the heap

Memory Allocation in Java

Memory Allocation in Java is the process in which the virtual memory sections are set aside in a program for storing the variables and instances of structures and classes. However, the memory isn't allocated to an object at declaration but only a reference is created. For the memory allocation of the object, new() method is used, so the object is always allocated memory on the heap.

The Java Memory Allocation is divided into following sections :

1. Heap
2. Stack

3. Code
4. Static

This division of memory is required for its effective management.

- The **code** section contains your **bytecode**.
- The **Stack** section of memory contains **methods, local variables, and reference variables.**
- The **Heap** section contains **Objects** (may also contain reference variables).
- The **Static** section contains **Static data/methods**.

Difference between Local and Instance Variable

Instance variable is declared **inside a class but not inside a method**

```
class Student{
int num; // num is  instance variable
public void showData{}
```

Local variable are declared **inside** a **method including** method **arguments**.

```
public void sum(int a){

int x = int a +  3;

// a , x are local variables;

}
```

Difference between Stack and Heap

Watch Video At: https://youtu.be/450maTzSIvA

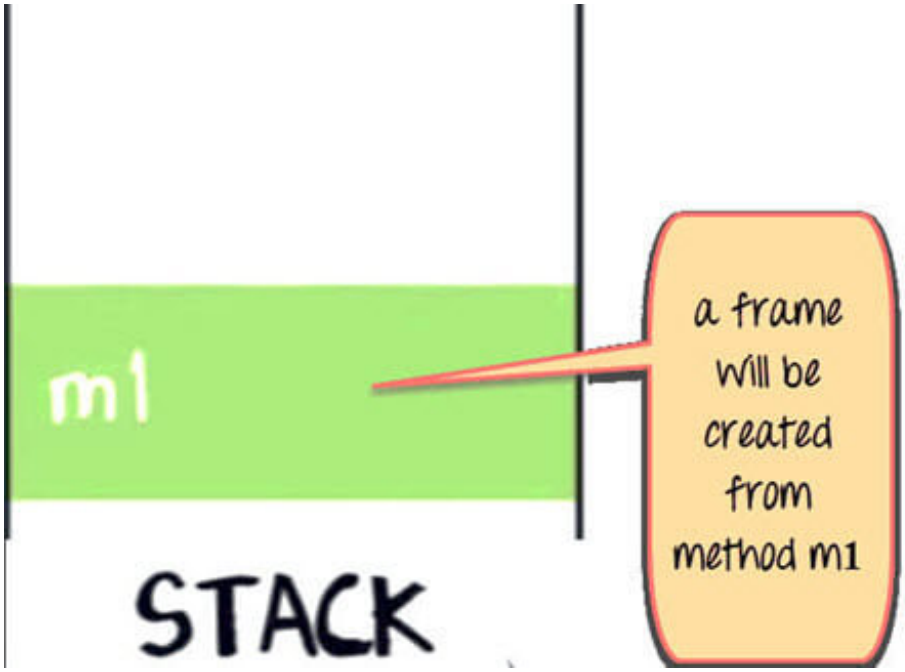
Click here if the video is not accessible

Let's take an example to understand this better.

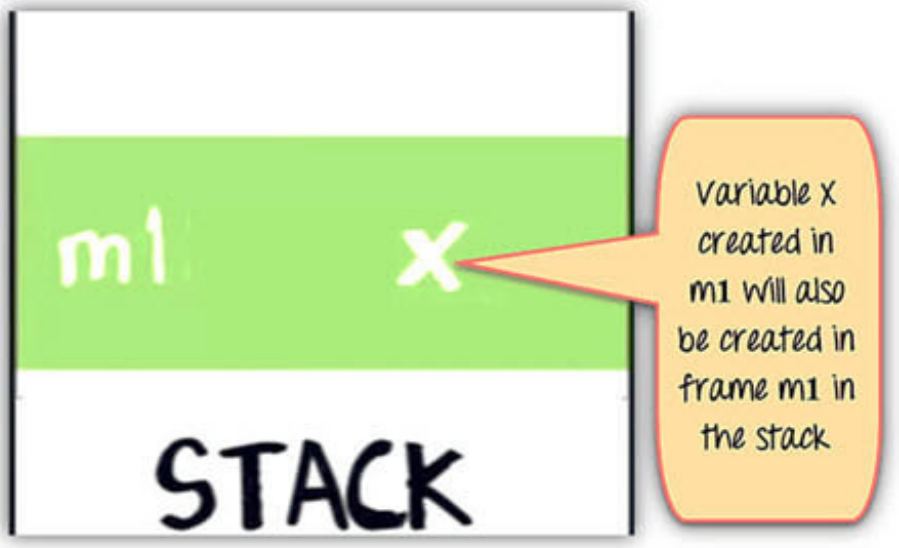
Consider that your main method calling method m1

```
public void m1{
int x=20
}
```

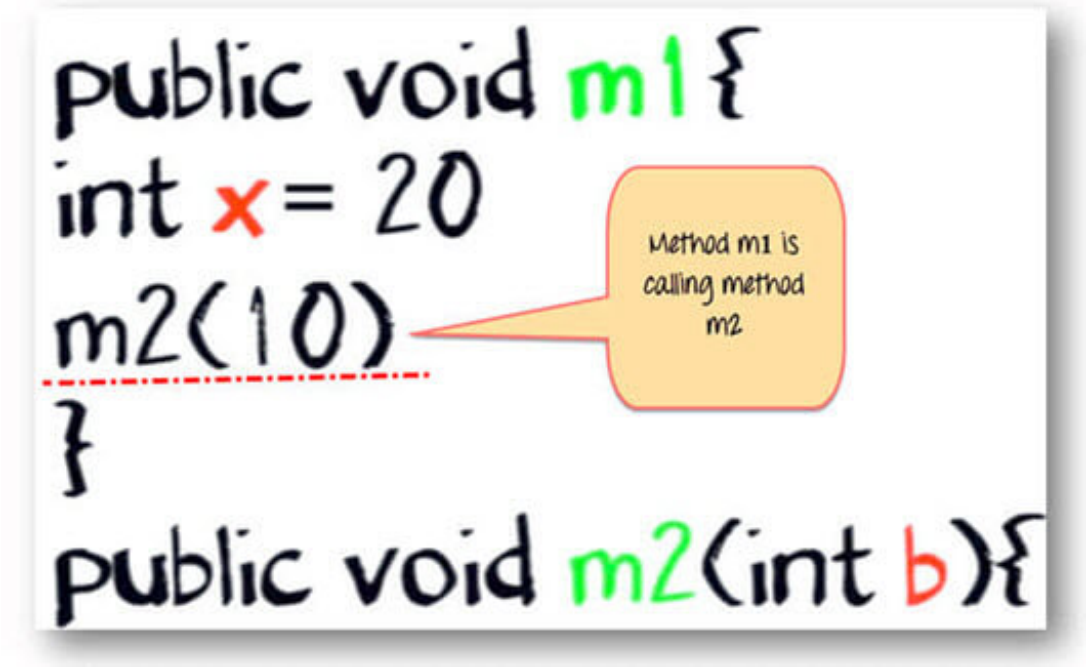
In the stack java, a frame will be created from method m1.

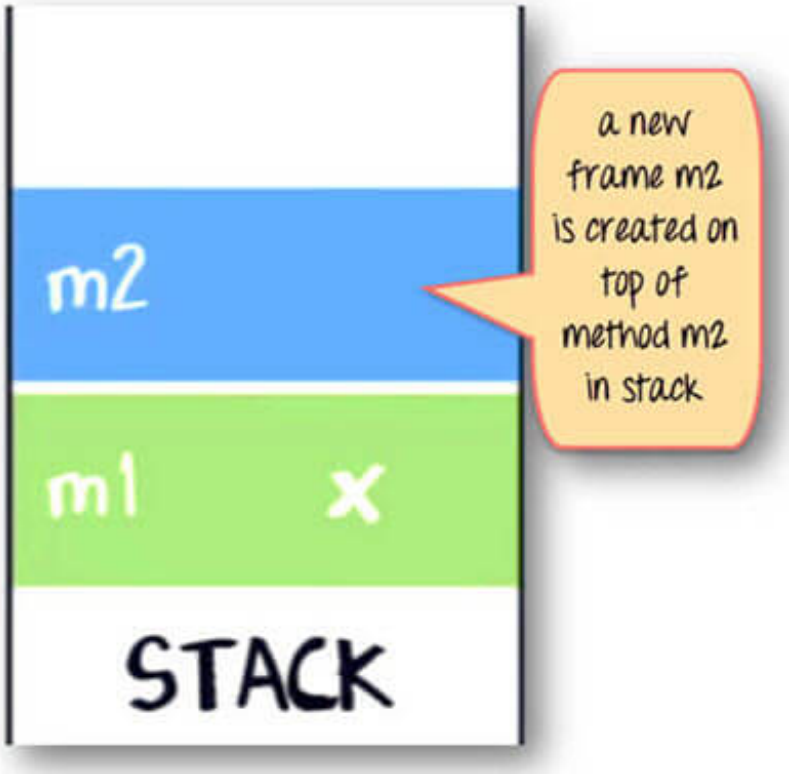


The variable X in m1 will also be created in the frame for m1 in the stack. (See image below).



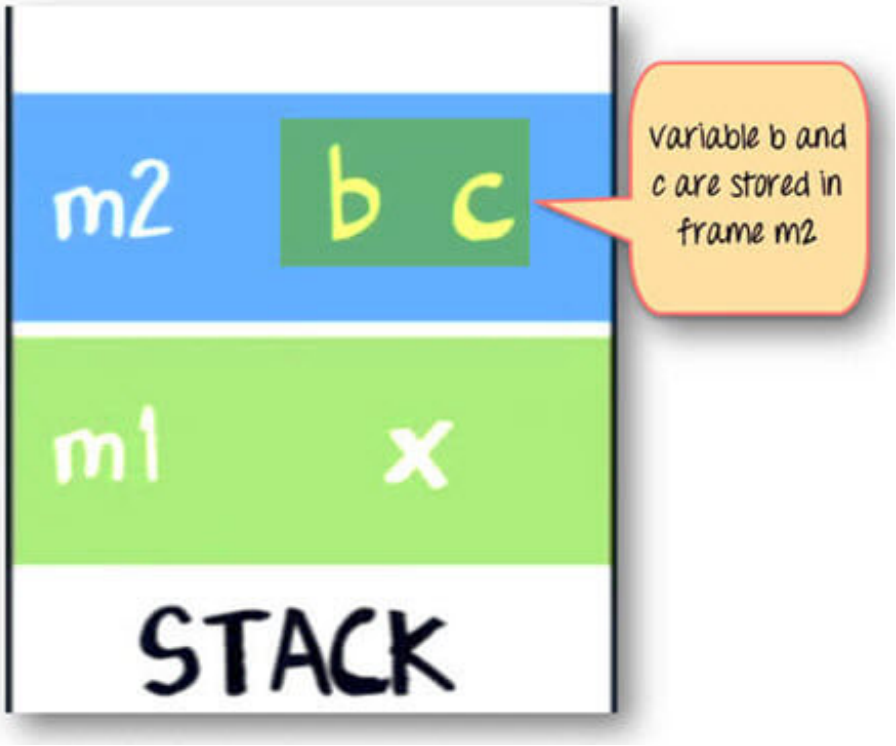
Method m1 is calling method m2. In the stack java, a new frame is created for m2 on top of the frame m1.



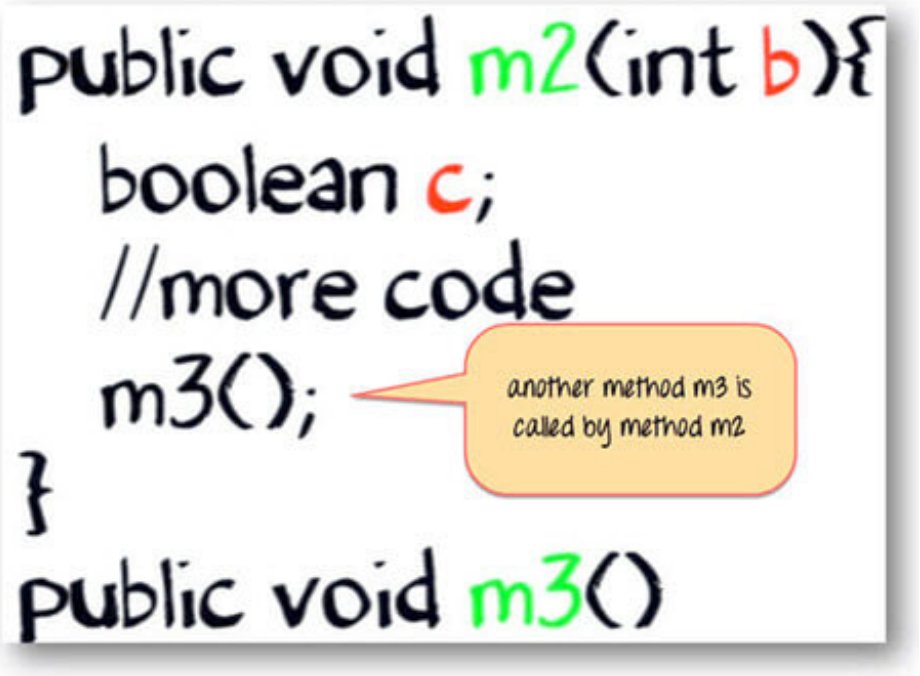


Variable b and c will also be created in a frame m2 in a stack.

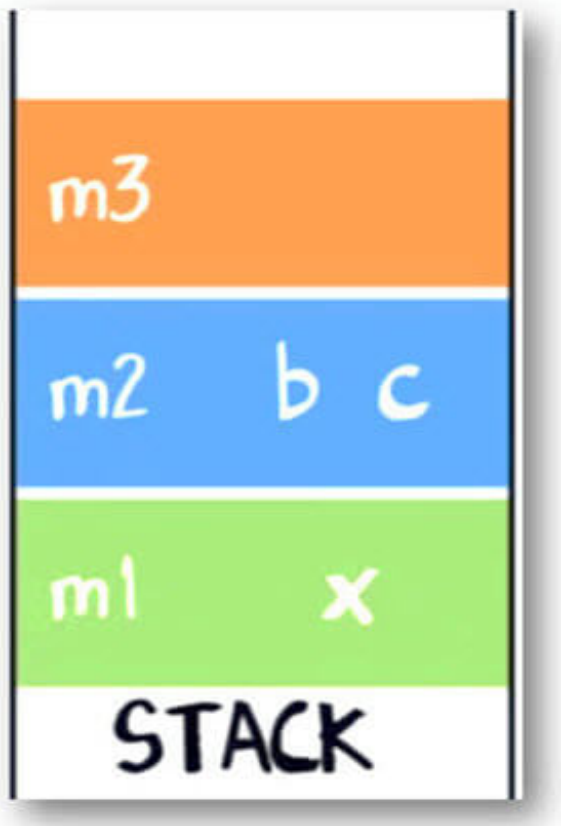
```
public void m2(int b){
boolean c;
}
```



Same method m2 is calling method m3. Again a frame m3 is created on the top of the stack (see image below).



Now let say our method m3 is creating an object for class "Account," which has two instances variable int p and int q.

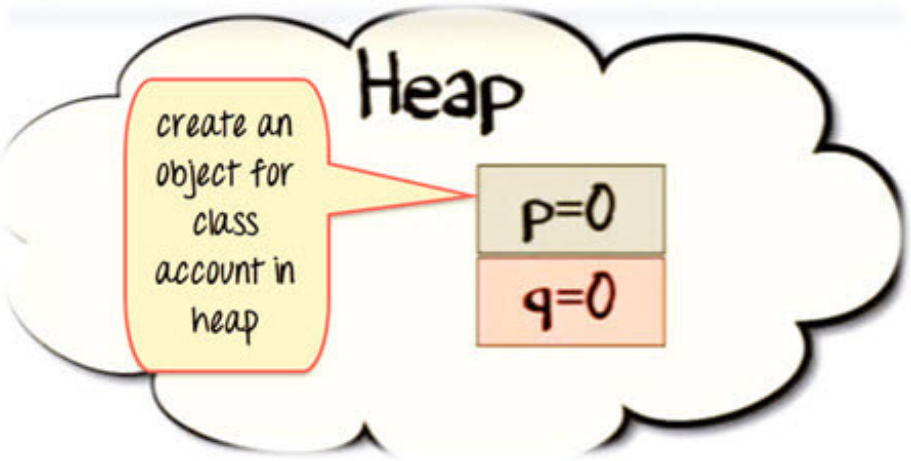



```
Account {
        Int p;
        Int q;
    }
```

Here is the code for method m3

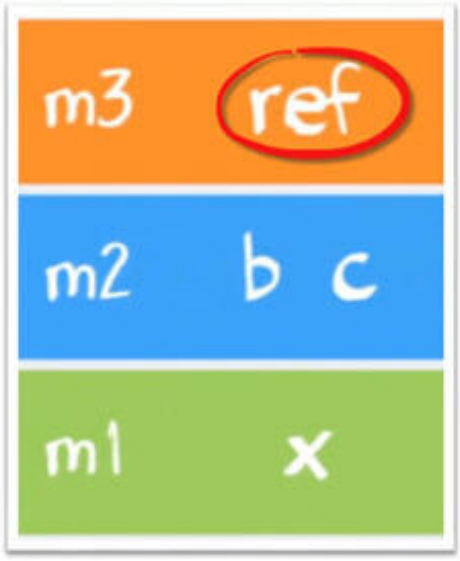
```
public void m3(){
 Account ref = new Account();
 // more code
}
```

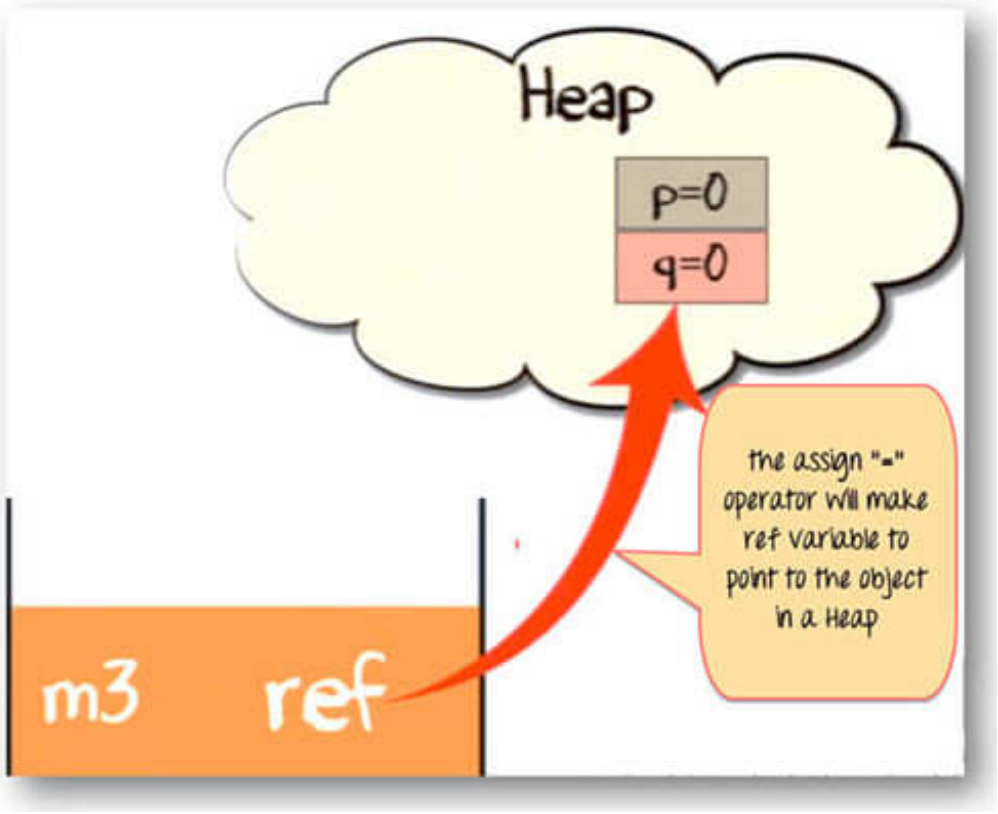
The statement new Account() will create an object of account in heap.



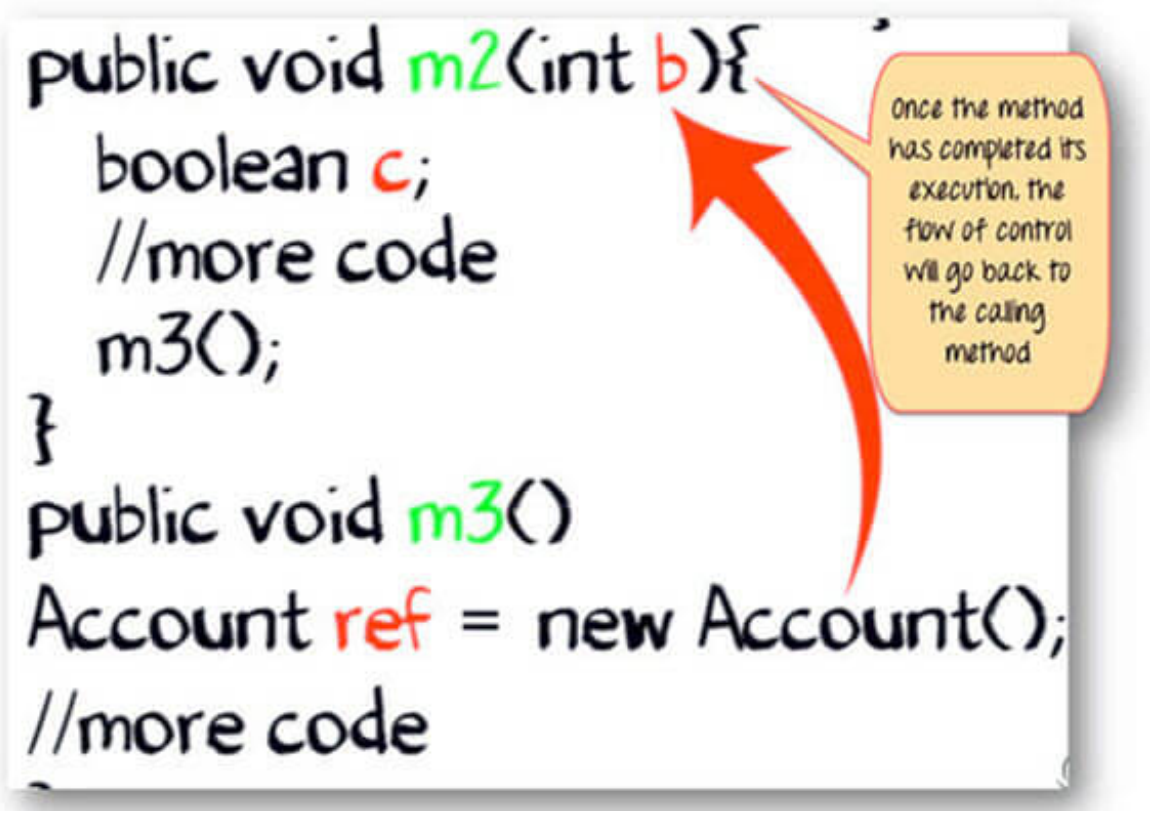
The reference variable "ref" will be created in a stack java.

The assignment "=" operator will make a reference variable to point to the object in the Heap.

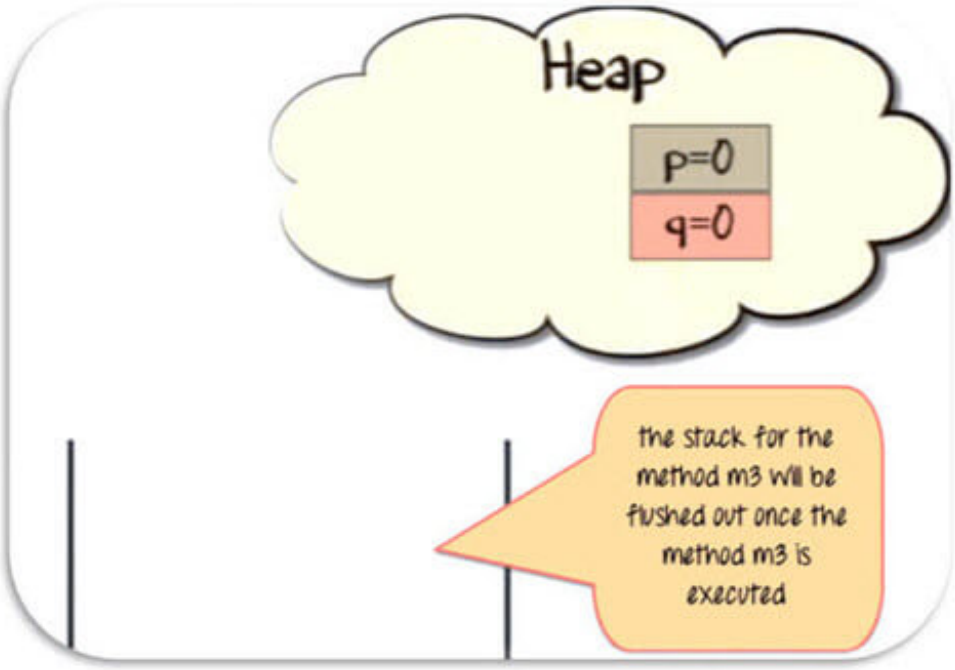




Once the method has completed its execution. The flow of control will go back to the calling method. Which in this case is method m2.



The stack from method m3 will be flushed out.



Since the reference variable will no longer be pointing to the object in the heap, it would be eligible for garbage collection.

Once method m2 has completed its execution. It will be popped out of the stack, and all its variable will be flushed and no longer be available for use.

Likewise for method m1.

Eventually, the flow of control will return to the start point of the program. Which usually, is the "main" method.

What if Object has a reference as its instance variable?

```
public static void main(String args[]) {
  A parent = new A(); //more code } class A{ B child = new B(); int e; //more code } class B{ int c;
int d;  //more code }
```

In this case , the reference variable "child" will be created in heap ,which in turn will be pointing to its object, something like the diagram shown below.

Summary:

- When a method is called, a frame is created on the top of the stack.
- Once a method has completed execution, the flow of control returns to the calling method and its corresponding stack frame is flushed.
- Local variables are created in the stack
- Instance variables are created in the heap & are part of the object they belong to.
- Reference variables are created in the stack.

- Prev
  Report a Bug
- Next