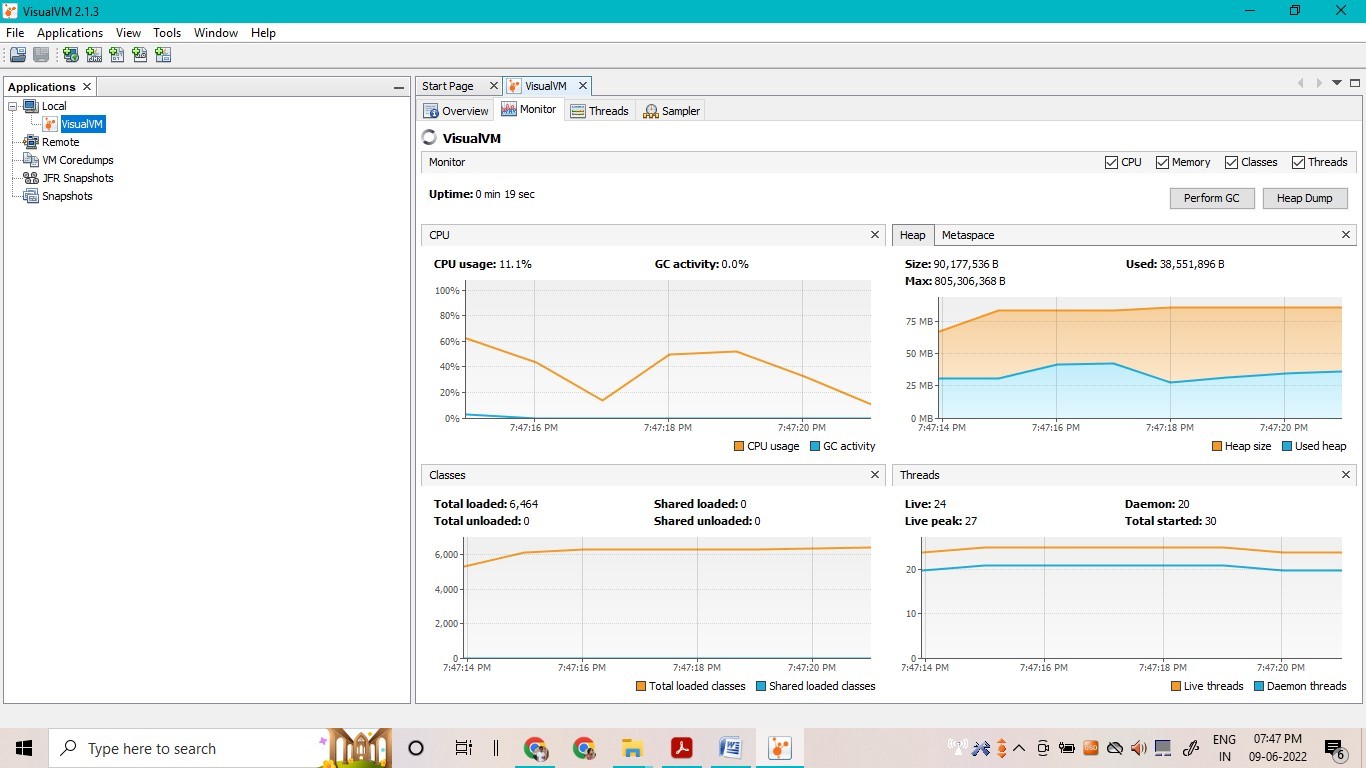
# Program No.4: Install memory monitoring tool and observe how JVM allocates Memory.

* **VisualVM** provides detailed information about Java applications while they are running on the Java Virutal Machine (JVM). VisualVM's graphical user interface enables us to quickly and easily see information about multiple Java applications.
* **Installing VisualVM**
  + Download the VisualVM installer from the VisualVM project page.
  + Extract the VisualVM installer to an empty directory on local system.
* **Starting VisualVM**
  + To start VisualVM on Windows, run the **visualvm.exe** program that is in the \bin folder under the VisualVM install folder.
  + After opening VisualVM, double click on VisualVM icon which appears on left panel.
* **Exploring VisualVM:**
  + Click on overview tab to know more about JVM arguments, System Properties etc.
  + Click on monitor tab to understand about CPU usage, Heap Space, Classes and Threads. Observe how memory is managed by JVM.
  + Click on Heap Dump(right-side corner) in monitor tab to know about classes, instances, and environment.
  + Click on threads to know how many types threads are running in live and finished threads.
  + Click on sampler to know CPU samples, Memory Samples.



# Program No.5: Explain memory allocation through Java programs.

**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 variables** 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**

* Let’s take an example to understand this better.
* Consider that the main method calling method **m1**

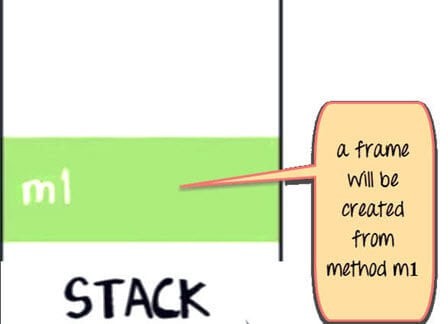
public void m1

{

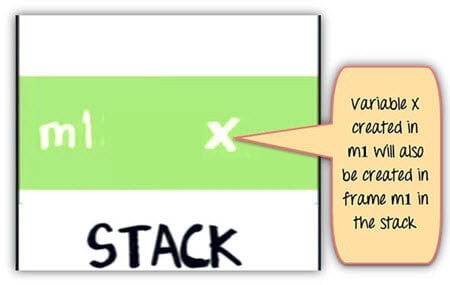
int x=20;

}

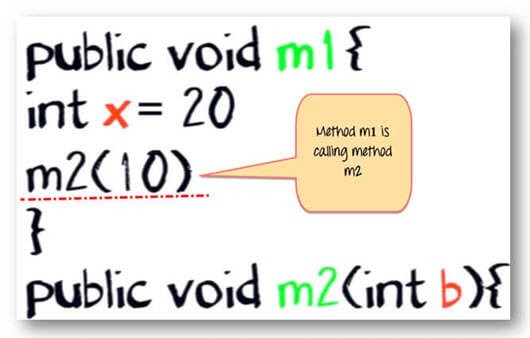
In the stack java, a frame will be created from method m1.

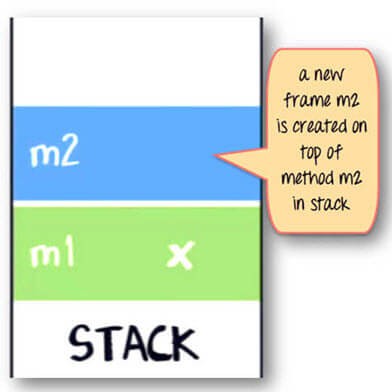


The variable **x** in m1 will also be created in the frame for m1 in the stack. (See image below).



As shown below Method m1 is calling method m2. In the stack Java, a new frame is created for m2 on top of the frame m1.



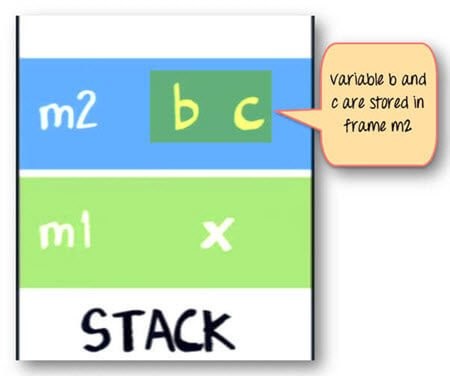


Variable b and c will also be created in a frame m2 in a stack. public void m2(int b)

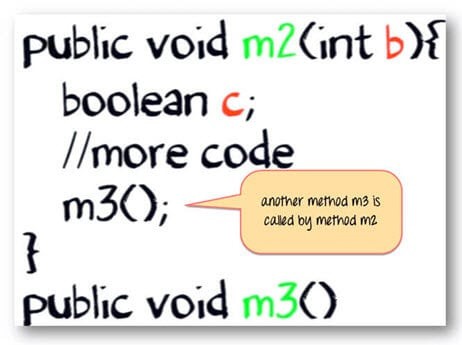
{

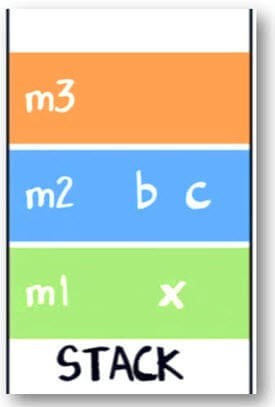
boolean c;

}



As shown below 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 variables int p and int q.**

Account

{

int p; int q;

}

Here is the code for method m3

public void m3()

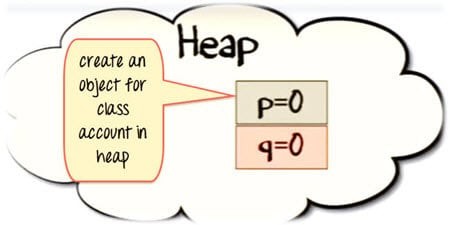
{

Account ref = new Account();

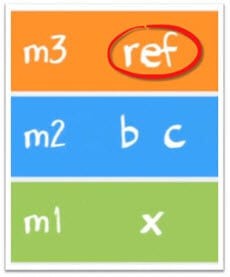
// more code

}

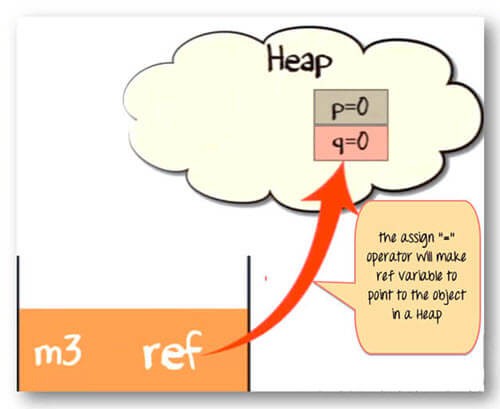
The statement new Account() will create an object of account in **heap.**



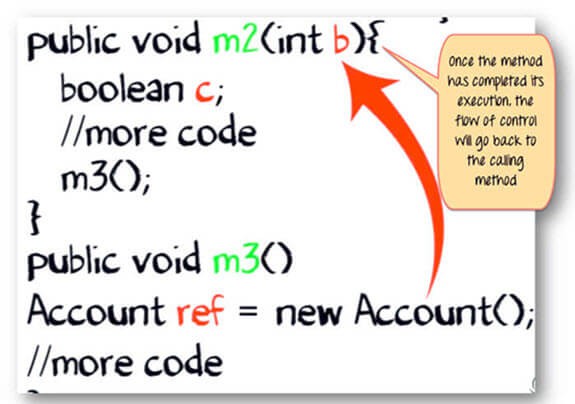
The reference variable “ref” will be created in a stack java.



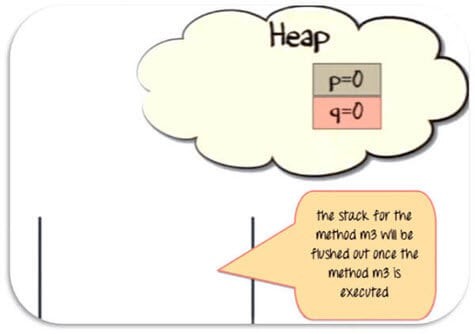
The assignment “=” operator will make a reference variable to point to the object in the Heap.



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.

**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.