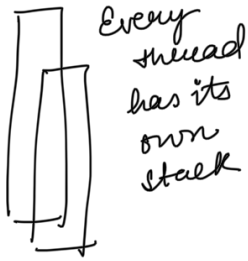
# Java Memory Model.

## The Stack



Every thread has its own stack

## The Heaps.

Data that has a longer lifetime than a single code block.
For eg. Objects that need to be shared access multiple methods.

→ In other languages you must state when an object no longe needed by calling a method like free()
Objects that are not freed continue to consume memory.
This is called memory leak
Overtime more and more memory of computere it is runni on will get used up, eventually leading to system crash.

→ Java avoids memory leaks by.

- Running on virtual machine. new key word does not allocat memory from O.S. The memory is acquired by virtual machine. VM controls allocation and freeing of object on its own

- Adopts a Garbage collection strategy. Here when a heap exammer asks for objects to be allocated on heap

do not need to free them later. It is deleted by automatic program by JVM.
Any object on heap which cannot be reached through a reference from stack is eligible for "Garbage collection"

→ gc

```
public static void gc()
```

Runs garbage collector
Calling gc method suggests that the JVM spend effort toward recycling unused objects to make the memory they currently occupy available for quick reuse. When control returns from method call, JVM has made a best effort to reclaim space from all discarded objects.

```
System.gc()
Runtime.getRuntime().gc()
```

→

```
Runtime runtime = Runtime.getRuntime();
long availableMemory = runtime.freeMemory();
```

OR

```
Sysout( runtime.freememory());
```

Just a suggestion to run garbage collector, but it's not a guarantee

It is not recommended to run gc command. Only tell garbage collector to run :-
- To evaluate whether two different but alternative codes are more/less efficient than each other.
There is no guarantee that gc will run garbage collector for sure. Till the garbage collector runs, application will be suspended and will starts once it is done.

After garbage collection removes object from heap, it runs finalize() method

```
protected void finalize() throws Throwable
```

→ Soft Leaks are objects being referenced on the stack even though it will never be used again.

Multithreaded program often produce errors when multiple threads try to access same resources.