**JAVA MEMORY MANAGEMENT:**

Everything (classes,methods,variables etc..) we create in java is about memory.

Java memory is divided into 2 parts.

**Heap:**

It is used to store class objects. It stores all the default JRE classes. GC runs on heap memory only to free some memory by destroying some objects which don’t have any references.

**Java Stack Memory:**

It is used for execution of threads.

It contains method specific values; local variables are stored here. All the object references referring heap memory are stored in stack. Memory will be stored in LIFO format.

Stack memory size is very less compared to heap memory.

Object will be stored inside heap memory, but reference will be stored inside stack memory.

Heap memory is divided into 2 parts.

1st part is called Young generation.

2nd part is called old generation.

When we create class object it will be stored in young generation space.

Young generation memory is divided into 3 parts.

1. Eden
2. survivor (s0)
3. survivor (s1)

When we create a new object it will be stored in eden. When eden is full minor gc will be performed automatically.

Some objects will be shifted from eden to S0. When S0 filled up it goes to S1.

Young Generation has less number of objects compared to Old Generation.

Old Generation Memory:

When S0, S1 is filled, then new objects are created in old generation. When this memory is full major gc is called.

This major gc takes longer time compared to minor gc.

Some times we get timeout error because when major gc is full and it is busy in destroying objects, and cpu utilization is high. Major gc activity should be minimized.

There is another part of memory is permenant generation. MetaData of classes and methods is stored here. This will be generated at the run time.

Inside permenant generation there is another memory called method area.

In method area class structure, static variables, constants will be stored.

There is another area in heap memory that is memory pool. All the immutable objects will be stored in memory pool. This will be defined at the run time by jvm memory managers.

There is run time constant pool inside method area in permenant memory. It is used to store static variables and constants.

**Garbage collection:**

In old languages like c++, programmer is responsible to create new object, and to destroy useless objects.

Usually programmenr is taking very much care while creating objects and neglecting destruction of useless objects because of his neglectence at certain point for creation of new object sufficient memory may not be available. (because total memory filled with useless objects only) and total application will be down with memory problems, hence out of memory error is very common problem in old languages like c++.

But in java, programmer is responsible onl;y for creation of objects and programmer is not responsible to destroy useless objects.

Sun people provided one assistant to destroy useless objects, this assistant is always running in the background( demon thread), and destroy useless objects, just because of this assistant the chance of failing java program with memory problems is very very low. This assistant is nothing but garbage collector.

When a object will be available for gc?

1. By nullifiying the reference.
2. By assigning reference to another object.
3. By anonymous objects.

2 methods used in gc():

1. Finalize():

Invoked each time before object is garbage collected.

Found in object class.

1. Gc():

Invoke garbage collection. Found in system and runtime class.

Types of garbage collections in java:

1. Serial GC
2. Parllel GC
3. Parllel old GC
4. Concurrent Mark-Sweep collector
5. GI Garbage Collector

**Serial GC :**

This is useful in client machines such as our simple stand alone applications and machines with smaller CPU.

**Parallel GC :**

Parallel GC is also called as throughput collector because it uses multiple CPU's to increase GC performance.

**Parallel Old GC :**

This is same as Parallel GC except that it uses multiple threads for both Young Generation and Old Generation garbage collection.

**Concurrent Mark Sweep Collector :**

CMS Collector is also referred as concurrent low pause collector. It does the garbage collection for Old generation. CMS collector tries to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads.

**G1 Garbage Collector:**

This is available from Java7, Its goal is to replace the CMS collector. G1 collector is a parallel, concurrent and incrementally compacting low pause garbage collector. This removes the concept of Young and Old generations. The heap is divided into equal sized regions. When garbage collector is invoked it first collects the region with lesser live data.