**GARBAGE COLLECTION INTRODUCTION**

\*\*Garbage collection is required to collect all the unused objects through Garbage Collector.

\*\* Garbage Collector in java is running always in the background to destroy useless object.

Background executing threads are known as Daemon Threads. Garbage collector is an example of daemon threads.

\*\* Java is a robust language i.e chance of failing it is very less . Garbage collector is also a backbone for this feature.

\*\*If the object has no reference then it is eligible for Garbage Collection.

**MAKING AN OBJECT READY FOR GARBAGE COLLECTION**

1. Nullify the reference variable pointing to the object then the object will be eligible for Garbage Collection.
2. Reassigning the reference variable if the reference variable pointing to one object is reference to other object the the first dereference object is eligible for GC.
3. Objects created inside a method is eligible for GC when method execution completes as inside main methods references pointing to that objects are local variables and once out of the method those variables does not exists so the objects are eligible for GC.

But if inside the method we have created a object that is reference by global static objects then that object will not be eligible for GC and also if the method returns some object which is referenced by some reference variable in calling method then also the Object created inside a method is not eligible for GC.

1. **Islands of Isolation :🡪**

**Class test**

**{**

Test i;

Public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

Test t3=new Test();//at this point no object is eligible for gc

t1.i=t2;

t2.i=t3;

t3.i=t1;//at this point also no object is eligible for gc

t1=null//at this point no object is eligible for gc as t1 there are links for all the objects

t2=null;

t3=null//at this point every object is eligible for gc as there is no external link to any of the object although all the objects have link between themselves i.e they are isolated linked object so they all are eligible for GC.

**THE METHODS FOR REQUESTING JVM FOR CALLING GARBAGE COLLECTOR**

1. By using System.gc() we can request JVM to run Garbage collector . gc() is the static method in the System class.
2. By using runtime class Runtime.gc().

But it is not guaranteed that the JVM will listen to this request but most of the time it listens to the request.

\*\* A java application can be communicating to JVM through Runtime Class object.

Runtime class is present in java.lang package and it is a Singleton class.

A singleton class means only one object of this class can be created. Through factory method we can create Runtime object:🡪

Runtime r=Runtime.getRuntime(); //Here getRuntime() is a static factory method.

Then we can call **r.freeMemory()** 🡪 to get the total free memory

**r.totalMemory()** 🡪 to get the total memory

**r. gc()** 🡪 to call the GC.

**\*\*gc() method present in System class is static method.**

**\*\* gc() method present in Runtime class is instance method.**

**Which one is the best to use 🡪**

1)Convenience wise System.gc() is better as we don’t have to create object for that.

2) performance wise Runtime.getRuntime().gc();

**Internally System class gc() method is calling Runtime.getRuntime().gc();** So it is better to call 2nd type performancewise.

**THE METHODS POSTMORTERM ON FINALIZATION**

After garbage collector came to destroy useless objects it calls finalize() method on the to be destroyed object to perform cleanup activities before destruction. This process is called finalization.

Clean up activities is actually resource deallocation, closing connection.

\*\*For any object finalize() method is available , it is inside Object class. Syntax is 🡪

Protected void finalize() throws Throwable

{

}

In object class finalize() method has empty implementation. But not abstract method, empty implementation is there.

**\*\*Case 1 :🡪** **Before garbage collection the finalize method of the destroying object will be called by GC only.**

When we call System.gc() then one different thread is created which is responsible for calling finalize() method on the destroying object . Main thread is executed parallel. It has no dependency on System.gc() e’s thread.

**\*\*Case 2 :🡪**  **We can call** finalize **method explicitly then it will be called as normal method object will not be destroyed.**

**\*\*Case 3:🡪**If the programmer call finalize() method explicitly and any uncaught exception occur then the program will be terminated due to the exception.

But while GC calls the finalize() method and any exception occurs the exception will be ignored program will not be terminated.

But if the catch block is there then in both cases catch block will be executed. i.e JVM ignores only uncaught exception if finally is called by GC .

**\*\*Case 4:🡪 Garbage collector calls finalize method only once although the same object is eligible for garbage collection multiple times.**

**\*\*Case 5:🡪** If we don’t explicity request GC to call finalize() method then also if memory problem comes garbage collector will be called and it’s finalize method.

**Out Of Memory Case**

If programmer create too many objects with valid reference then although the JVM calls GC the GC cannot do anything as they have reference. Such type of objects which our program is not using but as they have some reference they are not also eligible for Garbage collection are called Memory Leaks.

If memory leaks are there in our program then after some time program will stop and Out of memory error will come and this is the error made by Programmer only.

There are some Third party tool to find memory leaks.

**Mark and Sweep Algorithm for Garbage collection**

Most of the garbage collector follow mark and sweep algorithm.

(Have To Study)