Garbage Collection

① Introduction :-

- In old Languages like C++, Programmer is responsible for Both Creation and Destruction of objects. Usally Programmer taking very much care while Creating objects and neglecting Destruction of useless objects. Due to his negligence at Creation point, for Creation of New object, sufficient Memory may not be Available and Entire Application will be Down with Memory Problems. Hence OutOfMemoryError is very Common Problem in old languages like C++;
- But in Java, Programmer is Responsible only for Creation of objects and he is Not Responsible for Destruction of useless objects. SUN People Provided one Assistant which is always Running in the Background for Destruction of Useless Objects. Just because of Assistant, the chance of failing Java Program with Memory Problems is very very less (It is also one reason for Robustness of JAVA). This Assistant is Nothing but Garbage Collector.
- Hence the Main objective of Garbage Collector is to Destroy useless objects.
- Garbage Collector is best example for Deamon Thread as it is always running in the background.

The ways to Make an object Eligible for GC :-

- Even though Programmer is Not Responsible to Destroy useless objects but it is Highly Recommended to make an object Eligible for GC. if it is No Longer required.
- An object is said to be Eligible for GC if and only if it doesn't Contain any Reference.

The following are various Possible ways to make an object Eligible for GC.

① Nullifying the Reference Variable :-

If an object is No longer required, then Assign null to all its Reference Variables, Then that all object Automatically eligibe for Garbage Collection.

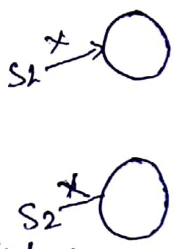
Example :-

No objects are Eligible for GC

```
Student s1 = new Student();
Student s2 = new Student();
    :
    :
s1 = null;
    :
    :
s2 = null; //2nd object is also Eligible for GC
```

one object is Eligible for GC

2nd object is also Eligible for GC



② Re Assigning the Reference Variable ;-

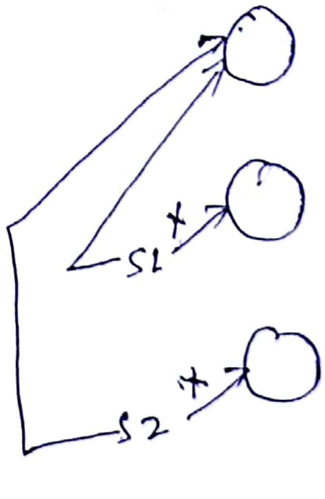
If an object is No longer Required then ReAssign its Refence variable to any other object then old object is Automatically Eligible for GC.

No Objects eligible for GC.

```
Student s1 = new Student();
Student s2 = new Student();
    :
    :
s1 = new Student();
    :
    :
s2 = s1;
```

one object is elegible for GC

Two objects are elegible for GC

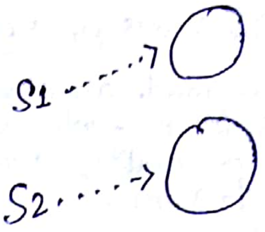


3) Objects Created Inside a Method:

objects created Inside a Method are by default Eligible for Garbage Collection. Once Method completes (Because the Reference Variables are Local Variables of that Method and once Method completes all local variables will be destroyed).

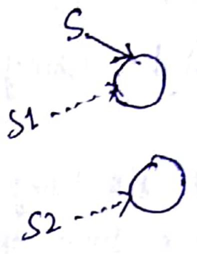
After m1() Two objects are eligible for GC

```
class Test {
    Public static void main(String[] args){
        m1();
    }
    Public static void m1() {
        Student s1 = new student();
        Student s2 = new student();
    }
}
```



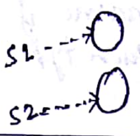
After m1() only 1 object is elegible for GC

```
class Test{
    Public static void main(String [] args){
        Student s = m1();
    }
    ~~Student s2~~
    Public static Student m1(){
        student s1 = new student();
        student s2 = new student();
        return s1; //Here s1 object Referenced s, s1 and s2 is Not Pointing After
                   //m1() execution.
    }
}
```



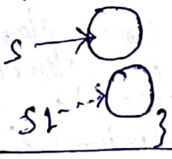
After m1() 2 objects are Elgible for GC

```
class Test {
    Public static void main (String args[]){
        m1();
    }
    Public static Student m1(){
        student s1 = new student();
        student s2 = new student();
        return s1;
    }
}
```



After m1() only 1 object is eligible for GC

```
Class Test {
    Static Student s;
    Public static void main(String[] args){
        m1();
    }
    public static void m1(){
        student s1 = new student();
        S = new Student();
    }
}
```



---

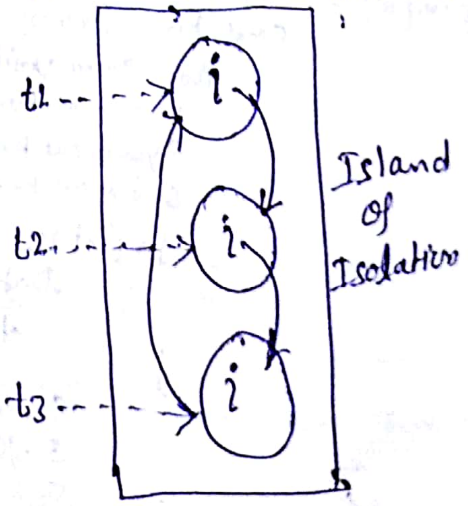
4) Island of Isolation :-

here total group of object Isolated from outside world. Eventhough internal references are there but from outside world there is no any reference.

No object is elegible for GC

All 3 objects are elegible for GC

```
class Test{
    Test i;
    Public static void main(String [] args){
        Test t1 = new Test();
        Test t2 = new Test();
        Test t3 = new Test();
        t1.i = t2;
        t2.i = t3;
        t3.i = t1;
        t1 = null;
        t2 = null;
        t3 = null;
    }
}
```



Note:- 1. If an object doesn't have any reference variable then it is always eligible for GC

2. Even though object having Reference variable still sometimes it may be a Eligible for Garbage Collection (If All References are Internal References)

Eg: Island of Isolation.

★ The ways for requesting JVM to Run Garbage Collector: ★

- once we made an object elegible for GC, It may not Destroy Immediately by the Garbage Collector. Whenever JVM Runs Garbage Collector then only object will be Destroyed. But when exactly JVM runs GC, we can't Expect. It Depends on JVM and varied fo JVM to JVM.
- Instead of waiting until JVM Runs GC, we can Request JVM to Run Garbage Collector. But there is No Guarantee whether JVM accept our request OR Not. But most of the times JVM accepts our request.

The following are various ways for requesting JVM to Run Garbage Collector:-

1) By Using System Class:- System class contains a static Method gc() for the purpose.

$\boxed{\text{System.gc();}}$

2) By Using Runtime Class:-

- A Java Application can communicate with JVM by using Runtime object.
- Runtime class present in java.lang Package and it is a Singleton Class.
- we can create a Runtime object by using getRuntime().

$\boxed{\text{Runtime r = Runtime.getRuntime();}}$

once we got the Runtime object we can call the following methods on that object.

1) freeMemory():- Returns Number of Bytes of free Memory present in the Heap.
2) totalMemory():- Returns Total Number of Bytes of Heap (i.e Heap Size).
3) gc();- Requesting JVM to Run Garbage Collector.

Note1:- Singleton Class:- for any Java Class if we are allowed to create only one object, such type of class is called Singleton class.
Eg:- Runtime, BusinessDelegate, ServiceLocater

Note2:- Factory Method:- By using class name if we are calling a Method and if that method returns same class object such type of methods are called factory Methods (static factory methods).

```
Eg:- Runtime r = Runtime.getRuntime();
     DateFormate df = DateFormat.getInstance();
```

Example!-
```
Import java.util.Date;
class RuntimeDemo{
    Public static void main(String[] args){
        Runtime r= Runtime.getRuntime();
        System.out.println(r.totalMemory()); //16252928
        System.out.println(r.freeMemory()); //15956808
        for(int i=0; i<10000; i++){
            Date d = new Date();
            d=null;
        }
        System.out.println(r.freeMemory()); // 15772752
        r.gc();
        System.out.println(r.freeMemory()); // 16074976
    }
}
```

Note:- gc() method present in System class is static method where as gc() method present in Runtime class is Instance Method.

Q. which of the following is valid way for requesting JVM to Run Garbage Collector?

1) System.gc(); ✓
2) Runtime.gc(); ✗ [b'z gc() method present in Runtime class is not static]
3) new Runtime().gc(); ✗ [b'z we can not create object by using new operator as Runtime class is a Singleton class]
4) Runtime.getRuntime().gc(); ✓

Note:- with Respect to Convenience, it is Recommended to use System.gc() method when compared to Runtime class gc() method.

- With Respect to Performance, It is Recommended to use Runtime class gc() method when compared with System class gc() method, because Internally System.gc() method calls Runtime class gc() method.

```
class System {
    Public static void gc() {
        Runtime.getRuntime().gc();
    }
}
```

Finalization:

- Just Before Destroying an object Garbage Collector call finalize() to Perform cleanup Activities. once finalize() Completes Automatically GC Destroys that object.
- finalize() Present in Object class with the following Prototype.

$\boxed{\text{Protected void finalize() throws Throwable}}$

- Based on our Requirement we can Override finalize() method in our class to define our own cleanup Activities.

Case 1 • Just before Destroying an object Garbage Collector Always calls finalize() method on that object, then the corresponding class finalize() will be executed. for Example, if String Object Eligible for GC, then String class finalize() will be executed, but Not Test class finalize() method.

```
class Test {
    Public static void main(String[] args) {
        String s = new String("Durga");
        s=null;
        System.gc();
        System.out.println("End of Main");
    }
    Public void finalize() {
        System.out.println("finalize Method Called");
    }
}
```

Output:- End of Main.

- In the above example String object eligible for GC and Hence String class finalize() got executed which is empty implementation. Hence In this case output is { End of Main }.
- If we Replace string object with test object, then Test object is eligible for GC and hence Test class finalize() method will be executed. In this case output is.

```
End of Main                 OR    finalize Method Called
finalize method Called            End of Main
```

Case-2 Based on Our Requirement we can call finalize() method Explicitly, then I will be executed Just Like a Normal Method Call and object won't be Destroyed. But before destroying an Object Garbage Collector Allways Calls finalize() method.

```
class Test{
    Public Static void main(String[] args){
        Test t=new Test();
          t.finalize();
          t.finalize();
          t=null;
          System.gc();
          System.out.println(" End of main()");
    }
    Public void finalize(){
          System.out.println(" finalize() called");
    }
}
```

Output

```
finalize() called
finalize() called
finalize() called
End of main()
```

- In the Above Example finalize() got executed 3 Times, In that 2 Times explicitly by the Programmer and 1 Time by the Garbage Collector.

Note:-

- Before destroying Servlet Object, Web Container Allways Calls destroy() to Perform Cleanup Activities. But Based on our Requirement we can call destroy() from init() and service() Methods Explicitly, then It will be executed Just like Normal Method Call and Servlet Object won't be Destroyed.

Case-3:- If the Programmer Calls finalize() Explicitly and while executing that finalize() if any Exception occurs and which is Uncaught (that is there is no catch blo then the Program will be terminated Abnormally by raising that exception.

If Garbage Collector Calls finalize() method and while executing that finalize() method, If any Exception raised which is Uncaught then JVM ignores that exception and rest of the Program will be executed Normally.

```
Class Test{
    Public static void main(String[] args){
          Test t= new Test();
        // t.finalize(); →1
          t=null;
        System.gC();
        System.out.println("End of main()");
    }
    Public void finalize(){
          System.out.println ("finalize() called");
          System.out.println( 10/0 );
    }
}
```

output

```
finalize() called
End of main()
```

If we are Not Commenting Line 1 then Programmer Calls finalize() method an while executing that finalize() method ArithmeticException raised which is Uncaught. Hence the Program will be terminated abnormally by raising ArithmeticException. In this the output is

```
finalize() called
Exception in thread "main" Java.lang.ArithmeticException: / by ze
```

• If we comment Line 1 then Garbage Collector Calls finalize() method and while executing that finalize() method ArithmeticException raised which is uncaught. JVM Ignores that Exception and Rest of the program will be executed Normally.

Q. Which of the following is true?

• JVM Ignores Every Exception which are raised while Executing finalize() method. //False
• JVM Ignores only uncaught Exceptions which are raised while executing finalize method. //True.

Case-4:- On any Object Garbage Collector Calls finalize() only once, Even though that Object Eligible for GC Multiple Times.

```
class FinalizeDemo{
    static FinalizeDemo s;
   Public static void main(String[] args) throws InterruptedException{
    FinalizeDemo f=new FinalizeDemo();
     System.out.println(f.hashCode());
         f=null
       System.gc();
       Thread.sleep(5000);
     System.out.println(s.hashCode());
          S=null;
       System.gc();
    Thread.Sleep(10000);
      System.out.println("End of main()");
   }
   Public void finalize(){
        System.out.println("finalize() called");
           s=this;
   }
}
```

output

```
30090737
finalize() called
30090737
End of main()
```

In the Above Example even though Object Eligible for GC two Times, but GC Calls finalize() only once.

Case 5:- We Can't Expect Exact Behaviour of the Garbage Collector. It is JVM Vendor Dependent. It is varied from JVM to JVM. Hence for the following Questions we can't Answer Exactly.

1) Exactly at what Time JVM Runs Garbage Collector?
2) In which order Garbage Collector Identifies Eligible Objects?
3) In which order Garbage Collector Destroys the objects?
4) Whether Garbage Collector Destroys All Eligible Object's OR Not?
5) What is the Algorithm followed by Garbage Collector. Etc....

Note:- Usually whenever the program Runs with Low memory JVM will Run Garbage Collector. But We can't Expect Exactly at what time.

• Most of the Garbage Collectors follow Mark & Sweep Algorithm. But It doesn Means Every Garbage Collector follow the Same Algorithm.

```
Class Test{
        Static int Count =0;
        Public static Void main(String[] args){
            for(int i=0; i<10; i++){
                Test t= new Test();
                t=null;
            }
        }
        Public Void finalize(){
            Count ++;
            System.out.println("finalize() Called : "+ Count);
        }
}
```

In the above Program if we Keep on increasing i value at Certain Poin memory Problem will be raised and JVM runs automatically Garbage Collector.

Case 6: Memory Leaks :→

• The objects which are Not using in our Program and which are Not eligible for Garbage Collection, such type of useless objects are Called Memory Leaks.

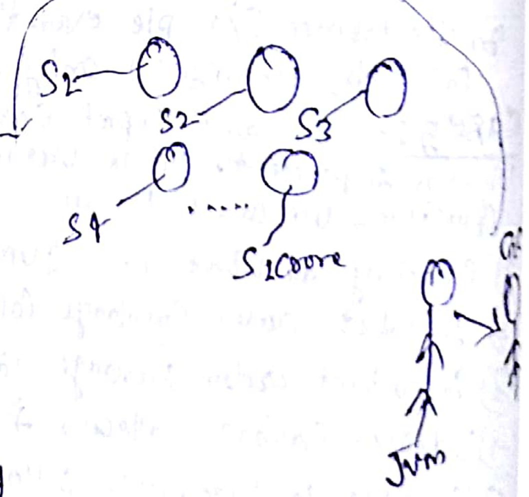
• In our Program if Memory Leaks Present then at certain point our Program fails by raising RE: OutOfMemoryError.

• To overcome this Problem if an object is no longer required, then it is Highly Recommended to make that object Eligible for GC.

• In our Program if Memory Leaks Present, then it is Purely Programmer's Mistake.

• The following are various memory Management Tools to Identify Memory Leaks in our Application.

- Hp-J-METER
- HP-OVO
- J-PROBE
- HP-PATROL
- IBM-TIVOLI

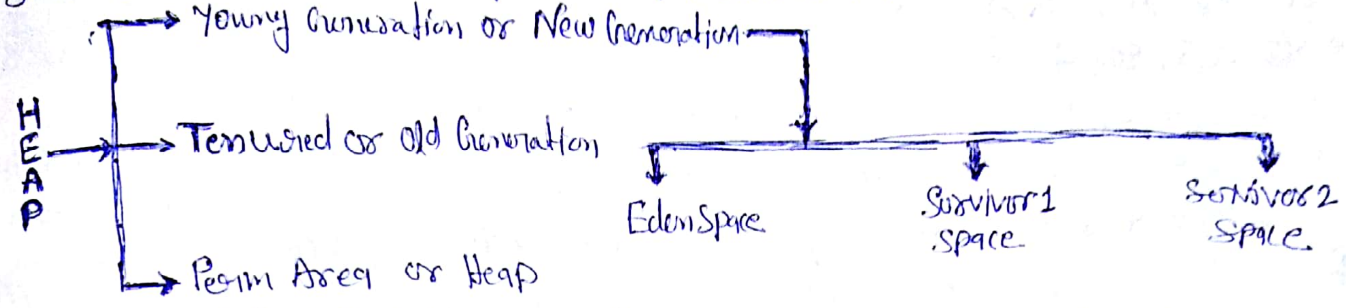


```
Student s1 = new Student();
Student s2 = new Student();
Student s3 = new Student();
        :
        :
Student s10000 = new Student();
```

RE: OutOfMemory error

* Heap Generations for Garbage Collections in Java *

• Java objects are Created in Heap and Heap is divided into three Parts or Generations



• when an object first Created in heap its gets Created in new generation inside Eden space and after subsequent minor Garbage Collection if object Survives its gets moved to Survivor 1 and then Survivor 2 before major garbage Collection moved that object to old or tenured generation.

• Permanent generation of Heap or Perm Area of Heap is Somewhat Special and it is used to store metadata related to Classes and method in JVM, It also hosts String Pool provided by JVM. There are many opinions around whether garbage Collection in Java happens in Perm Area of Java heap or not, as per my knowledge this is something which is JVM dependent and happens at least in Sun's implementation of JVM.

→• JVM Command line options -Xmx and -Xms is used to setup starting and max size for Java heap. ideal ration of this parameter is either 1:1 or 1:1.5 based on my Experience for example you can have either -Xmx and -Xms as 1GB or -Xms 1.2 GB and 1.8 GB.

• There is no manual way of doing garbage collection in Java.

Scanned by CamScanner