1. Introduction

- In old languages like C++ , programmer is Responsible for Both Creation and Destruction of Objects . Usually programmer taking very much care while creating Objects and neglecting Destruction of useless Objects . Due to his negligence at certain 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 this Assistant , the chance of failing Java Program with Memory problem 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 Daemon Thread as it is always running in the background .

1. Introduction
2. The ways to make an object Eligible for GC
   - Nullifying the reference variable .
   - Re-Assigning the reference variable .
   - Objects created inside a method .
   - Island of Isolation .
3. The ways for requesting JVM to run GC
   - By using System Class .
   - By using Runtime class .
4. Finalization .

The Ways to Make an Object Elidigible 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 Object Automatically Eligible for Garbage Collection .

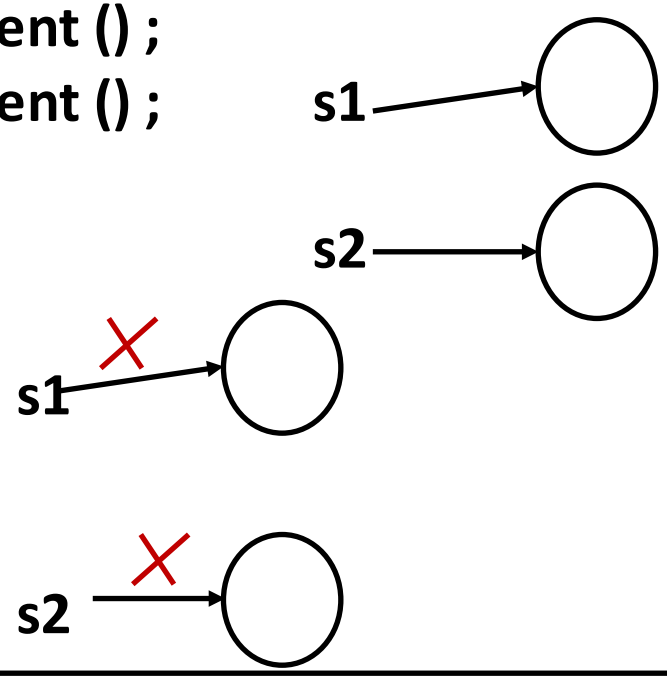
Example :

No object is eligible for GC

One object is eligible for GC

2nd object is eligible for GC

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



2. Re Assigning The reference Variable .

- If an object is no longer required , then Re Assign its Reference Variable to other object then Old Object is Automatically Eligible for GC .

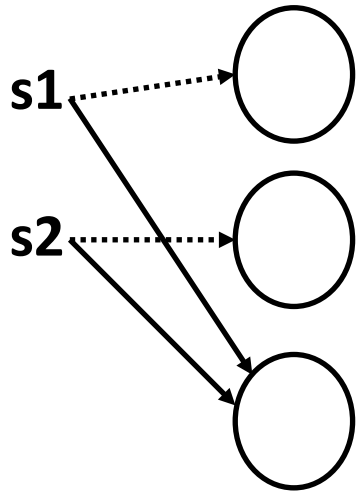
Example :

No object is eligible for GC

One object is eligible for GC

2nd object is eligible for GC

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



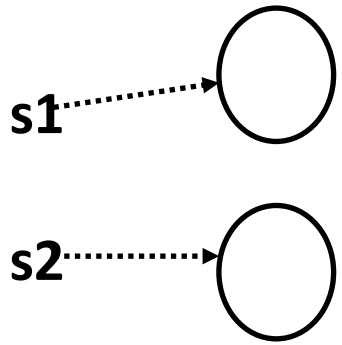
3. Objects created inside a method .

- Object created inside a method are Default eligible for Garbage Collection once Method Completes (Because the reference variable are local variable of that method and once method completes all local variables will be destroyed) .

Example :

After doStuff() two objects are eligible for GC

```
Class Test {
        public static void main(String[] a)
        {
                m1();
        }
        public static void doStuff()
        {
                Student s1 = new Student() ;
                Student s2 = new Student();
        }
}
```



Garbage Collection

After doStuff() only one objects are eligible for GC

```
Class Test {
        public static void main(String[] a)  {
                Student S=doStuff();
        }
        public static Student doStuf()  {
                Student s1 = new Student() ;
                Student s2 = new Student();
                return s1 ;
        }
}
```

After doStuff() two objects are eligible for GC

```
Class Test {
        public static void main(String[] a)  {
                doStuff();
        }
        public static Student doStuf()  {
                Student s1 = new Student() ;
                Student s2 = new Student();
                return s1 ;
        }
}
```

Garbage Collection

After doStuff() only one objects are eligible for GC

```
Class Test
 {
        static  Student s ;
        public static void main(String[] a)
        {
                doStuff();
        }
        public static Student doStuf()
         {
                Student s1 = new Student() ;
                s = new Student();
        }
}
```

4. Island of Isolation

No objects are eligible for GC

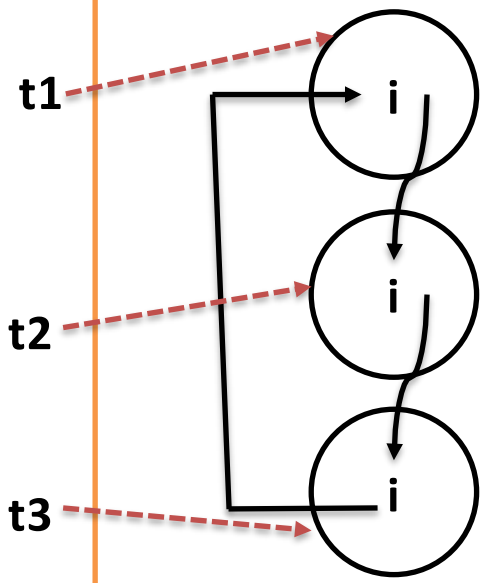
```
Class Test {
        Test  I ;
        public static void main(String[] a)
        {
                Test t1 = new Test () ;
                Test t2 = new Test () ;
                Test t3 = new Test () ;

                t1.i = t2 ;
                t2.i = t3 ;
                t3.i = t1 ;

                t1 = null ;
                t2 = null ;
                t3 = null ;
        }
}
```

All 3 objects are eligible for GC

Island Of Isololation



Note

- if an Object doesn't have any Reference variable then it is always eligible for GC .
- Even though Object having reference variable Still sometime it may be 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 eligible for GC , it May not Destroy Immediately by the Garbage Collector . Whenever JVM Runs Garbage Collector then only Objects will be Destroyed . But when exactly JVM runs GC , we can't expect . It depends on JVM and varied from 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 time 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 this purpose

   ```
   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()

```
Runtime r = Runtime.getRuntime();
```

Once we got 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 .

Note 1 : Singleton class

For any java class if we are allowed to create only one object , such type of class is called Singleton class .

Example

Runtime

BussinessDeligate

ServiceLocator

Note 2 : 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).

Example

```
Runtime r = Runtime.getRuntime();
DateFormat df = DateFormat.getInstance();
```

Garbage Collection

```
import java.util.Date;
class RuntimeDemo
{
        public static void main(String[] args)
        {
                Runtime r = Runtime.getRuntime();
                System.out.println(r.totalMemory());
                System.out.println(r.freeMemory());
                for(int i=0;i<10000;i++)
                {
                        Date d = new Date();
                        d=null;
                }
                System.out.println(r.freeMemory());
                r.gc();
                System.out.println(r.freeMemory());
        }
}
```

Note

- gc() method present in System class is static Method whereas gc() method present in Runtime class is Instance Method .

Q. Which of the following is valid way for representing JVM to run Garbage Collector ?

1. System.gc();
2. Runtime.gc();
3. New Runtime().gc();
4. Runtime.getRuntime().gc();

Ans

- 2nd is Wrong because gc() present in Runtime class is not static
- 3rd is wrong because we cannot create object by using new operator as Runtime class is a Singleton class

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 calls finalize() to perform cleanup activities . Once finalize() Completes Automatically GC Destroys that Object .
- finalize() present in Object class with the following prototype .

```
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()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 .

Garbage Collection

```
class Test
{
        public static void main(String[] args)
        {
                String s = new String("prolog");
                s=null;
                System.gc();
                System.out.println("End of Main");
        }
        public void finalize()
        {
                System.out.println("Finalize method called");
        }
}
```

- In the Above example String Object is eligible for GC and hence String Class finalize() got executed , which has Empty Implementation . Hence in this case Output is End Of Main .
- If we replace String Object with Test Object , then Test Object eligible for GC and hence Test class finalize() method will be executed . In this case output is

```
End of Main
Finalize method called
```

OR

```
Finalize method called
End of Main
```

Case 2

- Based on our Requirement we can call finalize() method Explicitly , then it will be executed just like a normal method call and object won't be destroyed . But before destroying an object Garbage Collector Always calls finalize() method .

Garbage Collection

```
class Test1
{
        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 method called");
        }
}
```

```
Finalize method called
Finalize method called
End of Main
Finalize method called
```

- 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 Always calls destroy() to perform cleanup activities . But based on our requirement we can call destroy() from init() and service() method explicitly , then it will be executed just like normal method call and Servlet object won't be Destroyed .

Case 3

- On any Object Garbage Collector calls finalize() only once , even though that Object is eligible for GC Multiple Times .

Garbage Collection

```
class Test2
{
        static Test2 s;
        public static void main(String[] args)throws InterruptedException
        {
                Test2 t= new Test2();
                System.out.println(t.hashCode());
                t=null;
                System.gc();
                Thread.sleep(5000);

                System.out.println(s.hashCode());
                s=null;
                System.gc();
                Thread.sleep(5000);
                System.out.println("End of Main");
        }
        public void finalize()
        {
                System.out.println("Finalize method called");
                s=this;
        }
}
```

Case 4

- **If the programmer calls finalize() explicitly and while executing that finalize() method , if any exception occurs and which is uncaught (that is there is no catch block) , the the program will be terminated Abnormally by raising that exception .**
- **If the Garbage collector calls finalize() explicitly and while executing that finalize() method , if any exception occurs and which is uncaught then JVM ignores that exception and rest of the program will be executed normally .**

Garbage Collection

```
class Test3
{
        public static void main(String[] args)throws Exception
        {
                Test3 t= new Test3();
                //t.finalize();   →1
                t=null;
                System.gc();
                System.out.println("End of Main");
        }
        public void finalize()
        {
                System.out.println("Finalize method called");
                System.out.println(10/0);
        }
}
```

- If we are not commenting Line 1 then programmer calls finalize() method and while executing that finalize() method ArithmeticException is raised which is uncaught . Hence the program will be terminated abnormally by raising ArithmeticException .
- If we comment Line 1 then garbage collector calls finalize() method and while executing that finalize() method if ArithmeticException is raised which is uncaught , JVM Ignores that Exception and rest of the program will be executed normally .

Q. Which of the following is true ?

1. JVM ignores every Exception which are raised while Executing finalize() method .
2. JVM ignores only Uncaught Exception which are raised while Executing finalize() method .

Ans:

1 is false

2 is true

Case 5

- We can't expect exact behavior of the garbage Collector . It is JVM vendor Dependent . It is varied from JVM to JVM . Hence we can't Answer exactly the following Question .

1. Exact 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 Objects 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 at what time .
- Most of the Garbage Collector follows Mark and Sweep Algorithm . But it doesn’t Means Every Garbage Collector follows the same Algorithm .

Garbage Collection

```
class Test4
{
        static int count =0;
        public static void main(String[] args)throws Exception
        {
                for(int i=0;i<10;i++)
                {
                        Test4 t=new Test4();
                        t=null;
                }
        }
        public void finalize()
        {
                count++;
                System.out.println("Finalize method called");
        }
}
```

- In the above program if we keep on increasing I value at certain point of memory problem will be raised and JVM runs automatically Garbage Collector.

1. Memory Leaks

- The objects which are not used 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 .

1. HP-J_METER
2. HP-OVO
3. J-PROBE
4. HP-PATROL
5. IBM-TIVOLI