15. Garbage Collection

1. Introduction
2. The ways to make an object eligible for GC
3. The methods for requesting JVM to run Garabage Collector
4. finalization.

1. Introduction:—

→ In old languages like C++, programmer responsible for both creation & destruction of objects.

→ Usually the programmer taking very much care while creating objects & neglecting destruction of useless objects.

→ Due to this neglectance at certain point for creation of new objects sufficient memory may not be available & entire application will be crashed due to memory problems.

→ Hence OutOfMemoryError is very common problem in old languages like C++.

→ But in Java programmer is responsible only for creation of objects & 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 becoz of this assistant the chance of failing Java program is very less becoz of memory problems. This assistant is nothing but Garbage Collector.

→ Hence the main objective of Garbage Collector is to destroy useless objects.

2. The ways to make an object eligible for GC:—

→ Eventhough programmer not responsible to destroy objects but its always a good programming practice to make an object eligible for GC if it is no longer required.

→ An object is said to be eligible for GC iff it doesn't contain any references.

→ The following are various ways to make an object eligible for GC.

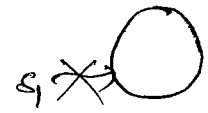
1) Nullifying the reference variable:—

→ If an object no longer required then assign null to all its reference variable then that object automatically will become eligible for GC.

Ex:

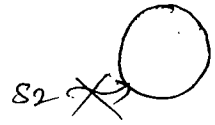
```
Student s1=new Student();
Student s2=new Student();
```

No object eligible for GC →




```
s1=null;
```

One object eligible for GC →




```
s2=null;
```

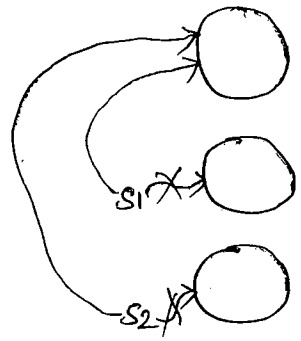
Two objects eligible for GC →

2) Reassigning reference variable:—

→ If an object no longer required then reassign its reference variable to some other object then old object is by default eligible for GC.

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

No object eligible for GC →




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

One object eligible for GC →

Two objects eligible for GC → s2 = s1;

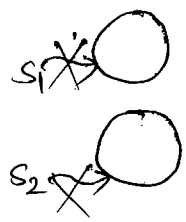
3) Objects created inside a method :—

→ The objects created inside a method are by default eligible for GC once method completes.

Ex: ①

```
class Test
{
  p s v m(—)
  {
    m1();
  }
  p s v m1()
  {
    Student s1=new Student();
    Student s2=new Student();
  }
}
```

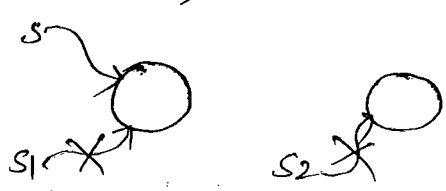
Two objects eligible for GC



Ex ②:

```
class Test
{
  p s v m(—)
  {
    Student s=m1();
  }
  p s Student m1()
  {
    Student s1=new Student();
    Student s2=new Student();
  } return s1;
}
```

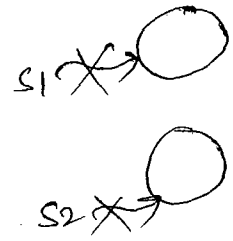
Two objects eligible for GC



Ex③:

```
class Test
{
  p s v m(-)
  {
    m1();
  }
  p s Student m1()
  {
    Student s1=new Student();
    Student s2=new Student();
    return s1;
  }
}
```

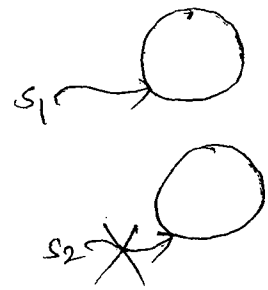
Two objects eligible for GC



Ex④:

```
class Test
{
  static Student s1;
  p s v m(-)
  {
    m1();
  }
  p s v m1()
  {
    s1=new Student();
    Student s2=new Student();
  }
}
```

One object eligible for GC



4) Island of Isolation:-

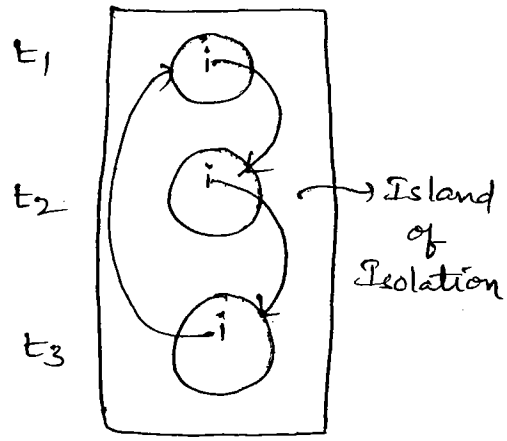
Ex:

```
class Test
{
  Test i;
  p s v m(-)
  {
    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;
}
}
```

No Object eligible for GC

3 objects eligible for GC



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

② Eventhough object having the reference still that object eligible for GC sometimes (if all references are internal reference Ex: Island of Isolation).

3) The methods for requesting JVM to run Garbage Collector:-

→ Once we made an object eligible for GC it may not be destroyed immediately by the Garbage Collector.

→ Whenever JVM runs GC then only that object will be destroyed, but when exactly JVM runs GC we can't expect it depends on JVM.

→ Instead of waiting until JVM runs GC, we can request JVM to run Garbage Collector but whether JVM accept our request or not there is no guarantee.

→ But most of the times JVM will accept our request.

→ The following are various ways for requesting JVM to run Garbage Collector.

1) By using System.gc();

→ 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 & it is a singleton class.

→ We can create Runtime object by using Runtime.getRuntime() method.

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

→ Once we get Runtime object we can apply the following methods on that object.

① freeMemory()

returns no. of bytes of free memory present in JVM.

② totalMemory() returns no. of bytes of total memory on the Heap (i.e., Heap size).

③ gc()

for requesting JVM to run Garbage Collector.

Ex:
```
import java.util.*;
class RuntimeDemo
{
  p s v m(-)
  {
    Runtime r = Runtime.getRuntime();
    S.o.p(r.totalMemory());
    S.o.p (r.freeMemory());
```

```
for (int i=0; i<10000; i++)
{
  Date d=new Date();
  d=null;
}
S.o.p(r.freeMemory());
r.gc();
S.o.p (r.freeMemory());
}
}
```

O/P: 5177344
4995928
4762144
5067808

Note:— ① gc() method present in System class is static method whereas gc() method present in Runtime class is instance method.

② It is convenient to use System.gc() becoz it is a static method, but it is recommended to use Runtime class gc() method becoz internally System class gc() method calls Runtime class gc() method.

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

Q: Which of the following is valid way for requesting JVM to run Garbage Collector?

✓ ① System.gc();
✗ ② Runtime.gc();
✗ ③ new Runtime().gc();
✓ ④ Runtime.getRuntime().gc();

4) finalization :—

→ Just before destroying an object Garbage Collector calls finalize() method to perform cleanup activities.

→ Once finalize() method completes automatically Garbage collector destroys that object.

→ finalize() method present in Object class with the following declaration.

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

Case(i): Just before destroying an object Garbage Collector always calls finalize() method to perform clean up activities on that object then the corresponding class finalize() method will be executed.

For example, if String object eligible for GC then String class finalize() method will be executed but not Test class finalize() method.

```
class Test
{
  p s v m(—)
  {
    String s=new String("durga");
    s=null;
    System.gc();
    S.o.p ("End of main");
  }
  public void finalize()
  {
    S.o.p("finalize method called");
  }
}
```

GC

durga.finalize()
{
}

→ In the above program, String object eligible for GC and hence String class finalize() method got executed, which has empty implementation. In this case the o/p is

end of main

→ If we replace String object with Test object then Test class finalize() method will be executed. In this case o/p is

End of main	finalize method called
finalize method called	End of main

Case (ii): We can call finalize() method explicitly then object won't be destroyed & it will be executed just like a normal method call.

But before destroying an object Garbage Collector always calls finalize() method.

Ex:
```
class Test
{
  p s v m(-)
  {
    Test t=new Test();
    t.finalize();
    t.finalize();
    t=null;
    System.gc();
    S.o.p("End of main");
  }
  public void finalize()
  {
    S.o.p("finalize method called");
  }
}
```

→ In the above example finalize() method will be executed 3 times, in that 2 times by the programmer like a normal method

call & one time by the Garbage Collector.. In this case the O/P is

```
finalize method called
finalize method called
end of main
finalize method called
```

***

Note:- Just before destroying Servlet object web container always calls destroy() method, but based on our requirement we can call destroy() method explicitly from init(-) & service(-,-) methods then it will be executed just like a normal method call.

Case(iii): If programmer calls finalize() method & while executing that finalize() method if an Exception raised & uncaught then the program will be terminated abnormally by raising that Exception

If Garbage Collector calls finalize() method & while executing that finalize() method if an exception raised & uncaught then JVM ignores that Exception & rest of the program will be executed normally.

Ex:
```
class Test
{
  p s v m(-)
  {
    Test t=new Test();
    t.finalize();  ---> (1)
    t=null;
    System.gc();
    S.o.p("End of main");
  }
  public void finalize()
  {
    S.o.p("finalize method called");
  } } S.o.p (10/0);
```

→ If we are not commenting line ① then programmer calls finalize() method & while executing that finalize() method ArithmeticException raised which is uncaught.

→ Hence the program will be terminated abnormally by raising that Exception.

→ If we are commenting line ① then Garbage Collector calls finalize() method & while executing that finalize() method AE raised which is uncaught.

→ Hence JVM will ignore that Exception & rest of the program will be executed normally. In this case o/p is.

```
end of main
finalize method called
```

Q: Which of the following is true?

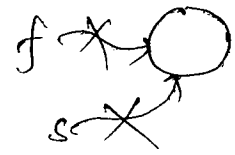
① JVM ignores every Exception which is raised while executing finalize() method.

② JVM ignores only uncaught Exceptions which are raised while executing finalize() method.

Case (iv): On any object Garbage Collector calls finalize() method only once eventhough object eligible for GC multiple times.

Ex:

```
class FinalizeDemo
{
  static FinalizeDemo s;
  P S V m(_) throws Exception
  {
    FinalizeDemo f=new FinalizeDemo();
    S.o.p(f.hashCode());
    f=null;
    System.gc();
    Thread.sleep(5000);
```



```
    S.o.p(s.hashCode());
    s=null;
    System.gc();
    Thread.sleep(10000);
    S.o.p("End of main");
  }
  public void finalize()
  {
    S.o.p("finalize method called");
    s=this;
  }
}
```

O/P: 4072869
finalize method called
4072869
End of main.

→ In the above program, eventhough object eligible for GC multiple times but Garbage Collector calls finalize() method only once.

Case(v): We can't expect exact behaviour of Garbage Collector which is varied from JVM to JVM.

Hence for the following questions we can't answer exactly.

① when exactly JVM runs Garbage Collector?
② In which order JVM identifies eligible objects?
③ In which order GC destroys objects?
④ Whether Garbage Collector destroys all eligible objects or not.
⑤ What is the algorithm followed by Garbage Collector? etc.

Note:- ① Whenever program runs with low memory then JVM runs Garbage Collector, but exactly at what time we can't expect.

② Most of the Garbage Collectors follow Mark & Sweep algorithm, it doesn't mean every Garbage Collector follows same algorithm.

Ex:
```
class Test
{
    static int count=0;
  p s v m(-)
  {
```

```
for (int i=0; i<100000; i++)
{
    Test t=new Test();
    t=null;
}
public void finalize()
{
  S.o.p("finalize method called :"++count);
}
}
}
```

Case (vi): Memory Leaks:-

→ The objects which are not using in our program which are not eligible for GC such type of useless objects are called Memory Leaks.

→ In our program, if memory leaks present then at certain point the program will be terminated by raising OutOfMemoryError.

→ Hence if an object is no longer required then it is highly recommended to make that object eligible for GC.

→ The following are various memory management tools to identify Memory Leaks.

HP OVO
HP J Meter
JProbe
Patrol
IBM Tivoli