**Garbage Collection**

**Introduction**  
In ‘C++’ the programmer is responsible for both creation and destruction of objects but usually the programmer is giving very much importance for creation of objects and he is ignoringthe destruction of objects. Due to this at creation point of time there may not be sufficient memory for the creation of new objects and entire program may fails. But in java programmer is responsible only for creation of objects but sun people has introduced one assistance which is running continuously in the background for destruction of objects. Due to this assistance there is no chance of failing the java program due to memory problem, this assistance is nothing but “Garbage Collector”.

**The ways to make an object eligible for Garbage Collector**  
Even though the programmer is not responsible for destruction of objects it’s good programming practice to make an object eligible for Garbage Collector if it is no longer required.  
The following are different ways for this  
**Nullifying the reference Variable**  
If an object is no longer required assign null to all its reference variables.

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

No Object eligible for Garbage Collector

s1 = null;

One Object eligible for Garbage Collector

s2 = null;

Both Objects eligible for Garbage Collector

**Reassigning the reference Variable**  
Ex:  
Student s1 = new Student();  
Student s2 = new Student();

No Object eligible for Garbage Collector  
s1 = s2;  
One Object eligible for Garbage Collector

**The Objects Created inside a method**  
The objects which are created in a method are by default eligible for Garbage Collector once the method completes

Ex:  
1) class Test  
{  
public static void main(String arg[])  
{  
m1();

Both Objects eligible for Garbage Collector  
}  
public static void m1()  
{  
Student s1 = new Student();  
Student s1 = new Student();  
}  
}  
2) class Test  
{  
public static void main(String arg[])  
{  
Student s = m1();

One Object eligible for Garbage Collector  
}  
public static Student m1()  
{  
Student s1 = new Student();  
Student s1 = new Student();  
return s1;  
}  
}  
3) class Test  
{  
public static void main(String arg[])  
{  
int ref=m1();

Both Objects eligible for Garbage Collector  
}  
public static Student m1()  
{  
Student s1 = new Student();  
Student s1 = new Student();  
return s1;  
}  
}

**The Island of Isolation**  
Ex:  
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;

No Object eligible for Garbage Collector  
t1 = null;

No Object eligible for Garbage Collector  
t2 = null;

No Object eligible for Garbage Collector  
t3 = null;

All Object eligible for Garbage Collector  
}  
}

**Note:** 1) If an object doesn’t have any reference variable that object is always eligible for Garbage Collection  
3) Even though object having the reference variable still there may be a chance of that object eligible for Garbage Collection (Island of Isolation …Here ‘i’ is internal reference)

**The methods to request JVM to run Garbage Collector**  
We can request JVM to run Garbage Collector but there is no guarantee whether JVM accepts our request or not. We can do this by using the following ways.  
**By System class**  
‘System’ class contains a static ‘gc’ method for requesting JVM to run Garbage Collector.  
System.gc();  
**By Using Runtime Class**  
A java application can communicate with JVM by using Runtime Object. We can get Runtime Object as follows.  
Runtime r = Runtime.getRuntime();  
Once we get Runtime Object we can apply the following methods on that object.  
freeMemory(): returns the free memory available in the loop

totalMemory(): returns heap size  
gc(): for requesting JVM to run GarbageCollector

Ex:

import java.util.\*;  
class RuntimeDemo  
{  
public static void main(String arg[])  
{  
Runtime r = Runtime.getRuntime();  
System.out.println(r.totalMemory());

**Q) Which of the following is the valid way for requesting JVM to run gc?**  
System.gc();  
Runtime.gc(); –>>not a static  
(new Runtime()).ge();—>>we can’t create Object  
Runtime.getRuntime().gc();

**finalization**  
Just before destroying any object Garbage Collector always calls finalize() to perform clean up activities.  
finalize() is available in the Object class which is declared as follows.  
protected void finalize() throws Throwable  
{  
}  
**case1:** Garbage Collector always calls finalize() on the Object which is eligible for Garbage Collector and the corresponding class finalize method will be executed.  
**Ex:**  
class Test  
{  
public static void main(String arg[])  
{  
String s = new String(“raju”);  
//Test s = new Test();  
s = null;  
System.gc();  
System.out.println(“end of main method”);  
}  
public void finalize()  
{  
System.out.println(“finalize method called”);  
}  
}  
**O/P:- end of main method.**  
In this case String Object is eligible for G.C and hence String class finalize() method has been executed.  
In the above program if we are replacing String Object with Test Object then Test class finalize() will be executed.  
**In this case O/P is**

end of main method

finalize method called  
or finalize method called end of main method

**case2:** we can call finalize() explicitly in that case it will execute just like a normal method and object won’tbe destroyed.  
While executing finalize() method if any exception is uncaught it is simply ignored by the JVM but if we are  
calling finalize method explicitly and if an exceptions is uncaught then the program will be terminated abnormally.  
Ex:  
class Test  
{  
public static void main(String arg[])  
{  
Test s = new Test();  
//s.finalize();  
s = null;  
System.gc();  
System.out.println(“End of main method”);  
}  
public void finalize()  
{  
System.out.println(“finalize method”);  
System.out.println(10/0);  
}  
}  
O/P:- finalize method

end of main method

Q) which of the following statements are true  
1) JVM ignores all exceptions which are raised while executing finalize()  
2) JVM ignores only uncaught exceptions which are raised during execution of  
finalize()

**case3:-** 1) Garbage Collector calls finalize() only once on any object i.e it won’t call more than once.  
2) While executing finalize() there maybe a chance of object getting reference variable at that time  
G.C won’t destroy that object after completing finalize()  
3) If the same object is eligible for G.C second time , with out executing finalize() method G.C will destroy that object.  
Ex:  
class FinalizeDemo  
{  
//Test s;  
public static void main(String arg[])  
{  
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(5000);  
System.out.println(“End of main method”);  
}  
public void finalize()  
{  
System.out.println(“finalize method called”);  
s = this;  
}  
}  
The behavior of G.C is vendor dependent hence there is no guarantee for the following  
1) Whether the G.C follows mark & sweep algorithm or not  
2) What exact Algorithm followed by Garbage Collector  
3) In which order Garbage Collector destroys Object  
4) Whether Garbage destroys all eligible objects or not  
5) At what time exactly Garbage Collector will run.

The finalize method is never invoked more than once by a Java virtual machine for any given object.

The Java programming language does not guarantee which thread will invoke the finalize method for any given object.

It is guaranteed, however, that the thread that invokes finalize will not be holding any user-visible synchronization locks when finalize is invoked.

If an uncaught exception is thrown by the finalize method, the exception is ignored and finalization of that object terminates.