Garbage Collection

- **Introduction**
- **Various ways to make an object eligible for G.C.**
- **The methods for requesting JVM to run garbage collector.**
- **Finalization.**

Garbage Collector

- In old languages like C++, creation and destruction of object is responsibility of programmer only.
- Usually programmer takes very much care while creating objects neglecting destruction of useless objects. Due to this negligence of certain point of time for creation of new object sufficient memory may not be available and entire program will collapse due to memory problems.
- But in java, programmer is responsible only for creation of objects and not for destruction of useless objects.
- Sun people provided one assistant which is always running in the background for destruction of useless objects, due to this assistant chance of failure java program with memory problem is very rare. This assistant is nothing but **'Garbage collector'**.
- Hence, the main objective of garbage collection is to **"Destroy useless Objects".**

The Various ways to make an object eligible for G.C.

- Even though programmer is not responsible to destroy useless objects, it is always good programming practice to make an object eligible for G.C. if it is no longer required.
- An object is said to be eligible for G.C. if it doesn’t contain any references.
- The following are various possible ways to make an object eligible for G.C.

  I) nullify the reference variable:

  - If an object is no longer required then assign null to all its references, then automatically that object eligible for G.C.

    Example: Student s1=new Student ();

    ```
    Student s1=new Student ();
    -            <-------------------  No objects are eligible for G.C.
    -
    -
    s1=null;
    -            <-------------------  One objects are eligible for G.C
    -
    -
    s2=null;     <-------------------  Two object are eligible for G.C.
    ```

  II) Reassigning the reference variable:

  - If an object is no longer required then reassign its eference variable to some other objects then that old object automatically eligible for G.C.

    Example:

    ```
      Student s1=new Student();
     Student s2=new Student();
    -            <-------------------  No objects are eligible for G.C.
    -
    -
    s1=null;
    -            <-------------------  One objects are eligible for G.C
    -
    ```

-
s2=s1; $\longleftarrow$ Two object are eligible for G.C.

III) Reassigning the reference variable:

- The objects which are created inside a method one by default eligible for G.C. after completing that method.
- Example:

```
   class test
{
public static void main(String args[ ])
{
m1()         <------------------    Two objects are eligible for G.C.
}
public static void m1()
{
Student s1=new Student();
Student s2=new Student();

}
```

Example 2:

```
public class GarbageCollection {
   public static void main(String[] args) {
      Student s=m1();
   }
   public  static Student m1()
   {
      Student s1=new Student();
      Student s2=new Student();
      return s1;
   }
```

Example 3:

```
public class GarbageCollection1 {
    public static void main(String[ ] args) {
        m1(); ←————————————— Two objects are eligible for G.C.

    }
    public  static Student m1()
    {
        Student s1=new Student();
        Student s2=new Student();
        return s1;
    }}
```

Iv)Island of Isolation:

```
public class GarbageCollection1 {
Test i ;
    public static void main(String[] args) {
      Test t1=new Test();                Two objects are eligible for G.C.
      Test t2=new Test();
      Test t3=new Test();
      -    -
t1.i=t2;
t2.i=t3;
t3.i=t1;
-
-
t1=null;
-
t2=null;
-
t3=null; ←—————————— 3 objects eligible for G.C.-
}
}
```

Note:

- If an object have any reference then it is always eligible for Garbage Collector..
- Even though object having the reference still it is eligible for G.C.sometimes(island of isolation).

The methods for requesting JVM to run Garbage Collector:

- Whenever we are making an object eligible for G.C. it may not be destroyed by G.C. immediately whenever JVM runs garbage collector then only that object will be destroyed.
- We can request JVM to run garbage collector, programmatically weather JVM accept over request are not there is no guarantee.
- The following are various ways for requesting JVM to run.

  1)By System class:

    - System class contains a static method G.C. for this

  System.gc();

  2)By Runtime class.

    - By using runtime object a java application can communicate with JVM.
    - Runtime class is a singleton class hence we can't create Runtime object by using constructor.
    - We can create runtime object by using factory method getruntime()
      Runtime r=Runtime.getRuntime();
    - Once we got Runtime object we can apply the following methods on that object.
    - A)freeMemory() returns freeMemory in the Heap.
      b)totalMemory() returms totalmemory of the Heap.
      c)gc() For requesting JVM to Run garbage collector.

Example:

```
  public class Runtime1 {
    public static void main(String[] args) {
     Runtime r=Runtime.getRuntime()
        System.out.println(r.totalMemory());
         System.out.println(r.freeMemory());
         for(int i=1;i<=10000;i++)
         {
             Date d=new Date();d=null;
         }
         System.out.println(r.freeMemory());
         r.gc();
         System.out.println(r.freeMemory());
    }
}
```

Qu)Which of the following is the proper way to requested JVM to run g.c.

```
1)System.gc();                //Valid
2)Runtime.gc();               //Invalid
3)(new Runtime()).gc();     //Invalid
4)Runtime.getRuntime().gc();
```

Note:gc() present in the System class is static mrthod.Where as gc() present in the Runtime class is instance method and recommended to use System.gc();

Finalization

- Just before destroying any object. Garbage collector always calls finalize() method to perform clean-up activities on that object.
- finalize () method declare in object class with following declaration.

  Protected void finalize() throws Throwable

  Case 1):

  Garbage collector always calls finalize() on the object which is eligible for G.C. just before destruction, then the corresponding class finalize() will be executed. If String object eligible for G.C. then string class finalize() will be executed, but not test class finalize method.

Example:

```
   Class Test
{
public static void main(String ar[])
{
String s=new String("CST");
s=null;
System.gc();
System.out.println("End of main");
}
public void finalize()
{
System.out.println("Finalize called");
}
}
```

Output:End of main

- In the above example String object is eligible for G.C. Hence String class finalize() method got executed which has empty implementation.

- If we are replacing String object with test object. Then test class finalize() will be executed.
- In this case the output is
  1)Finalize called
  End of main.
  2)End of main
  Finalize called.
  Case 2)
  - We can call finalize () explicitly in this case it will be executed just like a normal method call and object won't destroyed.
  - But before destruction of an object G.C. always call finalize().

Example:

```
class Test
{
public static void main(String ar[]);
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");

}

}
```

Output:

```
Finalize method called

Finalize method called

End of main

Finalize method called

-

-

-
```

- In the above program finalize() got executed 3 times,2 times explicitly by the programmer and one time by the garbage collector.
  Note:

- Before destruction of servlet object web container always calls destroy method, to perform clean-up activities.
- It is possible to called destroy explicitly from init() and service() in this case it will be executed, just like normal method call and servlet object won't be destroyed.

  Case 3)
  - If we were calling finalize() explicitly and while executing that finalize() if any exception raised and uncaught, then the program will be terminated abnormally.
  - If G.C. calls finalize() and while Executing that finalize(),if any exception is uncaught then JVM simply ignores that uncaught exception and rest of the program will be executed normally.

    Example:

```
 class test
{
public static void main(String ar[])
{
```

```
Test t=new test();
t.finalize(); ←————————      Line 1
t=null;
System.gc();
System.out.println("End of main");
}
publc void finalize()
{
System.out.println("Finalize mehod called");
System.out.println(10/0);
}
}
```

- If we are on comment line (1),were calling the finalize() explicitly and the program will be terminated abnormally.
- If we are commenting line(1),then G.C. calls finalize() and the raised Arithmetic Exception is ignored by JVM Hence in this case the Output is
- Output: End of main
  Finalize method called
  Qu)Which of the following statement is true?
  1)While executing finalize() all exceptions are ignored by JVM.
  Ans:False
  2) While executing finalize() all uncaught exceptions ignored by JVM.
  Conclusion:
  - On any object G.C. calls finalize() only once.

  Example:

```
class FinalizeDemo
{
public static void main(String ar[])
{
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;

}

}
```

Output:

```
4073475
Finalize method called
4073475
End of main method
```

- The behavior of the G.C. is vendor dependent and hence we can't expect exactly because of this we can't answer the following questions exactly:
  1)When JVM runs G.C. exactly.
  2)What is the algorithm following by G.C.
  3)In which order G.C. destroys the objects.
  4)Wheather G.C. destroys all eligible objects or not e.t.c.
  Note:
  We can't tell exact algorithm followed by G.C.,but most of the cases it is mark and sweep Algorithm.

Memory Leak:

- If an object having the reference then it is not eligible for G.C. even though we are not using that object in our program still it is not destroyed by the G.C. such type of object is called "Memory Leak".(i.e. **Memory Leak is useless object which is not eligible for G.C.).**
- We can resolve memory leak by making useless objects for G.C. explicitly and by invoking G.C. programmatically.