Garbage Collections

Introduction

- In Old Languages Like C++, Programmer is Responsible for Both *Creation* and *Destruction* of Useless Objects.
- Usually Programmer taking to Very Much Care while creating Objects and neglecting Destruction of Useless Objects.
- Due to this 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 Problems is Very Less (Robust).
- This Assistant is Nothing but Garbage Collector.
- Hence the Main Objective of Garbage Collector is to Destroy Useless Objects.

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 References.

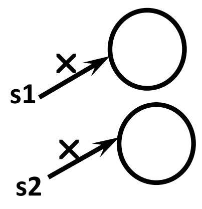
The following are Various Possible Ways to Make an Object Eligible for GC.

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.

Eg1:

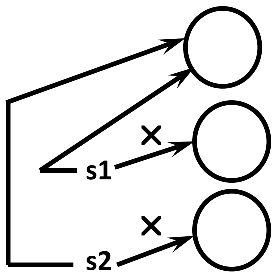
```
Student s1 = new Student();
Student s2 = new Student();
            :           ———> No Objects are Eligible for GC
            :
            :
           s1 = null; //One Object is Eligible for GC
            :
            :
           s2 = null; //2nd Object is Also Eligible for GC
```



2) Re Assigning the Reference Variables:

If an Object is No Longer to required then Re Assign its Reference Variable to any New Objects, then Old Object is Automatically Eligible for GC.

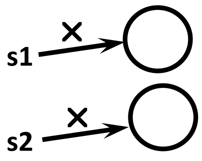
```
Student s1 = new Student();
Student s2 = new Student();
        :          ——> No Objects are Eligible for GC
        :
        :
      s1 = new Student(); //One Object is Eligible for GC
        :
        :
      s2 = s1; //2 Object are Eligible for GC
```



3) Obects Created Inside a Method:

Objects Created Inside a Method are by Default Eligible for GC Once Method Completes (Because the Reference Variables are Local Variables of that Method).

```
class Test {
    public static void main(String[] args) {
                m1(); //After m1() 2 Objects are Eligible for GC
    }
    public static void m1() {
        Student s1 = new Student();
        Student s2 = new Student();
    }
}
```



```
class Test {
    public static void main(String[] args) {
        Student s = m1(); //After m1() Only 1 Object is Eligible for GC
    }
    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
    }
}
```

```
class Test {
        public static void main(String[] args) {
                m1(); //After m1() 2 Objects are Eligible for GC
        }
        public static Student m1() {
                    Student s1 = new Student();
                    Student s2 = new Student();
                    return s1;
        }
}
```

```
class Test {
        static Student s;
        public static void main(String[] args) {
                m1(); //After m1() Only 1 Object is Eligible for GC
        }
        public static voide m1() {
                    Student s1 = new Student();
                    s = new Student();
        }
}
```

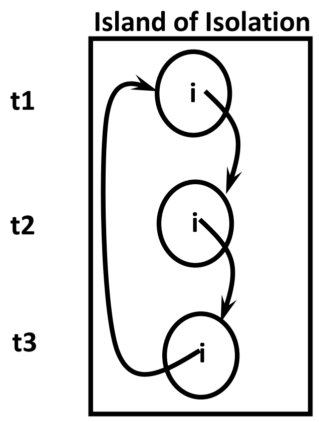
4) Island of Isolation:

```
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;
        }
}
```

No Object is Eligible for GC



Before t3 = null No Objects are Eligible for GC

After t3 = null All 3 Objects are Eligible for GC

Note:

1) If an Object doesn't have any Reference then it is Always Eligible for GC.
2) Even though Object having Reference Still there May be a Chance of Object Eligible for GC (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 if May Not Destroy Immediately by the Garbage Collector.
- Whenever JVM Runs GC then Only Object will be Destroyed. But Exactly at what Time JVM Run GC, we can't Expect. It Depends on JVM Vendor.
- 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 Accept 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()*.
     Eg: 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.

```
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() Present in System Class is Static Method whereas gc() Present in Runtime Class is Instance Method.

Which of the following is Valid Way for requesting JVM to Run Garbage Collector?

1) System.gc(); √
2) Runtime.gc(); ✕ Runtime is Static
3) new Runtime().gc(); ✕ Runtime is Singleton
4) Runtime.getRuntime().gc(); √

Note:
- With Respect to Convenience it is Recommended to Use *System.gc()*.
- With Respect to Performance wise it is Recommended to Use *r.gc()*. Because Internally System.gc() Calls r.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() in Our Class to Perform Cleanup Activities.

Case 1:

- Just before Destroying an Object Garbage Collector Always Calls finailze() 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().

```
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");
        }
}
```

<u>Output:</u> End of Main

- In the Above Example String Object 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 Class finalize() 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() 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().

```
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");
        }
}
```

```
finalize() Called
finalize() Called
finalize() Called
End of main()
```

- In the Above Example finalize() got executed 3 Times.
- 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()* 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, then the Program will be terminated Abnormally.
If Garbage Colector Calls finalize() and by executing that finalize(), if any Exception raised Uncaught then JVM Run 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);
        }
}
```

```
finalize() Called
End of main()
```

If we are Not Commenting Line 1 then Programmer Calls finalize() and while executing that finalize() ArthimeticException raised which is Uncaught. Hence the Program will be terminataed Ubnormally by raising ArthimeticException. In this the Output is

```
finalize() Called
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

If we Comment Line 1 then Garbage Collector Calls finalize() and while executing that finalize() ArthemeticException raised which is Uncaught. JVM Ignores that Exception and Rest of the Program will be executed Normally.

Which of the following is true?

- JVM Ignores Every Exception which raised while Execution fnialize().
- JVM Ignores Only Uncaught Exception which are raise by executing finalize() //true

Case 4:

On any Object Garbage Collector Calls fianlize() 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;
        }

}
```

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

In the Above Example even though Object Eligible for GC Multiple Times. But GC Calls finalize() Only Once.

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 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 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 Exactly at what Time.
- Most of the Garbage Collectors follow *Mark* and *Sweep* Algorithm. But it doesn't Means Every Garbage Collector follows 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() {
                        System.out.println("finalize() Called: "+count);
            }
}
```

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 we will get Runtime Exception Saying OutOfMemoryError.
- To Overcome this Problem if an Object 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 Programmers Mistake.

- The following are Various Memory Management Tolls to Identify Memory Leaks in Application
  - HP-J-METER
  - HP-OVO
  - J-PROBE
  - HP-PATROL
  - IBM-TIVOLI

Practice Questions on Garbage Collection

Q1. Given:

```
public class MarkList
{
   int num;
   public static void graceMarks(MarkList obj4)
   {
      obj4.num+=10;
   }
   public static void main(String[] args)
   {
      MarkList obj1= new MarkList();
      MarkList obj2=obj1;
      MarkList obj3=null;
      obj2.num=60;
      graceMarks(obj2);
   }
}
```

How many MarkList instances are created in memory at runtime?

A. 1
B. 2
C. 3
D. 4

Answer: A

Q2. Given the code fragment:

```
class Student
{
   String name;
   int age;
}
And,
public class Test
{
   public static void main(String[] args)
   {
      Student s1= new Student();
      Student s2= new Student();
      Student s3= new Student();
      s1=s3;
      s3=s2;
```

```
        s2=null;-->line-1
    }
}
```

Which statement is true?

A. After line-1, three objects eligible for garbage collection
B. After line-1, two objects eligible for garbage collection
C. After line-1, one object eligible for garbage collection
D. After line-1, no object eligible for garbage collection

Answer: C