Telegram link :: https://web.telegram.org/k/#-4132683294

Object class methods
++++++++++++++++++++

```
public class Object {

  public Object();

  public final native Class<?> getClass();

  //Commonly used methods on every object
  public java.lang.String toString();
  public native int hashCode();
  public boolean equals(Object object);


  //cloning :: shallowcopy, deep copy
  protected native Object clone() throws CloneNotSupportedException;


  //multithreading environment
  public final native void notify();
  public final native void notifyAll();
  public final void wait() throws InterruptedException;
  public final native void wait(long) throws InterruptedException;
  public final void wait(long, int) throws InterruptedException;


  //Garbage Collector
  protected void finalize() throws java.lang.Throwable;
}
```

Garbage Collector
==========

1. Introduction:
2. The way to make an object eligible for GC
    i. Nullifying the reference variable
    ii. Reassign the reference variable
    iii. Objects created inside a method
    iv. Island of Isolation
3. The methods for requesting JVM to run GC
    i. By System class
    ii. By Runtime class
4. Finalization

=> Case 1 : Just before destroying any object GC calls finalize() method on the object.
=> Case 2 : We can call finalize() method explicitly.
=> Case 3 : finalize() method can be call either by the programmer or by the GC.
=> Case 4 : On any object GC calls finalize() method only once Memory leak.

Introduction:

=> In old languages like C++ programmer is responsible for both creation and destruction of objects.

Usually programmer is taking very much care while creating object and neglect destruction of useless objects.

Due to his negligence at certain point of time for creation of new object sufficient memory may not be available

and entire application may be crashed due to memory problems.

=> But in java programmer is responsible only for creation of new object and his not responsible for destruction of objects.

=> Sun people provided one assistant which is always running in the background for destruction at useless objects.
Due to this assistant the chance of failing java program is very rare because of memory problems.

=> This assistant is nothing but garbage collector. Hence the main objective of GC is to destroy useless objects.

The ways to make an object eligible for GC:
=> Even through programmer is not responsible for destruction of objects but it is always a good programming practice
to make an object eligible for GC if it is no longer required.
=> An object is eligible for GC if and only if it does not have any references.

eg#1

```
import java.util.Date;

public class Test
{
   public static void main(String[] args)throws Exception
      {
            /*
                  public native long freeMemory();
                  public native long totalMemory();
                  public native long maxMemory();
            */
            Runtime r = Runtime.getRuntime();
            System.out.println("Total memory on heap is :: "+r.totalMemory());
            System.out.println("Free  memory on heap is :: "+r.freeMemory());

            for (int i =0;i<10000 ; i++)
            {
                  Date d = new Date();
                  d = null;

            }

            System.out.println("Free memory on heap is :: "+r.freeMemory());
            r.gc();
            System.out.println("Free  memory on heap is :: "+r.freeMemory());

      }
}
```

Output

```
Total memory on heap is :: 126877696
Free  memory on heap is :: 125535480
Free memory on heap is  :: 124859112
Free  memory on heap is :: 125911304
```

Note : Runtime class is a singleton class so not create the object to use constructor.

Which of the following are valid ways for requesting jvm to run GC ?
System.gc(); (valid)
Runtime.gc(); (invalid)
(new Runtime).gc(); (invalid)
Runtime.getRuntime().gc(); (valid)

Note: gc() method present in System class is static, where as it is instance method in Runtime class.

Note: Over Runtime class gc() method , System class gc() method is recommended to use because of the following reason.

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

This is because internally System class also uses Runtime class gc.

Note: In java it is not possible to find size of an object and address of an object.

```
Student std = new Student();
```

Q> what std holds?
ans. Address(wrong)
ans. object reference(internally hashCode value)

Finalization:
=> Just before destroying any object gc always calls finalize() method to perform cleanup activities.

=> If the corresponding class contains finalize() method then it will be executed otherwise Object class finalize()
method will be executed,which is declared as follows.

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

Case 1:
Just before destroying any object GC calls finalize() method on the object which is
eligible for GC then the corresponding class finalize() method will be executed.
For Example if String object is eligible for GC then String class finalize()method is
executed but not Test class finalize()method.

eg#1.

```
public class TestApp{
      public static void main(String... args){
            String s=new String("sachin");
            s=null;
            Runtime.getRuntime().gc();
            System.out.println("end of main()");
      }
      public void finalize() {
            System.out.println("finalized method called..");
      }
}
```

In the above program String class finalize()method got executed. Which has empty implementation.
If we replace String object with Test object then Test class finalize() method will

be executed.

eg#2.

```
public class TestApp{
      public static void main(String... args){
            String s=new String("sachin");
            s=null;
            TestApp t =new TestApp();
            t=null;
            Runtime.getRuntime().gc();
            System.out.println("end of main()");
      }
      public void finalize() {
            System.out.println("finalized method called..");
      }
}
```

case2
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 any object GC always calls finalize() method.

eg#1.

```
public class TestApp{
      public static void main(String... args){
            TestApp t =new TestApp();
            t.finalize();//Explicitly we are calling so object wont be cleaned
            t.finalize();

            t=null;
            System.gc();
            System.out.println("End of main()");
      }
      public void finalize() {
            System.out.println("finalized method called..");
      }
}
```

Output:

```
finalize() method called.
finalize() method called.
finalize() method called.
End of main.
```

In the above program finalize() method got executed 3 times in that 2 times explicitly by
the programmer and one time by the gc.

Note: In servlet coding

1. init()   => At the time of Servlet Initialisation.
2. service() => For every request it will be called.
3. destroy() => At the time of servlet destroy object by container.

Life cycle methods of Servlet

```
============================
init(){
    destroy();//just like normal methods call will be made no servlet deinstantion.
}
service(){
    destroy();
```

```
}
destroy(){
      //logic of deinstantiation
}
```

Case 3:
finalize() method can be call either by the programmer or by the GC .
 => If the programmer calls explicitly finalize() method and while executing the finalize()
    method if an exception raised and uncaught then the program will be terminated abnormally.

=> If GC calls finalize() method and while executing the finalize()method if an exception
    raised and uncaught then JVM simply ignores that exception and the program will be
    terminated normally

eg#1.

```
public class TestApp{
      public static void main(String... args){
            TestApp t =new TestApp();

            t.finalize();

            t=null;
            System.gc();
            System.out.println("End of main()");

      }
      public void finalize() {
            System.out.println("finalized method called..");
            System.out.println(10/0);
      }
}
```

=> If we are not comment line1 then programmer calling finalize() method explicitly and
   while executing the finalize()method ArithmeticException raised which is uncaught hence the program terminated abnormally.
=> If we are comment line1 then GC calls finalize() method and JVM ignores ArithmeticException and program will be terminated normally.