Garbage Collector
+++++++++++++++++
Different ways to make Garbage Object eligible for GC
 a. Nullifying the Object
 b. ReUsing the same reference
 c. Objects created inside the method
 d. Island of Isolation

Different ways of Calling Garbage Object
++++++++++++++++++++++++++++++++++++++
a. System.gc()
b. Runtime.getInstance().gc() [best suited to call GC]

Finalization
++++++++++++
 => Garbage Collector before cleaning the object,it would internally call finalize()
    to clean the reference associated with the object to avoid "Memory Leaks".
 => protected void finalize() throws Throwable

Case4: On a particular Object, JVM will call finalize() only once.

```
public class Test
{
       static Test t;
      public static void main(String[] args)throws Exception
      {
            Test t1 =new Test();
            System.out.println("T1 HASHCODE :: "+t1.hashCode());
            t1 =null;
            System.gc();//Called GC -> finalize()
            Thread.sleep(2000);


            System.out.println("T HASHCODE :: "+t.hashCode());
            t = null;
            System.gc();//Called GC
            Thread.sleep(2000);
            System.out.println("End of main method...");
      }

      @Override
      public void finalize()
      {
            System.out.println("finalized method called...");
            t = this;
      }
}
```

Output

```
T1 HASHCODE :: 366712642
finalized method called...
T HASHCODE :: 366712642
End of main method...
```

Note:
The behavior of the GC is vendor dependent and varied from JVM to JVM hence we can't expect exact answer for the following
1. What is the algorithm followed by GC.(mark and sweep is the common algorithm)
2. Exactly at what time JVM runs GC.

3. In which order GC identifies the eligible objects.
4. In which order GC destroys the object etc.
5. Whether GC destroys all eligible objects or not.

=> When ever the program runs with low memory then the JVM runs GC, but we can't except exactly at what time.
=> Most of the GC's followed mark & sweap algorithm , but it doesn't mean every GC follows the
   same algorithm.

```
eg#1.
public class Test
{
      static int counter =0;;
      public static void main(String... args){
            for (int i=1;i<=10000000; i++){
                  Test t =new Test();
                  t=null;
            }
      }
      public void finalize() {
            System.out.println("finalized method called:: "+ (++counter));
      }
}
```

Output varied based on i value if we keep increasing to 100,1000,10000,100000,1000000,....

Memory leaks:
=> An object which is not using in our application and it is not eligible for GC such type of objects are called "memory leaks".
=> In the case of memory leaks GC also can't do anything the application will be crashed due to memory problems.
=> In our program if memory leaks present then certain point we will get OutOfMemoryException. Hence if an object is no longer
   required then it's highly recommended to make that object eligible for GC.

```
eg#1.
   Student s1=new Student();
   Student s2=new Student();
   Student s3=new Student();
      ;;;;
      ;;;;
      ;;;;
   Student s10000000 =new Student();
      ;;;;
      ;;;;
      ;;;;
      //program crash =>memory leak(In this case JVM and Garbage Collector can't do anything)
        OutOfMemoryError
```

=> By using monitoring tools we can identify memory leaks.
   HPJ meter
   HP ovo
   IBM Tivoli These are monitoring tools.
   J Probe (or memory management tools)
   Patrol and etc

Note:
What is the difference b/w final,finally and finalize?
Ans. final => It is an access modifer applicable on class,method and varaible
            class   -> inheritance is not possible.
           method   -> we can't override.
           variable -> we can't change the value for the variable during execution[Compile Time constant].

     finally -> It is a block of code which gets executed irrespective of whether exception occurs or not in java application

```
try{
      //risky code
}catch(XXXX e){
      // handling exception
}finally{
      //resource releasing logic
}
```

    finalize() -> It is a method which gets called automatically by the GC, to perform clean up activities associated with the Objects to avoid memory leaks.

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