Garbage Collection:
—------------------
In C++ programming language Developers are responsible for creating objects and destroying objects. In the case id Developers are not destroying objects then we are able to get the following problems.

1. Memory is full with the present application objects, no memory for the next application objects.
2. No security for the data.

In Java , Developers are responsible for creating objects only, Developers are not responsible for Destroying objects.

In Java applications, to destroy objects JAVA has provided an internal component inside the JVM that is “Garbage Collector”.

In Java applications, when Heap memory is full then the Garbage Collector will destroy the useless objects.

IN Java applications, when the application execution is completed , automatically Garbage Collector will destroy objects.

In Java applications, JAVA has provided a process for the developers to destroy the objects explicitly as per the requirement.

If we want to destroy an object in java applications then we have to use the following steps.

1. Make Eligible an object for the Garbage Collection:
   To make an object eligible for the Garbage COllection we have to assign null value to the reference variable.
   A a = new A();
   a = null;

2. Activate Garbage Collector to destroy objects:
   To activate Garbage Collector to destroy objects we have to use the following method.

   System.gc();

   When we access the System.gc() method , internally JVM will access the finalize() method just before destroying the object.

EX:

```
class A{
A(){
System.out.println("Object Creating.....");
}

@Override
protected void finalize() throws Throwable {
System.out.println("Object Destroying......");
}
}
public class Main {
public static void main(String[] args) {
A a = new A();
a = null;
System.gc();
}
}
```

```
Object Creating.....
Object Destroying......
```

Approaches to make an object eligible for Garbage Collection:

---

1. Assign null value to the reference variable.
   EX:

```
class A{
A(){
System.out.println("Object Creating.....");
}

@Override
protected void finalize() throws Throwable {
System.out.println("Object Destroying......");
}
}
public class Main {
public static void main(String[] args) {
A a1 = new A();
a1 = null;
A a2 = new A();
a2 = null;
System.gc();
}
```

```
}
```

```
Object Creating.....
Object Creating.....
Object Destroying......
Object Destroying......
```

2. Reassign the reference variables:
   If assign one reference variable to another reference variable then the reference variable which refers to the previous object is eligible for Garbage Collection.

   EX:

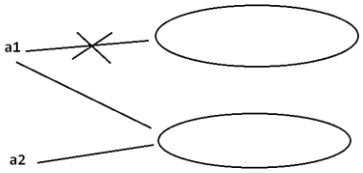
```
class A{
A(){
System.out.println("Object Creating.....");
}

@Override
protected void finalize() throws Throwable {
System.out.println("Object Destroying......");
}
}
public class Main {
public static void main(String[] args) {
A a1 = new A();
A a2 = new A();
a1 = a2;
System.gc();


}
}
```

```
Object Creating.....
Object Creating.....
Object Destroying......
```

A a1 = new A();
A a2 = new A();



This object is Eligible for Garbage Collection

a1 = a2;

3. Creating Object inside the Methods:
   If we provide an Object creation statement inside the methods then JVM will create an object when the respective method is executed, when method execution is completed , automatically the object which we have created inside the method is eligible for Garbage Collection.
   EX:

```
class A{
A(){
System.out.println("A Class Object Creating....");
}
public void finalize(){
System.out.println("A class Object Destroying.....");
}
}
class B{
void m1(){
A a = new A();
}
}
public class Main {
public static void main(String[] args) {
B b = new B();
b.m1();
System.gc();
}
```

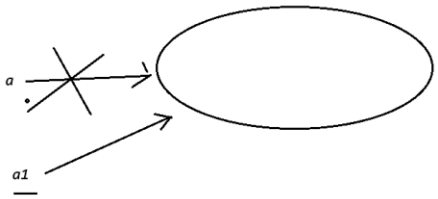
```
}
```

A Class Object Creating....
A class Object Destroying.....

EX:

```
class A{
    A(){
        System.out.println("Object Creating....");
    }

    @Override
    protected void finalize() throws Throwable {
        System.out.println("Object Destroying.....");
    }
}
class B{
    A getA(){
        A a = new A();
        return a;
    }
}
public class Main {
    public static void main(String[] args) {
        B b = new B();
        A a1 = b.getA();
        System.gc();
    }
}
```



EX:

```
class A{
A(){
System.out.println("Object Creating....");
}

@Override
protected void finalize() throws Throwable {
System.out.println("Object Destroying.....");
}
}
class B{
A getA(){
A a = new A();
return a;
}
}
public class Main {
public static void main(String[] args) {
B b = new B();
A a1 = b.getA();
System.gc();
}
}
```

Object Creating....

Approach to activate Garbage Collector and to request Garbage Collector about the Object Destruction:

In Java applications, when we make an object eligible for Garbage Collection then the Garbage Collection may destroy the objects immediately or may not destroy the objects immediately , there is no guarantee to destroy the objects immediately, but in maximum cases Garbage Collector will destroy objects immediately.

If we want to request a Garbage Collector to destroy objects then we have to use the following approaches.

1. By Using the static gc() method from System class:
   System.gc();

2. By Using non static method gc() method from Runtime class:
   Runtime rt = Runtime.getRuntime();
   rt.gc();

   EX:

```
class A{
A(){
System.out.println("Object Creating....");
}

@Override
protected void finalize() throws Throwable {
System.out.println("Object Destroying.....");
}
}
class B{
A getA(){
A a = new A();
return a;
}
}
public class Main {
public static void main(String[] args) {
B b = new B();
b.getA();
Runtime runtime = Runtime.getRuntime();
runtime.gc();
```

```
}
}
```

```
Object Creating....
Object Destroying.....
```

```
Q)Find the valid method calls for gc()?
—---------------------------------------
   1. System.gc(); —-----> Valid
   2. Runtime.gc(); —-----> Invalid
   3. Runtime.getRuntime().gc(); —--> Valid
   4. (new Runtime()).gc(); —-----> Invalid
```

finalize() method:

—----------------

The main intention of finalize() method is

1. To give final intimation about destroying an object.
2. To perform cleanup operations just before destroying an Object.

IN Java applications, the finalize() method will be executed by JVM automatically when we destroy an object.

In Java, finalize() method was declared in the java.lang.Object class like below.

protected void finalize()throws Throwable

EX:

```
class A{

public void finalize(){
System.out.println("finalize()....");
}
}
public class Main {
public static void main(String[] args) {
A a = new A();
a = null;
System.gc();
}
}
```

EX:

```
public class Main {
@Override
protected void finalize() throws Throwable {
System.out.println("finalize()-Test");
}

public static void main(String[] args) {
String str = new String("Durgasoft");
str = null;
System.gc();
}
}
```

Status: No Output.

In the above example, the String class object is eligible for Garbage Collection, so the String class finalize() method will be executed, not the Main class finalize() method.

EX:

```
public class Main {
@Override
protected void finalize() throws Throwable {
System.out.println("finalize()-Test");
}

public static void main(String[] args) {
String str = new String("Durgasoft");
Main m = new Main();
m = null;
System.gc();
}
}
```

Q) In Java applications, if we access the finalize() method explicitly like a normal Java method , will GarbageCollector destroy the respective object?

—--
Ans:
—----
No, if we access finalize() method explicitly then JVM will execute finalize() method like a normal java method , not as per Garbage Collection, when JVM access finalize() method automatically then JVM will perform garbage Collection.

EX:

```
class A{
@Override
protected void finalize() throws Throwable {
System.out.println("finalize()-A");
}
void m1(){
System.out.println("m1-A");
}
}
public class Main {
public static void main(String[] args)throws Throwable {
A a = new A();
a.finalize();
a.m1();
}
}
```

finalize()-A
m1-A

EX:

```
class A{
@Override
protected void finalize() throws Throwable {
System.out.println("finalize()-A");
}

}
public class Main {
public static void main(String[] args)throws Throwable {
A a = new A();
a = null;
System.gc();
System.gc();
```

```
System.gc();
}
}
```

finalize()-A

Q)What is memory Leak?
—----------------------
Ans:
—---
In Java applications, if any object is available without using it in the application and which is not eligible for Garbage Collection then that object is called a Memory Leak.

If More number of memory leaks are identified in the java applications automatically JVM is able to raise an exception like OutofMemoryException, we are not having any solution explicitly ,we have to cache those objects by using third party cache mechanisms.