




# Garbage Collection

- 
- The main objective of Garbage Collector is to free memory by destroying unreachable objects.
  - Garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.
  - Java programs compile to bytecode that can be run on a Java Virtual Machine. When Java programs run on the JVM, objects are created on the heap. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.
  - **Note:** All objects are created in **Heap** Section of memory.



## ► 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 Reference.

## ► 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.

## 2) Re Assigning the Reference Variable:-

- If an Object is No Longer required then Re Assign its Reference Variable to any other object then Old Object is Automatically Eligible for GC.

## 3) Objects Created Inside a Method

- Objects Created Inside a Method are by Default Eligible for Garbage Collection Once Method Completes (once method completes all local variables will be destroyed).

## 4) Island of Isolation

- If an Object doesn't have any Reference variable then it is Always Eligible for GC.
- Even though Object having Reference variable Still sometimes it May be a eligible for Garbage Collection (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, it May Not Destroy Immediately by the Garbage Collector. Whenever JVM Runs Garbage Collector then Only Object will be Destroyed. But when exactly JVM runs GC, We can't Expect. It Depends on JVM and varied from JVM to JVM.
- 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 Accepts 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().

`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.

**Note1: Singleton class:**

For any Java class if we are allowed to create only one object, such type of class is called Singleton class.

Eg: `Runtime`


`BusinessDelegate`

**Note2: Factory Method:**

By using class name if we are calling a method and if that method returns same class object, such type of methods are called factory methods (static factory methods).

Eg: `Runtime r = Runtime.getRuntime();`

CDAC PATNA



**Note:** gc() method present in System Class is Static Method whereas gc() method Present in Runtime Class is Instance Method.

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

- 1) System.gc();
- 2) Runtime.gc();

CDAC PATNA



### Note:

- With respect to Convenience, it is Recommended to Use System.gc() method when compared to Runtime class gc() method.
- With Respect to Performance, it is Recommended to Use Runtime class gc() method when compared with System class gc() method ,because Internally System.gc() method calls Runtime class gc() method.

```
class System
```

```
{  
    public static void gc()
```

```
{
```

```
    Runtime.getRuntime().gc();
```

```
}
```

```
}
```

## Finalization:

- Just Before Destroying an Object Garbage Collector Calls **finalize()** to Perform Cleanup Activities. Once **finalize()** Completes their execution after that Automatically GC Destroys that Object.
- This method helps Garbage Collector to close all the resources used by the object
- After executing **finalize()** method completely, the object automatically gets destroyed.
- The **finalize()** method is a method of **Object** class [ **java.lang.Object.finalize()** ] with the following Prototype.  
**protected void finalize() throws Throwable**
- Based on Our Requirement we can Override **finalize()** method in Our Class to define our own Cleanup Activities.

## Case 1:

- Just before Destroying an Object Garbage Collector Always Calls finalize() 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 DemoGc Class finalize() method.

```
public class DemoGc {  
  
    public static void main(String[] args) {  
        String s = new String("Rohan");  
        s = null;  
        System.gc();  
        System.out.println("End of Main");  
    }  
  
    public void finalize(){  
        System.out.println("finalize Method Called");  
    }  
}
```

- Output: End of Main

- In the given 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 DemoGc Object, then DemoGc object eligible for GC and hence DemoGc Class finalize() method will be executed. in this Case Output is

```
public class DemoGc {  
  
    public static void main(String[] args) {  
        DemoGc s = new DemoGc ();  
        s = null;  
        System.gc();  
        System.out.println("End of Main");  
    }  
  
    public void finalize(){  
        System.out.println("finalize Method Called");  
    }  
}
```


End of Main  
finalize Method Called

finalize Method Called  
End of Main

## Case 2:

- Based on Our Requirement 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 an Object Garbage Collector Always Calls finalize() method.

```
public 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");  
    }  
}
```

- 
- In the given Example finalize() got executed 3 Times, In that 2 Times explicitly by the Programmer and 1 Time by the Garbage Collector.

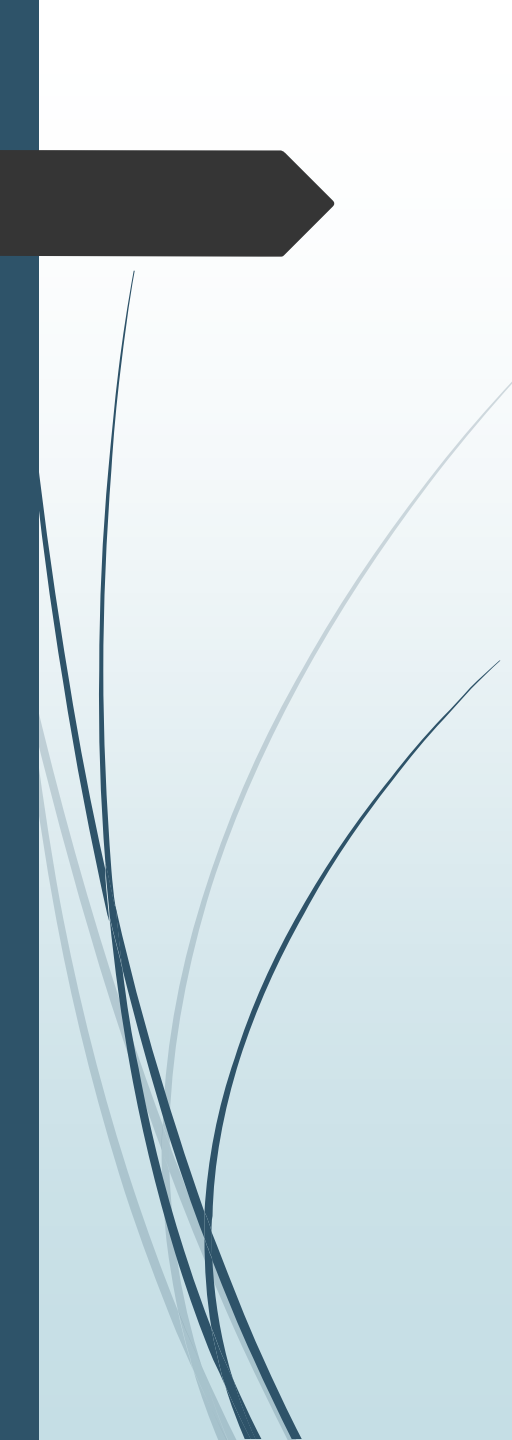
CDAC PATNA






### Case 3:

- If the Programmer Calls finalize() Explicitly and while executing that finalize() if any Exception Occurs and which is Uncaught (that is there is no catch block), then the Program will be terminated Abnormally by raising that exception.
- If Garbage Collector Calls finalize() method and while executing that finalize() method, if any Exception raised which is Uncaught then JVM ignores that exception and rest of the Program will be executed Normally.



```
public class DemoGc {  
  
    public static void main(String[] args) {  
        DemoGc t = new DemoGc();  
        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);  
    }  
  
}
```




- 
- If we are Not Commenting Line 1 then Programmer Calls finalize() method and while executing that finalize() method ArithmeticException raised which is Uncaught. Hence the Program will be terminated abnormally by raising **ArithmeticException**.

In this the Output is  
finalize() Called

Exception in thread "main" java.lang.ArithmeticException: /by zero

CDAC PATNA

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

**Q: Which of the following is true?**

- JVM Ignores Every Exception which are raised while Executing finalize() method. **//False**
- JVM Ignores Only Uncaught Exceptions which are raised while executing finalize() method. **//true**



## ► Advantage of Garbage Collection –

- No manual memory allocation/deallocation handling because unused memory space is automatically handled by garbage collector(part of JVM).

CDAC PATNA



# THANK YOU

CDAC PATNA