**Garbage Collection in Java**

Garbage Collection

o   Overview of Memory Management and Garbage Collection

o   Overview of Java's Garbage Collector

Writing Code That Explicitly Makes Objects Eligible for Garbage Collection

**Important terms :**

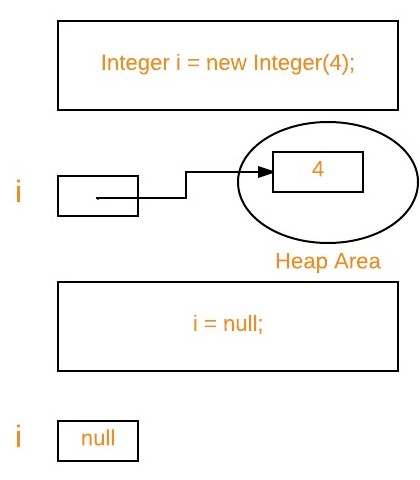
1. **Unreachable objects :**An object is said to be unreachable iff it doesn’t contain any reference to it. Also note that objects which are part of [island of isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/) are also unreachable.

Integer i = new Integer(4);

// the new Integer object is reachable via the reference in 'i'

i = null;

// the Integer object is no longer reachable.



**2.Eligibility for garbage collection :** An object is said to be eligible for GC(garbage collection) iff it is unreachable. In above image, after *i = null;* integer object 4 in heap area is eligible for garbage collection.

**Ways to make an object eligible for GC**

* Even though the programmer is not responsible to destroy useless objects but it is highly recommended to make an object unreachable(thus eligible for GC) if it is no longer required.
* There are generally four different ways to make an object eligible for garbage collection.
  + 1. Nullifying the reference variable
    2. Re-assigning the reference variable
    3. Object created inside method
    4. [Island of Isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/)

Once we made object eligible for garbage collection, it may not destroy immediately by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed. But when JVM runs Garbage Collector, we can not expect.

 We can also request JVM to run Garbage Collector. There are two ways to do it :

* 1. **Using *System.gc()* method** : System class contain static method *gc()* for requesting JVM to run Garbage Collector.
  2. **Using *Runtime.getRuntime().gc()* method** : [Runtime class](https://www.geeksforgeeks.org/java-lang-runtime-class-in-java/) allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

// Java program to demonstrate requesting

// JVM to run Garbage Collector

public class Test

{

public static void main(String[] args) throws InterruptedException

{

Test t1 = new Test();

Test t2 = new Test();

// Nullifying the reference variable

t1 = null;

// requesting JVM for running Garbage Collector

System.gc();

// Nullifying the reference variable

t2 = null;

// requesting JVM for running Garbage Collector

Runtime.getRuntime().gc();

}

@Override

// finalize method is called on object once

// before garbage collecting it

protected void finalize() throws Throwable

{

System.out.println("Garbage collector called");

System.out.println("Object garbage collected : " + this);

}

}

Output:

Garbage collector called

Object garbage collected : Test@46d08f12

Garbage collector called

Object garbage collected : Test@481779b8

* **Note :**
  + 1. There is no guarantee that any one of above two methods will definitely run Garbage Collector.
    2. The call *System.gc()* is effectively equivalent to the call : *Runtime.getRuntime().gc()*

**Finalization**

* Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.
* *finalize()* method is present in [Object class](https://www.geeksforgeeks.org/object-class-in-java/) with following prototype.
* protected void finalize() throws Throwable

Based on our requirement, we can override *finalize()* method for perform our cleanup activities like closing connection from database.

**Note :**

* 1. The *finalize()*method called by Garbage Collector not [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). Although Garbage Collector is one of the module of JVM.
  2. [Object class](https://www.geeksforgeeks.org/object-class-in-java/) *finalize()* method has empty implementation, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.
  3. The *finalize()*method is never invoked more than once for any given object.
  4. If an uncaught exception is thrown by the *finalize()* method, the exception is ignored and finalization of that object terminates.

For examples on *finalize()* method, please see [Output of Java programs | Set 10 (Garbage Collection)](https://www.geeksforgeeks.org/output-of-java-programs-set-10-garbage-collection/)

**Let’s take a real-life example, where we use the concept of garbage collector.**

Suppose you go for the internship at GeeksForGeeks and their you were told to write a program, to count the number of Employees working in the company(excluding interns).To make this program, you have to use the concept of a garbage collector.  
This is the actual task you were given at the company:-

**Q.**Write a program to create a class called Employee having the following data members.  
1.An ID for storing unique id allocated to every employee.  
2.Name of employee.  
3.age of an employee.

Also, provide the following methods-

1. A parameterized constructor to initialize name and age. The ID should be initialized in this constructor.
2. A method show() to display ID, name, and age.
3. A method showNextId() to display the ID of the next employee.

Now any beginner, who doesn’t have knowledge on garbage collector will code like this:

//Program to count number

//of employees working

//in a company

class Employee

{

private int ID;

private String name;

private int age;

private static int nextId=1;

//it is made static because it

// is keep common among all and

// shared by all objects

public Employee(String name,int age)

{

this.name = name;

this.age = age;

this.ID = nextId++;

}

public void show()

{

System.out.println

("Id="+ID+"\nName="+name+"\nAge="+age);

}

public void showNextId()

{

System.out.println

("Next employee id will be="+nextId);

}

}

class UseEmployee

{

public static void main(String []args)

{

Employee E=new Employee("GFG1",56);

Employee F=new Employee("GFG2",45);

Employee G=new Employee("GFG3",25);

E.show();

F.show();

G.show();

E.showNextId();

F.showNextId();

G.showNextId();

{ //It is sub block to keep

// all those interns.

Employee X=new Employee("GFG4",23);

Employee Y=new Employee("GFG5",21);

X.show();

Y.show();

X.showNextId();

Y.showNextId();

}

//After countering this brace, X and Y

//will be removed.Therefore,

//now it should show nextId as 4.

E.showNextId();//Output of this line

//should be 4 but it will give 6 as output.

}

}

Output:

Id=1

Name=GFG1

Age=56

Id=2

Name=GFG2

Age=45

Id=3

Name=GFG3

Age=25

Next employee id will be=4

Next employee id will be=4

Next employee id will be=4

Id=4

Name=GFG4

Age=23

Id=5

Name=GFG5

Age=21

Next employee id will be=6

Next employee id will be=6

Next employee id will be=6

**Now to get the correct output:**  
Now garbage collector(gc) will see 2 objects free. Now to decrement nextId,gc(garbage collector) will call method finalize() only when we programmers have override it in our class. And as mentioned previously, we have to request gc(garbage collector) and for this, we have to write the following 3 steps before closing brace of sub-block.

1. Set references to null(i.e X = Y = null;)
2. Call, System.gc();
3. Call, System.runFinalization();

Now the correct code for counting the number of employees(excluding interns)

// Correct code to count number

// of employees excluding interns.

class Employee

{

private int ID;

private String name;

private int age;

private static int nextId=1;

//it is made static because it

// is keep common among all and

// shared by all objects

public Employee(String name,int age)

{

this.name = name;

this.age = age;

this.ID = nextId++;

}

public void show()

{

System.out.println

("Id="+ID+"\nName="+name+"\nAge="+age);

}

public void showNextId()

{

System.out.println

("Next employee id will be="+nextId);

}

protected void finalize()

{

--nextId;

//In this case,

//gc will call finalize()

//for 2 times for 2 objects.

}

}

// it is closing brace of Employee class

class UseEmployee

{

public static void main(String []args)

{

Employee E=new Employee("GFG1",56);

Employee F=new Employee("GFG2",45);

Employee G=new Employee("GFG3",25);

E.show();

F.show();

G.show();

E.showNextId();

F.showNextId();

G.showNextId();

{

//It is sub block to keep

// all those interns.

Employee X=new Employee("GFG4",23);

Employee Y=new Employee("GFG5",21);

X.show();

Y.show();

X.showNextId();

Y.showNextId();

X = Y = null;

System.gc();

System.runFinalization();

}

E.showNextId();

}

}

Output:

Id=1

Name=GFG1

Age=56

Id=2

Name=GFG2

Age=45

Id=3

Name=GFG3

Age=25

Next employee id will be=4

Next employee id will be=4

Next employee id will be=4

Id=4

Name=GFG4

Age=23

Id=5

Name=GFG5

Age=21

Next employee id will be=6

Next employee id will be=6

Next employee id will be=4

MCQ’s

* 1. Which of the following has the highest memory requirement?  
     a) Heap  
     b) Stack  
     c) JVM  
     d) Class

Answer:c  
Explanation: JVM is the super set which contains heap, stack, objects, pointers, etc.

2. Where is a new object allocated memory?  
a)Youngspace  
b)Oldspace  
c)YoungorOldspacedependingonspaceavailability  
d) JVM

Answer:a  
Explanation: A new object is always created in young space. Once young space is full, a special young collection is run where objects which have lived long enough are moved to old space and memory is freed up in young space for new objects.

3. Which of the following is a garbage collection technique?  
a)Cleanupmodel  
b)Markandsweepmodel  
c)Spacemanagementmodel  
d) Sweep model

Answer:b  
Explanation: A mark and sweep garbage collection consists of two phases, the mark phase and the sweep phase. I mark phase all the objects reachable by java threads, native handles and other root sources are marked alive and others are garbage. In sweep phase, the heap is traversed to find gaps between live objects and the gaps are marked free list used for allocating memory to new objects.

1. What is -Xms and -Xmx while starting jvm?  
   a)Initial;Maximummemory  
   b)Maximum;Initialmemory  
   c)Maximummemory  
   d) Initial memory

Answer:a  
Explanation: JVM will be started with Xms amount of memory and will be able to use a maximum of Xmx amount of memory. java -Xmx2048m -Xms256m.

1. Which exception is thrown when java is out of memory?  
   a)MemoryFullException  
   b)MemoryOutOfBoundsException  
   c)OutOfMemoryError  
   d) MemoryError

Answer:c  
Explanation: The Xms flag has no default value, and Xmx typically has a default value of 256MB. A common use for these flags is when you encounter a java.lang.OutOfMemoryError.