Garbage Collection in Java

Difficulty Level : Easy   Last Updated : 07 Dec, 2021

Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

What is Garbage Collection?
In C/C++, a programmer is responsible for both the creation and destruction of objects. Usually, programmer neglects the destruction of useless objects. Due to this negligence, at a certain point, sufficient memory may not be available to create new objects, and the entire program will terminate abnormally, causing **OutOfMemoryErrors**.

But in Java, the programmer need not care for all those objects which are no longer in use. Garbage collector destroys these objects. The main objective of Garbage Collector is to free heap memory by destroying **unreachable objects**. The garbage collector is the best example of the Daemon thread as it is always running in the background.

How Does Garbage Collection in Java works?
Java 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. An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. The programmer does not need to mark objects to be deleted explicitly. The garbage collection implementation lives in the JVM.

Types of Activities in Java Garbage Collection

Two types of garbage collection activity usually happen in Java. These are:

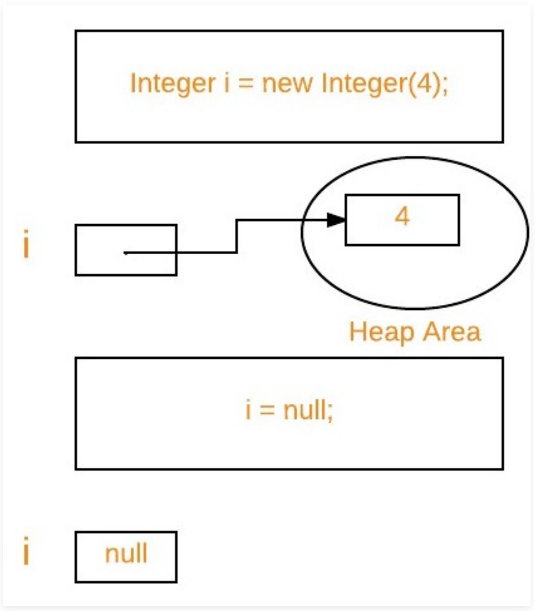
1. **Minor or incremental Garbage Collection:** It is said to have occurred when unreachable objects in the young generation heap memory are removed.

2. **Major or Full Garbage Collection:** It is said to have occurred when the objects that survived the minor garbage collection and copied into the old generation or permanent generation heap memory are removed. When compared to the young generation, garbage collection happens less frequently in the old generation.

Important Concepts Related to Garbage Collection in Java

1. Unreachable objects: An object is said to be unreachable if it doesn't contain any reference to it. Also, note that objects which are part of the island of isolation 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) if it is unreachable. After *i = null*, integer object 4 in the heap area is suitable for garbage collection in the above image.

Ways to make an object eligible for Garbage Collector

- Even though the programmer is not responsible for destroying 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 ways to make an object eligible for garbage collection.
  1. Nullifying the reference variable
  2. Re-assigning the reference variable

3. An object created inside the method
4. Island of Isolation

Ways for requesting JVM to run Garbage Collector

- Once we make an 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 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.
  3. There is no guarantee that any of the above two methods will run Garbage Collector.
  4. 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 with the following prototype.

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

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

1. The *finalize()* method is called by Garbage Collector, not JVM. However, Garbage Collector is one of the modules of JVM.
2. Object class *finalize()* method has an empty implementation. Thus, it is recommended to override the *finalize()* method to dispose of system resources or perform other cleanups.
3. The *finalize()* method is never invoked more than once for any object.
4. If an uncaught exception is thrown by the *finalize()* method, the exception is ignored, and the finalization of that object terminates.

Advantages of Garbage Collection in Java

The advantages of Garbage Collection in Java are:

- It makes java memory-efficient because the garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector(a part of JVM), so we don't need extra effort.

Real-World Example

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

Question: Suppose you go for the internship at GeeksForGeeks, and 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:

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 know Garbage Collector in Java will code like this:

```
// Java 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.

          // Output of this line
        E.showNextId();
        // 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 to finalize() only when we programmers have overridden 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)

```java
// 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.
    }
}

public 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
```

Related Articles:

- How to Make Object Eligible for Garbage Collection in Java?
- Island of Isolation in Java

- Output of Java programs | Set 10 (Garbage Collection)
- How to Find Max Memory, Free Memory, and Total Memory in Java?
- How JVM Works – JVM Architecture?

This article is contributed by **Chirag Agarwal and Gaurav Miglani**. Please write comments if you find anything incorrect or you want to share more information about the topic discussed above.