Garbage collection in Java is an automatic memory management process that helps Java programs run efficiently.

- Objects are created on the heap area.
- Eventually, some objects will no longer be needed.
- Garbage collection is an automatic process that removes unused objects from heap.

Working of Garbage Collection

- It identifies which objects are still in use (referenced) and which are not in use (unreferenced).
- It removes the objects that are unreachable (no longer referenced).
- The programmer does not need to mark objects to be deleted explicitly. Garbage collection is implemented within the JVM.

Types of Activities in Java Garbage Collection

Java heap is divided into generations:

- **Young Generation:** In this new objects are allocated.
- **Old Generation:** In this long-lived objects are stored.

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

- **Minor or incremental Garbage Collection (GC)**: This occurs when unreachable objects in the Young Generation heap memory are removed.

- **Major or Full Garbage Collection (GC)**: This happens when objects that survived minor garbage collection are removed from the Old Generation heap memory. It occurs less frequently than minor garbage collection.

Key Concepts on Garbage Collection

1. Unreachable Objects

An object becomes unreachable if it does not contain any reference to it.

```
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. Making Objects Eligible for GC

An object is said to be eligible for 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.

How to Make an Object Eligible for Garbage Collection?

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.

- Nullifying the reference variable (obj = null).

- Re-assigning the reference variable (obj = new Object()).
- An object created inside the method (eligible after method execution).
- Island of Isolation (Objects that are isolated and not referenced by any reachable objects).

3. Requesting Garbage Collection

- Once an object is eligible for garbage collection, it may not be destroyed immediately.The garbage collector runs at the JVM's discretion and you cannot predict when it will occur.
- We can also request JVM to run Garbage Collector. There are two ways to do it. fiest using System.gc() and second using Runtime.getRuntime().gc():
- **Using System.gc()**: This static method requests the JVM to perform garbage collection.
- **Using Runtime.getRuntime().gc():** This method also requests garbage collection through the Runtime class.

```
System.gc();
// OR
Runtime.getRuntime().gc();
```

4. The finalize() Method (Deprecated in Java 9+)

Before destroying an object, the garbage collector calls the finalize() method to perform cleanup activities. The method is defined in the Object class as follows:

```
@Override
protected void finalize() throws Throwable {
System.out.println("GC cleaning up...");
}
```

Note:

- finalize() method is deprecated since Java 9 because it is unpredictable and can cause performance issues.
- Alternatives like try-with-resources or explicit cleanup methods are preferred.
- The garbage collector calls finalize() at most once per object.
- Exceptions thrown in finalize() are ignored.

Employee Management System Using Garbage Collection Concept

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

Problem Statement: 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:

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

Common Beginner Approach (Without Garbage Collection)

Now any beginner, who does not know Garbage Collector in Java will code like this:

Now the correct code for counting the 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
```

Now:

```
X = Y = null;
System.gc();
System.runFinalization();
```

This line:

- Removes references to `X` and `Y` → both objects become **unreachable**
- `System.gc()` → *requests* GC
- `System.runFinalization()` → forces JVM to run finalizers

Thus `finalize()` is called for both objects → `nextId` decremented twice.

🧮 `nextId` = 6 − 2 = **4**

Advantages of Garbage Collection

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.
-