**Q1.public** **class** UnreachableObject {

**private** String name;

**public** UnreachableObject(String name) {

**this**.name = name;

}

**public** **void** display() {

System.***out***.println("Display method called for: " + name);

// Create an object and call show method on it

UnreachableObject obj = **new** UnreachableObject("Inner Object");

obj.show();

}

**public** **void** show() {

System.***out***.println("Show method called for: " + name);

// Just display a message instead of calling display method again

System.***out***.println("Reached show method, not calling display again.");

}

@Override

**protected** **void** finalize() **throws** Throwable {

System.***out***.println("Object '" + name + "' is being garbage collected");

}

**public** **static** **void** main(String[] args) {

UnreachableObject obj = **new** UnreachableObject("Main Object");

obj.display();

// Explicitly invoke garbage collector

System.*gc*();

// Wait for a short period to allow the garbage collector to potentially execute finalize()

**try** {

Thread.*sleep*(1000); // Sleep for 1 second

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

o/p

Display method called for: Main Object

Show method called for: Inner Object

Reached show method, not calling display again.

Object 'Inner Object' is being garbage collected

Q2 **public** **class** Q2 {

**private** String name;

**public** Q2(String name) {

**this**.name = name;

}

@Override

**protected** **void** finalize() **throws** Throwable {

System.***out***.println("Object '" + name + "' is being garbage collected");

}

**public** **static** **void** main(String[] args) {

// Create two instances of ReassigningReference

Q2 obj1 = **new** Q2("Object 1");

Q2 obj2 = **new** Q2("Object 2");

// Reassign the reference

obj1 = obj2;

// Explicitly invoke garbage collector

System.*gc*();

// Wait for a short period to allow the garbage collector to potentially execute finalize()

**try** {

Thread.*sleep*(1000); // Sleep for 1 second

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

Object 'Object 1' is being garbage collected

The reference **obj1** is then reassigned to **obj2**, which means **obj1** no longer references the first object created.

Q3. **public** **class** Q3 {

**private** String name;

**public** Q3(String name) {

**this**.name = name;

}

@Override

**protected** **void** finalize() **throws** Throwable {

System.***out***.println("Object '" + name + "' is being garbage collected");

}

**public** **static** **void** main(String[] args) {

// Create an instance of NullifiedReference

Q3 obj = **new** Q3("Object");

// Nullify the object reference

obj = **null**;

// Explicitly invoke garbage collector

System.*gc*();

**try** {

Thread.*sleep*(1000); // Sleep for 1 second

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

Object 'Object' is being garbage collected

Q4 **public** **class** Q4 {

**private** String name;

**public** Q4(String name) {

**this**.name = name;

}

@Override

**protected** **void** finalize() **throws** Throwable {

System.***out***.println("Object '" + name + "' is being garbage collected");

}

**public** **static** **void** main(String[] args) {

// Creating an anonymous object of AnonymousObject class

**new** Q4("Anonymous Object");

// Explicitly invoke garbage collector

System.*gc*();

**try** {

Thread.*sleep*(1000); // Sleep for 1 second

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

Object 'Anonymous Object' is being garbage collected

Q5 **public** **class** Data {

**private** **int** intValue;

**private** **double** doubleValue;

**public** Data(**int** intValue, **double** doubleValue) {

**this**.intValue = intValue;

**this**.doubleValue = doubleValue;

}

**public** **void** setValues(**int** intValue, **double** doubleValue) {

**this**.intValue = intValue;

**this**.doubleValue = doubleValue;

}

**public** **void** updateValues(**int** intValueIncrement, **double** doubleValueIncrement) {

**this**.intValue += intValueIncrement;

**this**.doubleValue += doubleValueIncrement;

}

@Override

**protected** **void** finalize() **throws** Throwable {

System.***out***.println("Object '" + "' is being garbage collected");

}

**public** **static** **void** main(String[] args) {

// Create objects of Data class

Data obj1 = **new** Data(10, 20.5);

Data obj2 = **new** Data(30, 40.5);

// Calculate memory before making objects unreachable

**long** totalMemoryBefore = Runtime.*getRuntime*().totalMemory();

**long** freeMemoryBefore = Runtime.*getRuntime*().freeMemory();

System.***out***.println("Total memory before: " + totalMemoryBefore);

System.***out***.println("Free memory before: " + freeMemoryBefore);

// Make objects unreachable

obj1 = **null**;

obj2 = **null**;

// Run garbage collector

System.*gc*();

// Calculate memory after garbage collection

**long** totalMemoryAfter = Runtime.*getRuntime*().totalMemory();

**long** freeMemoryAfter = Runtime.*getRuntime*().freeMemory();

System.***out***.println("Total memory after: " + totalMemoryAfter);

System.***out***.println("Free memory after: " + freeMemoryAfter);

}

}

Total memory before: 132120576

Free memory before: 130797296

Object '' is being garbage collected

Object '' is being garbage collected

Total memory after: 8388608

Free memory after: 7726800

Q6 **import java.util.ArrayList;**

**import java.util.Date;**

**import java.util.List;**

**public class MemoryIntensiveProgram {**

**public static void main(String[] args) {**

**List<Object> objects = new ArrayList<>();**

**Runtime runtime = Runtime.getRuntime();**

**long startTime = System.currentTimeMillis();**

**// Creating a large number of objects to consume memory**

**for (int i = 0; i < 1000000; i++) {**

**objects.add(new Object());**

**if (i % 10000 == 0) {**

**long currentTime = System.currentTimeMillis();**

**System.out.println("Timestamp: " + new Date(currentTime));**

**System.out.println("Heap size: " + runtime.totalMemory());**

**System.out.println("Free memory: " + runtime.freeMemory());**

**System.out.println("---------------------------");**

**}**

**}**

**long endTime = System.currentTimeMillis();**

**System.out.println("Time taken to create objects: " + (endTime - startTime) + " milliseconds");**

**}**

**}**

1. The program creates a list **objects** to store a large number of objects.
2. It gets the runtime instance using **Runtime.getRuntime()** which provides information about the runtime environment.
3. It records the start time using **System.currentTimeMillis()**.
4. Inside the loop, it creates a large number of objects and adds them to the list.
5. Every 10,000 iterations, it prints the current timestamp along with the heap size and free memory.
6. After the loop finishes, it calculates the time taken to create objects and prints it.
7. To use the G1 garbage collector, you can specify it using the following JVM argument:
8. rubyCopy code
9. -XX:+UseG1GC
10. For example, you can run the program using:
11. rubyCopy code
12. java -XX:+UseG1GC MemoryIntensiveProgram
13. This will use the G1 garbage collector for garbage collection during program execution.

**G1 Garbage Collector**

**The G1 (Garbage First) garbage collector is one of the garbage collectors available in the Java Virtual Machine (JVM) for managing memory and reclaiming memory used by objects that are no longer needed by the application. It was introduced in Java 7 as an experimental feature and became the default garbage collector in Java 9.**

**G1 is designed to provide better performance and more predictable pause times compared to the traditional garbage collectors like the Concurrent Mark-Sweep (CMS) collector and the Parallel collector, especially for applications with large heaps.**

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.List;

**public** **class** MemoryIntensiveProgram {

**public** **static** **void** main(String[] args) {

List<Object> objects = **new** ArrayList<>();

Runtime runtime = Runtime.*getRuntime*();

**long** startTime = System.*currentTimeMillis*();

// Creating a large number of objects to consume memory

**for** (**int** i = 0; i < 10000; i++) {

objects.add(**new** Object());

**if** (i % 10000 == 0) {

**long** currentTime = System.*currentTimeMillis*();

System.***out***.println("Timestamp: " + **new** Date(currentTime));

System.***out***.println("Heap size: " + runtime.totalMemory());

System.***out***.println("Free memory: " + runtime.freeMemory());

System.***out***.println("---------------------------");

}

}

**long** endTime = System.*currentTimeMillis*();

System.***out***.println("Time taken to create objects: " + (endTime - startTime) + " milliseconds");

}

}

-XX:+UseG1GC

Q6

**public class Student {**

**private String name;**

**private int age;**

**private String course;**

**public Student(String name, int age, String course) {**

**this.name = name;**

**this.age = age;**

**this.course = course;**

**}**

**@Override**

**protected void finalize() throws Throwable {**

**System.out.println("Student " + name + " is being garbage collected");**

**}**

**public static void main(String[] args) {**

**// Creating students**

**Student student1 = new Student("Alice", 20, "Computer Science");**

**Student student2 = new Student("Bob", 21, "Engineering");**

**// Display initial memory usage**

**displayMemoryUsage("Initial memory usage");**

**// Creating some additional objects to consume memory**

**for (int i = 0; i < 100000; i++) {**

**new Object();**

**}**

**// Display memory usage after creating additional objects**

**displayMemoryUsage("Memory usage after creating additional objects");**

**// Making student objects unreachable**

**student1 = null;**

**student2 = null;**

**// Request garbage collection**

**System.gc();**

**try {**

**Thread.sleep(1000); // Sleep for 1 second**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**// Display memory usage after garbage collection**

**displayMemoryUsage("Memory usage after garbage collection");**

**}**

**private static void displayMemoryUsage(String message) {**

**Runtime runtime = Runtime.getRuntime();**

**long totalMemory = runtime.totalMemory();**

**long freeMemory = runtime.freeMemory();**

**long usedMemory = totalMemory - freeMemory;**

**System.out.println(message);**

**System.out.println("Total Memory: " + totalMemory);**

**System.out.println("Free Memory: " + freeMemory);**

**System.out.println("Used Memory: " + usedMemory);**

**System.out.println("------------------------------------------");**

**}**

**}**

Initial memory usage

Total Memory: 132120576

Free Memory: 130797304

Used Memory: 1323272

------------------------------------------

Memory usage after creating additional objects

Total Memory: 132120576

Free Memory: 128974848

Used Memory: 3145728

------------------------------------------

Student Alice is being garbage collected

Student Bob is being garbage collected

Memory usage after garbage collection

Total Memory: 8388608

Free Memory: 7516088

Used Memory: 872520

------------------------------------------