Laboratory 3

Name:-LAKSARA K.Y

Index.No:-210329E

Section 1

```
// Import the collection you intend to test. For example, ArrayList.
// You will have to import the parent classes also

import java.util.List;
import java.util.ArrayList;


public class PerformanceTest {
    public static void main(String[] args) {

        // First declarer a class for SomeObject.
        // For example a StudentRecord, SalesTransaction, SensorData, etc.
        // Then create a collection of those objects

        List<MathsMarks> list = new ArrayList<MathsMarks>();

        // You will be changing size variable from 10 to 100, 100, 100,
10000, etc
        // to get different performance values
        int size = 10000;

        for (int i = 0; i <= size; i++) {
            list.add(new MathsMarks(i));
        };

        // Get a Java runtime instance.
        // You need this to obtain system details at runtime.
        // RunTime class is available as a "static" class. What does that
mean?

        Runtime runtime = Runtime.getRuntime();

        // Get the runtime to execute the garbage collector.
        // This is making a direct request to run the GC.
        // Otherwise, the system decides when to run the GC.

        runtime.gc();

        // Calculate the amount of memory used by your program

        long memory = runtime.totalMemory() - runtime.freeMemory();
         int objSize=4;

        // Print the performance test data to the console
        // Find the size of the SomeObject instance that you are storing.
Let's call it objSize


        System.out.println("Size of the stored object: " + objSize + "
bytes");
        int calculatedSize = objSize*size;
        System.out.println("Size of the stored collection: " + calculatedSize
+ " bytes");
        System.out.println("Used memory: " + memory + " bytes");
```

```
        System.out.println("Memory overhead: " + (memory-calculatedSize) + "
bytes");
    }
}
class MathsMarks {
    int marks;

    public MathsMarks(int marks){
        this.marks = marks;

    }
}
```

Above Same Code As Screenshots

```
// Import the collection you intend to test. For example, ArrayList.
// You will have to import the parent classes also

import java.util.List;
import java.util.ArrayList;


no usages
public class PerformanceTest {
    no usages
    public static void main(String[] args) {

        // First declarer a class for SomeObject.
        // For example a StudentRecord, SalesTransaction, SensorData, etc.
        // Then create a collection of those objects

        List<MathsMarks> list = new ArrayList<MathsMarks>();

        // You will be changing size variable from 10 to 100, 100, 100, 10000, etc
        // to get different performance values
        int size = 10000;

        for (int i = 0; i <= size; i++) {
            list.add(new MathsMarks(i));
        };

        // Get a Java runtime instance.
        // You need this to obtain system details at runtime.
        // RunTime class is available as a "static" class. What does that mean?

        Runtime runtime = Runtime.getRuntime();

```

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

        // Get the runtime to execute the garbage collector.
        // This is making a direct request to run the GC.
        // Otherwise, the system decides when to run the GC.

        runtime.gc();

        // Calculate the amount of memory used by your program

        long memory = runtime.totalMemory() - runtime.freeMemory();
         int objSize=4;

        // Print the performance test data to the console
        // Find the size of the SomeObject instance that you are storing. Let's call it objSize


        System.out.println("Size of the stored object: " + objSize + " bytes");
        int calculatedSize = objSize*size;
        System.out.println("Size of the stored collection: " + calculatedSize + " bytes");
        System.out.println("Used memory: " + memory + " bytes");
        System.out.println("Memory overhead: " + (memory-calculatedSize) + " bytes");
    }
}
3 usages
class MathsMarks {
    1 usage
    int marks;

    1 usage
    public MathsMarks(int marks){
        this.marks = marks;
```

```
3 usages
class MathsMarks {
    1 usage
    int marks;

    1 usage
    public MathsMarks(int marks){
        this.marks = marks;

    }
}
```

Section 2

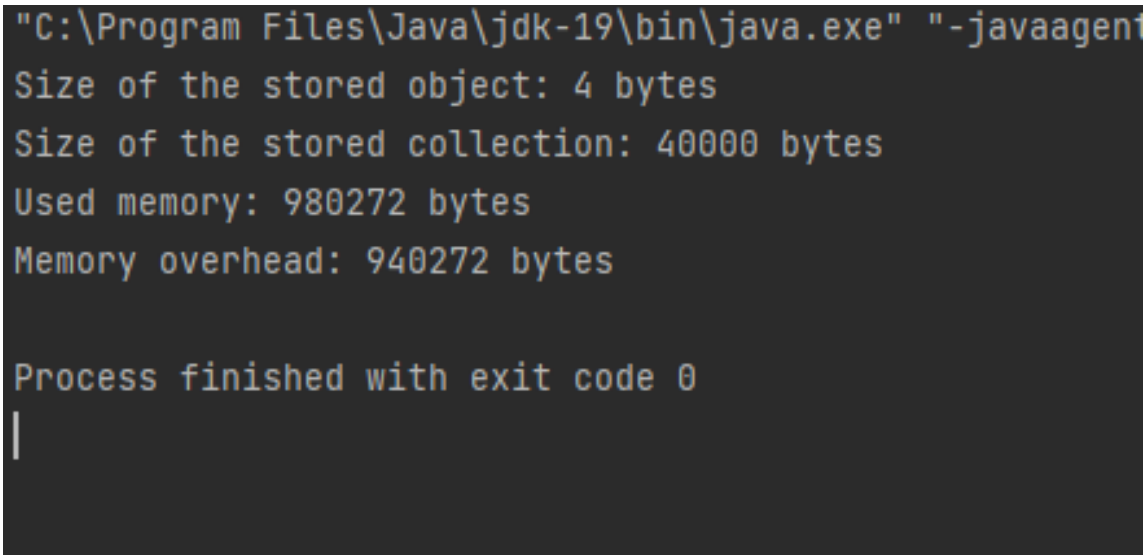
- **We need to import the appropriate collection classes relevant to the given type of collection.**
- **Then We have to declare a variable for the desired collection and initialize the variable.**
- **Then we can use the previous code by modifying the above code appropriately.**
- **Finally, we can run the program.**

Section 3

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaage
Size of the stored object: 4 bytes
Size of the stored collection: 40 bytes
Used memory: 736896 bytes
Memory overhead: 736856 bytes

Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaage
Size of the stored object: 4 bytes
Size of the stored collection: 400 bytes
Used memory: 738992 bytes
Memory overhead: 738592 bytes

Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:
Size of the stored object: 4 bytes
Size of the stored collection: 4000 bytes
Used memory: 1219272 bytes
Memory overhead: 1215272 bytes

Process finished with exit code 0
```


"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent
Size of the stored object: 4 bytes
Size of the stored collection: 40000 bytes
Used memory: 980272 bytes
Memory overhead: 940272 bytes
Process finished with exit code 0

Section 4

Object Amount	Object Size	Stored Collection Size	Memory Usage	Memory Overhead
10	4 bytes	40 bytes	736896 bytes	736856 bytes
100	4 bytes	400 bytes	738992 bytes	738592 bytes
1000	4 bytes	4000 bytes	1219272 bytes	1215272 bytes
10000	4 bytes	40000 bytes	980272 bytes	940272 bytes

Section 5

- **The size of the stored object affect the memory of collection.**

- **A collection's memory overhead is the sum of the memory consumed by the collection and the stored items.**

- **The size of the collection affect the memory usage of the collection.**

- **ArrayList often uses more memory than LinkedList because it allocates space for every entry in the array, even if some of them are null.**

- **Because of the intrinsic data structures used for hashing and storing keys and values, HashSet and HashMap may have a larger memory cost than ArrayList and LinkedList.**