PART A ◆ THEORY QUESTIONS

Q1)What are local and anonymous classes?

Ans:-Local Class:-
-A local class is a class that is declared inside a method, constructor, or block of a class.
-Its scope is limited to that block only, so it cannot be accessed from outside.
-Local classes are mainly used to encapsulate helper logic that is relevant only within a specific method.
Characteristics:
-Declared inside a method, constructor, or block
-Scope is restricted to that block
-Can access only final or effectively final local variables
-Cannot be declared as static
-Can contain fields, methods, and constructors
-Improves code readability by hiding unnecessary class details

2.Anonymous class
-An anonymous class is a nameless class that is declared and instantiated in a single statement.

-It is typically used to provide a one-time implementation of an interface or abstract class.
-Anonymous classes are useful when a class is needed only once and defining a separate named class would be unnecessary.
-Characteristics:
-Does not have a class name
-Created at the time of object creation
-Used to override methods of a class or interface
-Cannot define constructors
-Commonly used with interfaces and abstract classes
-Reduces boilerplate code for short implementations

Q2)What is a String in Java?

Ans:-In Java, a String is an object that represents a sequence of characters.
-The String class is present in the java.lang package and is one of the most commonly used classes in Java.
-A String in Java is immutable, which means once a String object is created, its value cannot be changed.
-Any modification to a String results in the creation of a new String object.
*Key Characteristics of String
-Strings are objects, not primitive data types
-The String class is final, so it cannot be inherited
-Strings are immutable
-Stored in a special memory area called the String Constant Pool (SCP)
-Supports Unicode characters
-Provides many built-in methods for string manipulation
-Creation of String Objects

Strings can be created in two ways:
1)Using String literals
2)Stored in the String Constant Pool

1)Using the new keyword
-Stored in heap memory

*Why Strings Are Immutable
-Improves security (used in passwords, URLs, class loaders)
-Supports String Constant Pool memory optimization
-Makes Strings thread-safe
-Improves performance through caching (hashcode)

Q3)Difference between String, StringBuilder, and StringBuffer.

Ans:1)-String
-String objects are immutable, meaning their value cannot be changed once created.
-Any modification creates a new object, which increases memory usage.
-Thread-safe by default because the value never changes.

2)StringBuilder
-StringBuffer is mutable, so its content can be changed.
-It is thread-safe because its methods are synchronized.
-Due to synchronization overhead, it is slower than StringBuilder.

3)StringBuffer
-StringBuilder is also mutable.
-It is not thread-safe, but provides better performance.
-Best choice when string manipulation is required in a single-threaded environment.

Q4)Why are Strings immutable?

Ans:-In Java, a String is immutable, which means once a String object is created, its value cannot be changed. Any operation that appears to modify a String actually creates a new object in memory.

Strings are immutable for the following reasons:
1. Security
-Strings are commonly used to store sensitive data such as passwords, file paths, database URLs, and network connections.
- If Strings were mutable, their values could be changed after being validated, leading to serious security issues. Immutability ensures that the value remains constant and secure.

2. String Constant Pool Optimization
-Java uses a special memory area called the String Constant Pool to store String literals.
- Multiple references can point to the same String object. Since Strings are immutable, sharing the same object is safe and memory-efficient.
-If Strings were mutable, changing one reference would affect all others pointing to the same object.

3. Thread Safety
-Strings are inherently thread-safe because they cannot be modified after creation.
- Multiple threads can access the same String object without synchronization, avoiding concurrency issues.

4. Hashcode Caching
-The hashcode of a String is calculated once and cached. Strings are widely used as keys in hash-based collections like HashMap and HashSet.
- Immutability guarantees that the hashcode remains constant, ensuring proper retrieval of objects from these collections.

5. Reliability and Predictability
-Immutability ensures that a String's value does not change during program execution, making programs more reliable and easier to understand, debug, and maintain.

Q5)What is a String Pool in Java?

Ans:-The String Pool (also called the String Constant Pool) is a special memory area in the Java Heap that stores String literals.
-It is used to reduce memory usage by allowing multiple references to share the same String object.
-When a String literal is created, Java first checks the String Pool:
-If the String already exists, Java returns a reference to the existing object.
-If it does not exist, Java creates a new String object and stores it in the pool.
Why is String Pool used?
The String Pool is used to:
-Improve memory efficiency
-Avoid creating duplicate String objects
-Improve performance by reusing existing objects

How does String Pool work?
-When we create Strings in different ways:
1. Using String literal
String s1 = "Java";
String s2 = "Java";
Both s1 and s2 point to the same object in the String Pool.

2. Using new keyword
String s3 = new String("Java");
-A new object is created in heap memory, even if "Java" already exists in the pool.
*Interning in String Pool
Java provides the intern() method:
It checks whether the String exists in the pool.
If yes, it returns the pooled reference.
If not, it adds the String to the pool.

PART B PROGRAMMING QUESTIONS

Q1)Write a program to reverse an array.

ANS:-import java.util.Scanner;
class ReverseArray {
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
System.out.print("Enter array size: ");
int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter array elements:");
for (int i = 0; i < n; i++) {
arr[i] = sc.nextInt();
}

```
int start = 0;
int end = n - 1;

while (start < end) {
int temp = arr[start];
arr[start] = arr[end];
arr[end] = temp;

start++;
end--;
}
System.out.println("Reversed array:");
for (int i : arr) {
System.out.print(i + " ");
}
}
}
```

Q2)Write a program to find maximum element in an array.
Ans:-

```
import java.util.Scanner;
class MaxElementArray {
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
System.out.print("Enter array size: ");
int n = sc.nextInt();

int[] arr = new int[n];
System.out.println("Enter array elements:");
for (int i = 0; i < n; i++) {
arr[i] = sc.nextInt();
}
int max = arr[0];

for (int i = 1; i < n; i++) {
if (arr[i] > max) {
max = arr[i];
}
}
System.out.println("Maximum element in array: " + max);
}
}
```

Q3)Write a program to find minimum element in an array.
Ans:-

```
import java.util.Scanner;
class MinElementArray {
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
System.out.print("Enter array size: ");
int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter array elements:");
for (int i = 0; i < n; i++) {
```

```
arr[i] = sc.nextInt();
}

int min = arr[0];

for (int i = 1; i < n; i++) {
if (arr[i] < min) {
min = arr[i];
}
}
System.out.println("Minimum element in array: " + min);
}
}
```

Q4)Write a program to calculate sum of elements in an array.
Ans:-

```
import java.util.Scanner;
class SumOfArray {
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
System.out.print("Enter array size: ");
int n = sc.nextInt();

int[] arr = new int[n];
System.out.println("Enter array elements:");
for (int i = 0; i < n; i++) {
arr[i] = sc.nextInt();
}
int sum = 0;
for (int i = 0; i < n; i++) {
sum = sum + arr[i];
}
System.out.println("Sum of array elements: " + sum);
}
}
```

Q5)Write a program to find average of array elements.

Ans:-

```
import java.util.Scanner;
class AverageOfArray {
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
System.out.print("Enter array size: ");
int n = sc.nextInt();

int[] arr = new int[n];
System.out.println("Enter array elements:");
for (int i = 0; i < n; i++) {
arr[i] = sc.nextInt();
}
int sum = 0;
```

```
for (int i = 0; i < n; i++) {
sum += arr[i];
}
double average = (double) sum / n;
System.out.println("Average of array elements: " + average);
}
}
```