USN | | | | | | | | | | |

Internal Assessment Test 1 – March 2025

Sub	Advanced Java					Sub Code:	BCS613D	Branch:	CSE
Date:	05/03/2025	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VI Professional Elective		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	What is a collection framework? Explain the methods defined by the following interfaces. i. Collection ii. List ii. SortedSet iv. Queue	[10]	CO1	L2
2	Explain the following map class with a suitable program i. HashMap ii. TreeMap	[10]	CO1	L2
3 (a)	Explain the constructors of ArrayList class with an example program	[05]	CO1	L2
3 (b)	Explain constructor of TreeSet class. Write a Java program to create a TreeSet collection.	[05]	CO1	L2
4 (a)	What is String in Java? Explain the different constructors of String class.	[06]	CO2	L2
4 (b)	List the difference between String class and StringBuffer class.	[04]	CO2	L2
5 (a)	What is toString() function? Write a program to override toString() and provide your own string representations.	[05]	CO2	L3
5 (b)	Write a program to sort names using Bubble Sort.	[05]	CO2	L3
6 (a)	Explain indexOf() and lastIndexOf() methods of String class with an example.	[05]	CO2	L3
6 (b)	Write a program in Java to replace all matching substrings with the given string.	[05]	CO2	L3

1. What is a collection framework? Explain the methods defined by the following interfaces.

i. **Collection**
ii. **List**
iii. **Sorted Set**
iv. **Queue**

Ans: The **Java Collection Framework** helps you **store, manage, and work with groups of objects** like lists, sets, and queues.

It is like a **toolbox** for handling data in different ways (e.g., ordered, sorted, unique, etc.).

(a) Collection Interface:

This is the root interface of the collection hierarchy. It extends the Iterable interface, so it can be iterated using an enhanced for loop or iterators.

Methods:

- add(obj) – Adds an element
- clear() – Removes all elements
- isEmpty() – Checks if the collection is empty
- iterator() – Returns an iterator
- remove(obj) – Removes an element

(b) List Interface:

Extends the Collection interface to define ordered sequences (indexed access).

Methods:

- add(index, obj) – Adds element at specific index
- addAll(index, collection) – Adds collection at specific index
- get(index) – Gets element at index
- indexOf(obj) – Gets first index of element
- lastIndexOf(obj) – Gets last index of element
- remove(index) – Removes element at index
- set(index, obj) – Replaces element at index

(c) SortedSet Interface:

Extends Set to define a set that maintains elements in **ascending order**.

Methods:

- first() – Gets the first (smallest) element
- last() – Gets the last (largest) element
- headSet(end) – Gets all elements less than end
- subSet(start, end) – Gets elements between start and end

(d) Queue Interface:

Extends the Collection to define First-In-First-Out (FIFO) queues, though some queues may have different orderings.

Methods:

- element() – Gets head element (throws error if empty)
- offer(obj) – Adds element (returns false if full)
- peek() – Gets head element (returns null if empty)
- poll() – Removes and returns head (null if empty)
- remove() – Removes and returns head (error if empty)

2. Explain the following map class with a suitable program

i. Hash Map
ii. Tree Map

Ans:- (i) Hash Map:

- HashMap stores **key-value pairs**.
- It **does not maintain any order**.
- **Keys must be unique**, but **values can be duplicated**.
- Fast for searching and inserting.

Example:

```
import java.util.*;
class HashMapDemo {
   public static void main(String[] args) {
      HashMap<Integer, String> map = new HashMap<>();
      map.put(101, "Apple");
      map.put(102, "Banana");
      map.put(103, "Cherry");
      System.out.println("HashMap: " + map);
   }
}
```

Output:- HashMap: {101=Apple, 102=Banana, 103=Cherry}

(ii)Tree Map:

- TreeMap also stores **key-value pairs**, just like HashMap.
- BUT – it **stores keys in sorted (ascending) order**.
- Slower than HashMap, but gives **sorted output** by keys.

Example:-

```
import java.util.*;
class TreeMapDemo {
   public static void main(String[] args) {
      TreeMap<Integer, String> map = new TreeMap<>();
      map.put(103, "Cherry");
      map.put(101, "Apple");
      map.put(102, "Banana");
      System.out.println("TreeMap: " + map);
   }
}
```

Output:- TreeMap: {101=Apple, 102=Banana, 103=Cherry}

3(a) Explain the construction of ArrayList class with an example program

Ans:- ArrayList which add or remove elements.

Construction:

ArrayList<Type> listName = new ArrayList<>();

Examples:

ArrayList<String> names = new ArrayList<>(); // for Strings
ArrayList<Integer> numbers = new ArrayList<>(); // for Integers

Example Program:

```
import java.util.*;
class ArrayListDemo {
   public static void main(String[] args) {
      ArrayList<String> fruits = new ArrayList<>();
      fruits.add("Apple");
      fruits.add("Banana");
      fruits.add("Mango");
      fruits.add("Orange");
      System.out.println("Fruits List: " + fruits);
      System.out.println("First fruit: " + fruits.get(0)); // Apple
      fruits.remove("Banana");
      System.out.println("After removing Banana: " + fruits);
      fruits.set(1, "Grapes");
      System.out.println("Final List: " + fruits);
   }
}
```

Output:

```
Fruits List: [Apple, Banana, Mango, Orange]
First fruit: Apple
After removing Banana: [Apple, Mango, Orange]
Final List: [Apple, Grapes, Orange]
```

3.(b). Explain constructor of TreeSet class. Write a java program to create a TreeSet collection

Ans:- TreeSet is a class in Java used to **store elements in sorted (ascending) order**. It **does not allow duplicate elements**.

TreeSet Constructors:

Constructor	Description
TreeSet()	Creates an empty sorted set.
TreeSet(Collection c)	Creates a TreeSet from another collection.
TreeSet(Comparator comp)	Creates a TreeSet with custom sorting logic.

Java Program:-

```
import java.util.*;

class TreeSetExample {
   public static void main(String[] args) {
      TreeSet<String> animals = new TreeSet<>();
      animals.add("Dog");
      animals.add("Cat");
      animals.add("Elephant");
      animals.add("Bear");
      animals.add("Zebra");
      animals.add("Cat");
      System.out.println("Animals in TreeSet (Sorted): " + animals);
      System.out.println("First animal: " + animals.first());
      System.out.println("Last animal: " + animals.last());
   }
}
```

Output:-
Animals in TreeSet (Sorted): [Bear, Cat, Dog, Elephant, Zebra]
First animal: Bear
Last animal: Zebra

4.(a). What is String in java? Explain the different constructors of string class

Ans:-

- In Java, a **String** is a **sequence of characters**.
- Strings are **objects** of the class java.lang.String.
- String is **immutable**, which means **once created, it cannot be changed**.

Example:-
String str = "Hello, World!";

1. String()

- Creates an **empty string**.

```
String s1 = new String();
System.out.println("s1: " + s1);
```

2. String(String original)

- Creates a **copy of an existing string**.

```
String s2 = new String("Hello");
System.out.println(s2);
```

3. String(char[] charArray)

- Creates a string from a **character array**.

```
char[] letters = {'J', 'a', 'v', 'a'};
String s3 = new String(letters);
System.out.println(s3);
```

4. String(char[] charArray, int startIndex, int count)

- Creates a string using **a portion** of a character array.

```
char[] letters = {'H', 'e', 'l', 'l', 'o'};
String s4 = new String(letters, 1, 3);
System.out.println(s4);
```

5. String(byte[] byteArray)

- Creates a string from a **byte array** using platform's default charset.

```
byte[] byteArr = {72, 101, 108, 108, 111};
String s5 = new String(byteArr);
System.out.println(s5);
```

6. String(byte[] byteArray, int offset, int length)

- Creates a string from **part of a byte array**.

```
byte[] byteArr = {65, 66, 67, 68};
String s6 = new String(byteArr, 1, 2);
System.out.println(s6);
```

4.(b). List the difference between the String class and StringBuffer class

Ans:-

Difference between String and StringBuffer

Feature	String	StringBuffer
Mutability	**Immutable** – cannot be changed after creation	**Mutable** – content can be changed
Package	java.lang.String	java.lang.StringBuffer
Performance	Slower when modifying (because new object is created)	Faster for multiple modifications
Thread Safety	Not thread-safe	**Thread-safe** (methods are synchronized)
Usage	Use when strings don't change often	Use when you need to modify strings frequently
Memory	Creates new object on modification	Modifies existing object
Common Methods	length(), charAt(), substring(), etc.	append(), insert(), delete(), reverse()
Efficiency	Less efficient for concatenation in loops	More efficient for repeated modifications

Example:-

Using String:-

```
String s = "Hello";

s = s + " World";

System.out.println(s);
```

Using StringBuffer:-

```
StringBuffer sb = new StringBuffer("Hello");

sb.append(" World");

System.out.println(sb);
```

5. What is toString() function? Write a program to override toString() and provide your own string representations.

Ans:- toString() is a method defined in the **Object class** (which is the parent of all classes in Java).

- It returns a **string representation** of the object.

Java Program to Override toString():-

```
class Employee {
   int id;
   String name;
   String department;
   Employee(int id, String name, String department) {
      this.id = id;
      this.name = name;
      this.department = department;
   }
   public String toString() {
      return "Employee ID: " + id + ", Name: " + name + ", Department: " + department;
   }
}
```

```
public class Main {
   public static void main(String[] args) {
      Employee emp1 = new Employee(101, "John Doe", "IT");
      System.out.println(emp1);
   }
}
```

- **Output:**

```
Employee ID: 101, Name: John Doe, Department: IT
```

5.(b). Write a program to sort names using Bubble Sort

Ans:

```
import java.util.Scanner;
public class BubbleSortNames {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter the number of names: ");
      int n = sc.nextInt();
      sc.nextLine();
      String[] names = new String[n];
      System.out.println("Enter the names:");
      for (int i = 0; i < n; i++) {
         names[i] = sc.nextLine();
      }
      for (int i = 0; i < n - 1; i++) {
         for (int j = 0; j < n - 1 - i; j++) {
            if (names[j].compareToIgnoreCase(names[j + 1]) > 0) {
               String temp = names[j];
               names[j] = names[j + 1];
               names[j + 1] = temp;
            }
         }
      }
```

```
System.out.println("\nSorted names:");
      for (String name : names) {
         System.out.println(name);
      }
      sc.close();
   }
}
```

- **Output:-**

```
Enter the number of names: 5
Enter the names:
Zara
Amit
John
Bella
Chris

Sorted names:
Amit
Bella
Chris
John
Zara
```

6(a) Explain indexOf() and lastIndexOf() methods of String class with an example

Ans: indexOf() and lastIndexOf() Methods in Java

These methods belong to the **String class** and are used to **find the position (index) of characters or substrings** inside a string.

- **indexOf()**

- **Purpose:** Returns the **first occurrence** of a character or substring.
- **Syntax:**

int indexOf(char ch)

int indexOf(String str)

- **lastIndexOf()**

- **Purpose:** Returns the **last occurrence** of a character or substring.
- **Syntax:**

int lastIndexOf(char ch)

int lastIndexOf(String str)

- **Example Program:-**

```
public class IndexOfExample {
   public static void main(String[] args) {
      String str = "Java Programming Language";

      // indexOf()
      System.out.println("First index of 'a': " + str.indexOf('a')); // 1
      System.out.println("First index of 'gram': " + str.indexOf("gram")); // 10

      // lastIndexOf()
      System.out.println("Last index of 'a': " + str.lastIndexOf('a')); // 24
      System.out.println("Last index of 'Java': " + str.lastIndexOf("Java")); // 0
   }
}
```

- **Output:-**

First index of 'a': 1

First index of 'gram': 10

Last index of 'a': 24

Last index of 'Java': 0

6.(b). Write a program in java to replace all matching substrings with the given string.

Ans:-

Java Program to Replace All Matching Substrings:-

```
import java.util.Scanner;
public class ReplaceSubstring {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter the original string: ");
      String original = sc.nextLine();
      System.out.print("Enter the substring to replace: ");
      String toReplace = sc.nextLine();
      System.out.print("Enter the new string to replace with: ");
      String replacement = sc.nextLine();
      String result = original.replaceAll(toReplace, replacement);

      System.out.println("\nResult after replacement: ");
      System.out.println(result);

      sc.close();
   }
}
```

- **Output:-**

Enter the original string: Java is fun. Java is powerful.

Enter the substring to replace: Java

Enter the new string to replace with: Python

Result after replacement:

Python is fun. Python is powerful.