Experiment 5

Student Name: Japneet Kaur **UID: 22BCS10390**

Branch: CSE **Section:618(A)**

Semester: 6th **DOP:21/02/2025**

Subject: Java **Subject Code: 22CSH-359**

Problem Statement 1

1. **Aim:** Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes.

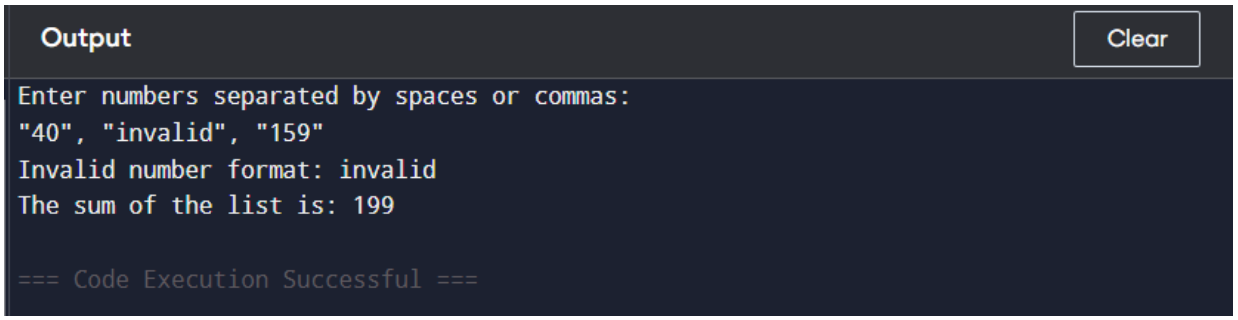
2. **Objective:** The objective of this experiment is to enhance understanding of Java's wrapper classes, I/O streams, and lambda expressions by implementing practical applications. This includes efficiently handling data conversions using autoboxing and unboxing, managing file operations through serialization and deserialization, and leveraging functional programming concepts to write concise and effective Java programs.

3. **Code:**

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class SumCalculator {
   // Method to parse a string into an Integer after removing extra quotes
   public static Integer parseStringToInteger(String str) {
      str = str.replaceAll("\"", "").trim();
      try {
         return Integer.parseInt(str);
      } catch (NumberFormatException e) {
         System.out.println("Invalid number format: " + str);
         return null; // Return null if invalid input
      }
   }
```

```
// Method to calculate sum
public static int calculateSum(List<Integer> numbers) {
    int sum = 0;
    for (Integer num : numbers) {
        if (num != null) {  // Ignore null values
            sum += num; }}
    return sum;
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    List<Integer> numbers = new ArrayList<>();
    System.out.println("Enter numbers separated by spaces or commas:");
    String input = scanner.nextLine();
    // Split input by comma and space, then trim each value
    String[] tokens = input.split("[,\\s]+");  // Handles spaces and commas
    // Convert each input to an integer
    for (String token : tokens) {
        numbers.add(parseStringToInteger(token));
    }
    // Calculate and display sum
    System.out.println("The sum of the list is: " + calculateSum(numbers));
    scanner.close(); }}
```

4. **Output:**



Problem Statement 2

1. **Aim:** Create a Java program to serialize and deserialize a Student object. The program should: Serialize a Student object (containing id, name, and GPA) and save it to a file. Deserialize the object from the file and display the student details. Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

2. **Code:**

```
import java.io.*;
import java.util.Scanner;

class Student implements Serializable {
   private static final long serialVersionUID = 1L;
   private int id;
   private String name;
   private double gpa;

   public Student(int id, String name, double gpa) {
      this.id = id;
      this.name = name;
      this.gpa = gpa;
   }
   public void displayStudent() {
      System.out.println("Student ID: " + id + ", Name: " + name + ", GPA: " + gpa);
   }
}
public class Main {
   private static final String FILE_NAME = "student.ser";
   public static void serializeStudent(Student student) {
      try(ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
         oos.writeObject(student);
         System.out.println("☑ Student object has been serialized and saved to file.");
      } catch (IOException e) {
         System.out.println("❌ Error: Unable to serialize student."); }}
   public static Student deserializeStudent() {
      try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
         return (Student) ois.readObject();
      } catch (FileNotFoundException e) {
         System.out.println("❌ Error: File not found.");
```

```
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("❌ Error: Unable to deserialize student.");
        }
        return null;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // User input for student details
        System.out.print("Enter Student ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Student Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Student GPA: ");
        double gpa = scanner.nextDouble();

        // Create Student object
        Student student = new Student(id, name, gpa);
        // Serialize the student object
        serializeStudent(student);
        // Deserialize the student object
        Student deserializedStudent = deserializeStudent();
        if (deserializedStudent != null) {
            System.out.println("\n☑ Student object has been deserialized.");
            deserializedStudent.displayStudent();
        }
        scanner.close();
    }
}
```

3. **Output:**

```
Enter Student ID: 101
Enter Student Name: JK
Enter Student GPA: 8.65
✅Student object has been serialized and saved to file.

✅Student object has been deserialized.
Student ID: 101, Name: JK, GPA: 8.65


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement 3

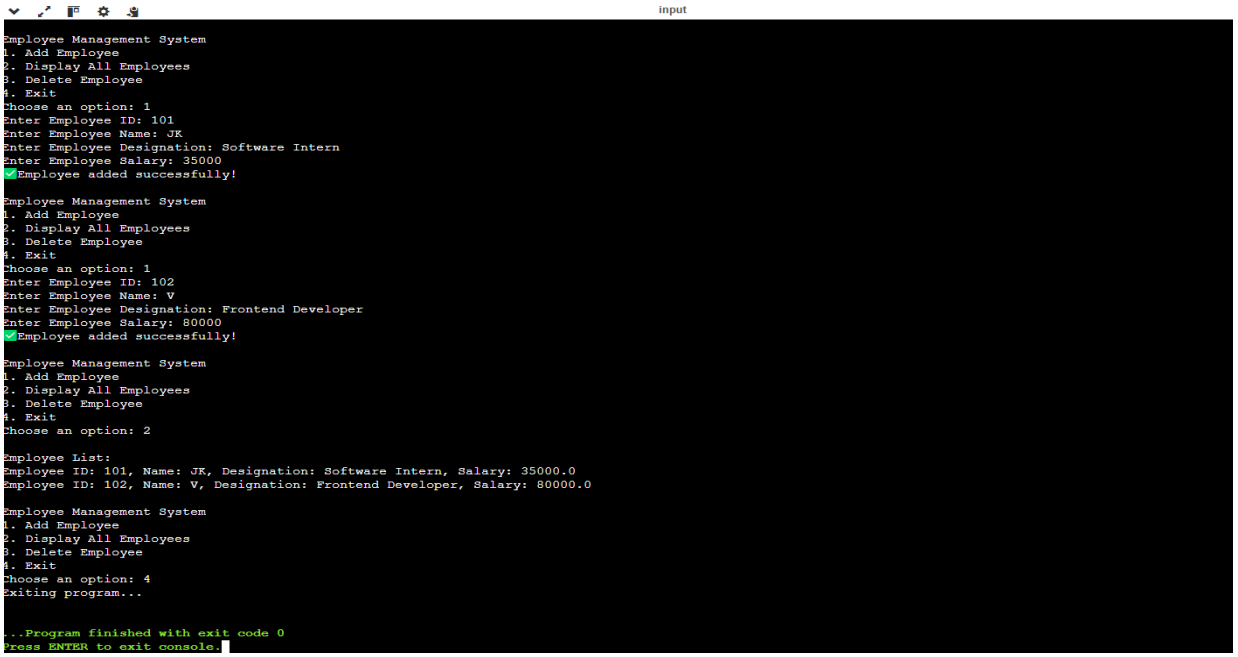
1. **Aim:** Create a menu-based Java application with the following options. 1.Add an Employee 2. Display All 3. Exit If option 1 is selected, the application should gather details of the employee like employee name, employee id, designation and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected the application should exit.

2. **Code:**

```
import java.io.*;
import java.util.*;

class Employee implements Serializable {
   private static final long serialVersionUID = 1L;
   int id; String name, designation; double salary;
   public Employee(int id, String name, String designation, double salary) {
      this.id = id; this.name = name; this.designation = designation; this.salary = salary;
   }
   public void display() {
      System.out.println("ID: " + id + ", Name: " + name + ", Designation: " + designation
+ ", Salary: " + salary);
   }
}
public class Main {
   private static final String FILE_NAME = "employees.ser";
   private static final Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
      while (true) {
System.out.println("\n1. Add Employee  2. Display Employees  3. Delete Employee  4. Exit");
         switch (scanner.nextInt()) {
            case 1 -> addEmployee();
            case 2 -> displayEmployees();
            case 3 -> deleteEmployee();
            case 4 -> { System.out.println("Exiting..."); System.exit(0); }
            default -> System.out.println("Invalid choice!");
         }
      }
   }
```

```
    private static void addEmployee() {
        System.out.print("Enter ID, Name, Designation, Salary: ");
        List<Employee> employees = readEmployees();
        employees.add(new Employee(scanner.nextInt(), scanner.next(), scanner.next(),
scanner.nextDouble()));

        saveEmployees(employees);
        System.out.println("☑ Employee added!");
    }

    private static void displayEmployees() {
        List<Employee> employees = readEmployees();
        if (employees.isEmpty()) System.out.println("❗ No employees found.");
        else employees.forEach(Employee::display); }
    private static void deleteEmployee() {
        System.out.print("Enter Employee ID to delete: ");

        List<Employee> employees = readEmployees();
        if (employees.removeIf(e -> e.id == scanner.nextInt())) {
            saveEmployees(employees);
            System.out.println("☑ Employee deleted!");
        } else System.out.println("❌ Employee not found.");
}

    private static void saveEmployees(List<Employee> employees) {
        try (var oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
        } catch (IOException e) {
            System.out.println("❌ Error saving employees.");
    }
}

    private static List<Employee> readEmployees() {
        try (var ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            return (List<Employee>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            return new ArrayList<>();
        }
    }
}
```

3. Output



4. Learning Outcomes

- Understand the concept of autoboxing and unboxing in Java and how primitive data types are automatically converted to their corresponding wrapper classes.
- Learn how to serialize and deserialize objects using Java's I/O streams, enabling efficient data storage and retrieval.
- Develop error-handling skills by managing exceptions like `FileNotFoundException`, `IOException`, and `ClassNotFoundException` during file operations.
- Gain experience in functional programming by implementing lambda expressions, method references, and functional interfaces for cleaner and more efficient code.