Experiment-5

Name-Mayank **UID-22BCS13254**

Branch-CSE **Section-639-A**

Semester-6th **Date of Performance-21-2-25**

Subject-PBLJ Lab **Subject Code-22CSH-359**

Que.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 (e.g., Integer.parseInt()).

2. Objective-To demonstrate autoboxing and unboxing in Java by computing the sum of a list of integers. The program will also include methods to parse string representations of numbers into their respective wrapper classes using Integer.parseInt().

3. Code-

```
import java.util.ArrayList;

import java.util.List;


public class AutoBoxingUnboxingSum {
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) { // Unboxing happens here
            sum += num;
        }
        return sum;
    }
     public static List<Integer> parseStringToIntegers(String[] strNumbers) {
        List<Integer> numbers = new ArrayList<>();
        for (String str : strNumbers) {
            numbers.add(Integer.parseInt(str)); // Autoboxing happens here
        }
        return numbers;
    }
     public static void main(String[] args) {
        String[] strNumbers = {"10", "20", "30", "40", "50"};
```

```
        List<Integer> numbers = parseStringToIntegers(strNumbers);
        int sum = calculateSum(numbers);
        System.out.println("Sum of numbers: " + sum);
    }
}
```

4.Output-

```
Sum of numbers: 150

=== Code Execution Successful ===
```

5. Learning Outcomes-

- **Understanding Autoboxing and Unboxing:** The code demonstrates how primitive data types (int) are automatically converted to wrapper classes (Integer) and vice versa.
- **Parsing Strings to Integers:** The program shows how to convert string representations of numbers into Integer objects using Integer.parseInt().
- **Using Lists with Wrapper Classes:** It illustrates how ArrayList<Integer> works with wrapper classes and benefits from autoboxing when adding elements.

Que.2

1. Aim-Medium Level: 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. Objective-To demonstrate Java object serialization and deserialization by saving and retrieving a Student object from a file. The program handles FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

3. Code-

```
import java.io.*;


// Serializable Student class
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 display() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("GPA: " + gpa);
    }
}

public class StudentSerialization {
    private static final String FILE_NAME = "student.ser";
```

```
    // Serialize the Student object
    public static void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME)))  {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (IOException e) {
            System.err.println("Error during serialization: " + e.getMessage());
        }
    }

    // Deserialize the Student object
    public static Student deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            return (Student) ois.readObject();
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Error during deserialization: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Class not found: " + e.getMessage());
        }
        return null;
    }

    public static void main(String[] args) {
        Student student = new Student(101, "John Doe", 3.8);

        // Serialize
        serializeStudent(student);

        // Deserialize
        Student deserializedStudent = deserializeStudent();
```

```
        if (deserializedStudent != null) {
            System.out.println("Deserialized Student Details:");
            deserializedStudent.display();
        }
    }
}
```

4.Ouput-

```
Student object serialized successfully.
Deserialized Student Details:
ID: 101
Name: John Doe
GPA: 3.8
```

5. Learning Outcomes-

- **Understanding Serialization & Deserialization:** Demonstrates how to convert a Java object into a byte stream and restore it from a file.
- **Exception Handling in File Operations:** Shows how to handle FileNotFoundException, IOException, and ClassNotFoundException during serialization and deserialization.
- **Usage of Serializable Interface:** Explains the importance of implementing Serializable and using serialVersionUID for class versioning.
- **Using ObjectOutputStream and ObjectInputStream:** Illustrates working with Java's built-in classes for object-based file handling.

Que.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. Objective-To develop a menu-driven Java application that allows users to add employee details (name, ID, designation, salary) to a file and display all stored employee records using file handling and serialization techniques.

3. Code-

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


// Employee class implementing Serializable
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private 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 EmployeeManagement {
    private static final String FILE_NAME = "employees.ser";
    private static Scanner scanner = new Scanner(System.in);

    // Method to add an employee
    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();  // Consume newline
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();

        Employee employee = new Employee(id, name, designation, salary);
        List<Employee> employees = readEmployees();
        employees.add(employee);
        writeEmployees(employees);
        System.out.println("Employee added successfully!\n");
    }

    // Method to display all employees
    public static void displayAllEmployees() {
        List<Employee> employees = readEmployees();
        if (employees.isEmpty()) {
            System.out.println("No employees found.\n");
        } else {
            System.out.println("\nEmployee List:");
           for (Employee emp : employees) {
```

```
            emp.display();
        }
        System.out.println();
    }
}

// Method to read employees from file
private static List<Employee> readEmployees() {
    List<Employee> employees = new ArrayList<>();
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        employees = (List<Employee>) ois.readObject();
    } catch (FileNotFoundException e) {
        // File not found initially, ignore
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Error reading employees: " + e.getMessage());
    }
    return employees;
}

// Method to write employees to file
private static void writeEmployees(List<Employee> employees) {
    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME)))  {
        oos.writeObject(employees);
    } catch (IOException e) {
        System.err.println("Error writing employees: " + e.getMessage());
    }
}

public static void main(String[] args) {
    while (true) {
        System.out.println("Menu:");
        System.out.println("1. Add Employee");
```

```
        System.out.println("2. Display All Employees");
        System.out.println("3. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayAllEmployees();
                break;
            case 3:
                System.out.println("Exiting program...");
                System.exit(0);
            default:
                System.out.println("Invalid choice! Please try again.\n");
        }
    }
  }
}
```

4.Output-

```
Menu:
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 101
Enter Employee Name: John Doe
Enter Employee Designation: Software Engineer
Enter Employee Salary: 75000
Employee added successfully!
```

```
Menu:
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 2

Employee List:
ID: 101, Name: John Doe, Designation: Software Engineer, Salary: 75000.0
```

```
Menu:
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 3
Exiting program...
```

5.Learning Outcomes-

- **File Handling in Java:** Demonstrates reading and writing serialized objects to a file using ObjectInputStream and ObjectOutputStream.
- **Menu-Driven Application Development:** Implements a user-interactive console-based application using a while loop and switch statement.
- **Exception Handling in File Operations:** Shows handling of FileNotFoundException, IOException, and ClassNotFoundException when dealing with file input/output.
- **Serialization and Deserialization:** Illustrates how objects can be stored persistently and retrieved using Java's Serializable interface.