Experiment 5

Student Name: Aditi Sharma **UID**: 22BCS10692

Branch: BE-CSE **Section/Group**: IOT-641/B

Semester: 6th **Date of Performance**: 5/03/2025

Subject Name: Project Based Learning Subject in Java with Lab **Code**: 22CSH-359

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:**

   a) Understand autoboxing and unboxing in Java.
   b) Utilize wrapper class methods like `Integer.parseInt()` to convert string values into integers.

3. **Algorithm**:

   a) Create a list of integers.
   b) Convert string inputs into integers using `Integer.parseInt()`.
   c) Use autoboxing to add integer values to the list.
   d) Use unboxing to compute the sum of integers.
   e) Display the result.

4. **Implementation/Code**:

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

public class Main {
    public static void main(String[] args) {
        String[] numberStrings = {"5", "15", "25", "35", "45"};  // Updated values

        List<Integer> numbers = parseStringArrayToIntegers(numberStrings);
```

```
        int sum = calculateSum(numbers);

        System.out.println("The sum of the numbers is: " + sum);
    }

    public static List<Integer> parseStringArrayToIntegers(String[] strings) {
        List<Integer> integerList = new ArrayList<>();

        for (String str : strings) {
            integerList.add(Integer.parseInt(str));
        }

        return integerList;
    }

    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;

        for (Integer num : numbers) {
            sum += num;
        }

        return sum;
    }
}
```

4. Output:

```
The sum of the numbers is: 125


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

5. Learning Outcomes:
   a) Understanding autoboxing and unboxing in Java.
   b) Using wrapper class methods like `Integer.parseInt()`.
   c) Handling lists of numeric values efficiently.

MEDIUM:

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. Objective:

a) Learn Java serialization and deserialization.
b) Understand how to save and retrieve objects from files.
c) Implement exception handling for file operations.

3. Algorithm:

1. Create a `Student` class implementing `Serializable`.
2. Define attributes like `id`, `name`, and `GPA`.
3. Write a method to serialize the object to a file.
4. Write a method to deserialize the object from a file.
5. Handle `FileNotFoundException`, `IOException`, and `ClassNotFoundException`.

4. **Implementation/Code**:

```
import java.io.*;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name;
    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("Student ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("GPA: " + gpa);
    }
}

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

    public static void serializeStudent(Student student) {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            out.writeObject(student);
            System.out.println("Student object serialized.");
        } catch (IOException e) {
            System.out.println("IOException occurred: " + e.getMessage());
        }
    }
```

```
    public static void deserializeStudent() {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            Student student = (Student) in.readObject();
            System.out.println("Deserialized Student:");
            student.display();
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error during deserialization: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        Student student = new Student(101, "Aditi Sharma", 8.5);
        serializeStudent(student);
        deserializeStudent();
    }
}
```

Output:

```
        System.out.println("Student ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("GPA: " + gpa);
    }
}

public class Main {
    public static void main(String[] args) {
        Student student = new Student(101, "Aditi Sharma", 7.5);

        // Serialize the student object
```

input

```
Student object serialized successfully!

Deserialized Student Details:
Student ID: 101
Name: Aditi Sharma
GPA: 7.5


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

HARD:

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:

a) Learn file handling in Java.
b) Implement a menu-based system with user inputs.
c) Store and retrieve structured data.

3. Algorithm:

1. Display menu options:
2. **Option 1:** Add an employee.
3. **Option 2:** Display all employees.
4. **Option 3:** Exit.
5. If the user selects **Option 1**, prompt for employee details and save them to a file.
6. If the user selects **Option 2**, read and display all employee records from the file.
7. If the user selects **Option 3**, exit the program.
8. Handle exceptions related to file operations.

4. **Implementation/Code**:

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

// Employee class to store details
class Employee {
    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;
    }

    // Method to display employee details
    public void display() {
        System.out.println("\nEmployee ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Designation: " + designation);
        System.out.println("Salary: " + salary);
    }
}

public class Main {
    private static final List<Employee> employees = new ArrayList<>();
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Add an Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayEmployees();
                    break;
                case 3:
```

```
                System.out.println("Exiting program. Goodbye!");
                return;
            default:
                System.out.println("Invalid choice! Please enter 1, 2, or 3.");
        }
    }
}

// Method to add an employee
private static void addEmployee() {
    System.out.print("\nEnter Employee ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    System.out.print("Enter Employee Name: ");
    String name = scanner.nextLine();

    System.out.print("Enter Designation: ");
    String designation = scanner.nextLine();

    System.out.print("Enter Salary: ");
    double salary = scanner.nextDouble();

    employees.add(new Employee(id, name, designation, salary));
    System.out.println("Employee added successfully!");
}

// Method to display all employees
private static void displayEmployees() {
    if (employees.isEmpty()) {
        System.out.println("\nNo employees found!");
    } else {
        System.out.println("\nEmployee Details:");
        for (Employee emp : employees) {
            emp.display();
        }
    }
}
}
```

Output:

```
input
Enter your choice: 1

Enter Employee ID: 10692
Enter Employee Name: Aditi Sharma
Enter Designation: engineer
Enter Salary: 150000
Employee added successfully!

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

Employee Details:

Employee ID: 10692
Name: Aditi Sharma
Designation: engineer
Salary: 150000.0

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


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

Learning Outcomes

a) Understanding file handling and object serialization.
b) Implementing a menu-driven program.
c) Managing structured data in a persistent way.