Experiment 5

Student Name: Praburam M UID: 22BCS16537
Branch: BE-CSE Section/Group: DL_903_A
Semester: 6th Date of Performance: 18-02-2025
Subject Name: Program Based Learning in Java with Lab Subject Code: 22CSH-359

1. Aim: Solving problems under the category of autoboxing, serialization, file handling, and efficient data processing and management in Easy, Medium and Hard

2. Objective: Introduction to Wrapper class. Understanding the concept of serialization, file handling, and efficient data processing and management in Java.

3. Implementation/Code:

   1. Easy Level: 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()).

   Code:

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

public class SumCalculator {

    // Method to parse a string into an Integer
    public static Integer parseStringToInteger(String number) {
        return Integer.parseInt(number);
    }

    // Method to calculate the sum using autoboxing and unboxing
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer number : numbers) {
            sum += number; // Unboxing happens here automatically
        }
        return sum;
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Integer> numbers = new ArrayList<>();

        System.out.println("Enter integers (type 'done' to finish):");
        while (true) {
            String input = scanner.nextLine();
            if (input.equalsIgnoreCase("done")) {
                break;
            }
            try {
                Integer number = parseStringToInteger(input); // Autoboxing
                numbers.add(number);
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid integer.");
            }
        }

        int sum = calculateSum(numbers);
        System.out.println("The sum of the entered integers is: " + sum);
    }
}
```

2. 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.

   Code:

```
import java.io.*;

// Student class implementing Serializable interface
class Student implements Serializable {
    private static final long serialVersionUID = 1L; // Ensures version compatibility during deserialization
    private int id;
    private String name;
    private double gpa;

    // Constructor
    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }
```

```
    // Method to display student details
    public void displayDetails() {
        System.out.println("Student ID: " + id);
        System.out.println("Student Name: " + name);
        System.out.println("Student GPA: " + gpa);
    }
}

public class StudentSerialization {

    // Method to serialize a Student object to a file
    public static void serializeStudent(Student student, String filename) {
        try (FileOutputStream fileOut = new FileOutputStream(filename);
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            out.writeObject(student);
            System.out.println("Student object has been serialized to " + filename);
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("I/O error during serialization: " + e.getMessage());
        }
    }

    // Method to deserialize a Student object from a file
    public static Student deserializeStudent(String filename) {
        Student student = null;
        try (FileInputStream fileIn = new FileInputStream(filename);
             ObjectInputStream in = new ObjectInputStream(fileIn)) {
            student = (Student) in.readObject();
            System.out.println("Student object has been deserialized from " + filename);
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("I/O error during deserialization: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println("Class not found: " + e.getMessage());
        }
        return student;
    }

    public static void main(String[] args) {
        String filename = "student.ser";

        // Creating a Student object
        Student student = new Student(101, "Alice Johnson", 3.8);

        // Serialize the object
        serializeStudent(student, filename);
```

```
        // Deserialize the object
        Student deserializedStudent = deserializeStudent(filename);
        if (deserializedStudent != null) {
            deserializedStudent.displayDetails();
        }
    }
}
```

3. Hard Level: 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.

   Code:

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

// Employee class representing the employee details
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

    // Constructor
    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 displayDetails() {
        System.out.println("Employee ID: " + id);
        System.out.println("Employee Name: " + name);
        System.out.println("Designation: " + designation);
```

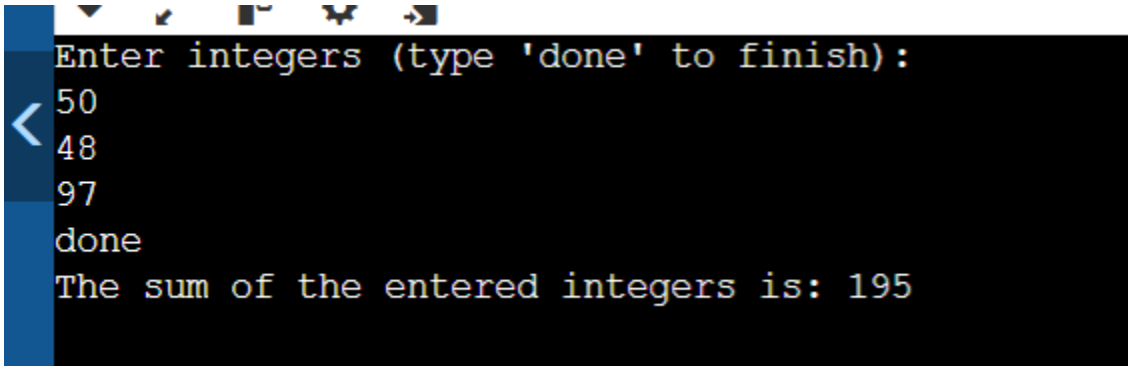
```
            System.out.println("Salary: $" + salary);
            System.out.println("--------------------------");
        }
    }

    public class EmployeeManagementSystem {

        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() {
            try {
                System.out.print("Enter Employee ID: ");
                int id = Integer.parseInt(scanner.nextLine());

                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 = Double.parseDouble(scanner.nextLine());

                Employee employee = new Employee(id, name, designation, salary);

                // Append the employee object to the file
                File file = new File(FILE_NAME);
                FileOutputStream fos = new FileOutputStream(file, true);
                ObjectOutputStream oos = file.length() == 0 ? new ObjectOutputStream(fos) : new
    AppendableObjectOutputStream(fos);

                oos.writeObject(employee);
                oos.close();
                System.out.println("Employee added successfully!\n");

            } catch (IOException | NumberFormatException e) {
                System.out.println("Error adding employee: " + e.getMessage());
```

```
    }
}

// Method to display all employees
public static void displayAllEmployees() {
    try (FileInputStream fis = new FileInputStream(FILE_NAME);
         ObjectInputStream ois = new ObjectInputStream(fis)) {

        System.out.println("\n--- Employee List ---");
        while (true) {
            try {
                Employee employee = (Employee) ois.readObject();
                employee.displayDetails();
            } catch (EOFException e) {
                break; // End of file reached
            }
        }

    } catch (FileNotFoundException e) {
        System.out.println("No employee records found. Please add employees first.");
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error reading employee data: " + e.getMessage());
    }
}

// Method to display the menu
public static void displayMenu() {
    while (true) {
        System.out.println("===== Employee Management System =====");
        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: ");

        String choice = scanner.nextLine();

        switch (choice) {
            case "1":
                addEmployee();
```

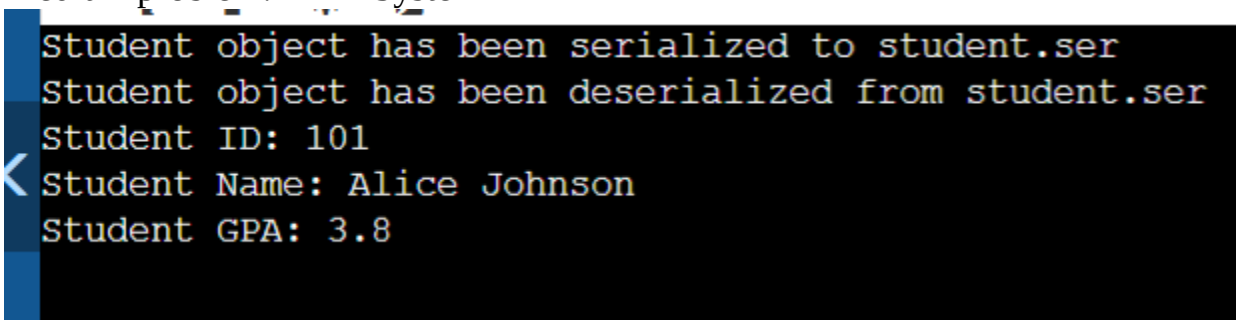
```
                    break;
                case "2":
                    displayAllEmployees();
                    break;
                case "3":
                    System.out.println("Exiting the application. Goodbye!");
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice. Please select 1, 2, or 3.");
            }
        }
    }

    public static void main(String[] args) {
        displayMenu();
    }
}

// Custom ObjectOutputStream to append objects without overwriting
class AppendableObjectOutputStream extends ObjectOutputStream {
    public AppendableObjectOutputStream(OutputStream out) throws IOException {
        super(out);
    }

    @Override
    protected void writeStreamHeader() throws IOException {
        reset(); // Prevent writing a new header
    }
}
```

5. Output

1.) Easy problem: Sum of entered integer



2.) Medium problem: ATM System



3.) Hard problem:

```
===== Employee Management System =====
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 16537
Enter Employee Name: Praburam
Enter Designation: SDE
Enter Salary: 150000
Employee added successfully!

===== Employee Management System =====
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2

--- Employee List ---
Employee ID: 16537
Employee Name: Praburam
Designation: SDE
Salary: $150000.0
--------------------------
===== Employee Management System =====
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 3
Exiting the application. Goodbye!
```

6. Learning Outcomes

1. Learned to use classes and objects for organizing employee and designation data in Java.
2. Implemented salary calculations using switch-case and array data handling.
3. Practiced input handling with the Scanner class and validating user input.
4. Gained experience in searching arrays and structuring conditional logic.
5. Displayed formatted output for real-world applications like employee management systems.