Experiment 5

Student Name: Nisha Kumari **UID: 22BET10118**

Branch: IT **Section/Group: 22BET_IOT-701/A**

Semester: 6th **Date of Performance:18.02.25**

Subject Name: PBLJ Lab **Subject Code: 22ITH-359**

Problem 1

1. **Aim:** To develop a program that demonstrates the use of wrapper classes with a focus on autoboxing and unboxing by calculating the sum of a list of integers.

2. **Objective:**
   - To demonstrate the use of wrapper classes like Integer in Java.
   - To implement autoboxing and unboxing for seamless conversion between primitive types and their wrapper objects.
   - To parse string inputs into integers using Integer.parseInt() for accurate data processing.
   - To calculate the sum of a list of integers efficiently.

3. **Code:**

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class SumUsingAutoboxing {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // List to store integers (autoboxing happens here)
        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 {
                int number = Integer.parseInt(input); // Parsing string to integer
                numbers.add(number); // Autoboxing: primitive int -> Integer object
```

```
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter an integer or 'done' to finish.");
            }
        }
        // Calculate sum (Unboxing happens automatically)
        int sum = calculateSum(numbers);
        System.out.println("The sum of the entered integers is: " + sum);
    }
    // Method to calculate the sum of integers (Unboxing happens here)
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer number : numbers) {
            sum += number; // Unboxing: Integer object -> primitive int
        }
        return sum;
    }
}
```

4. Output:

```
Enter integers (type 'done' to finish):
20
13
22
31
13
20
DONE
The sum of the entered integers is: 119
```

Fig 1. SUM OF INTEGER

5. Learning Outcomes:

- Understand the concept and usage of wrapper classes in Java.
- Implement autoboxing and unboxing for automatic type conversion between primitives and objects.
- Parse string inputs into numeric values using methods like Integer.parseInt().
- Develop skills in handling user input and managing exceptions effectively.
- Perform basic data processing tasks, such as calculating the sum of integers from user input.

Problem 2

1. Aim:

To develop a Java program that demonstrates **serialization** and **deserialization** of a Student object by saving its data (ID, name, and GPA) to a file and retrieving it later.

2. Objective:

- To implement serialization of a Student object and save it to a file.
- To perform deserialization and retrieve the object's data from the file.
- To handle file-related exceptions using proper exception handling mechanisms.
- To demonstrate persistent storage and retrieval of object data in Java.

3. Code:

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

// Student class implementing Serializable interface
class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    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("\n--- Student Details ---");
        System.out.println("Student ID  : " + id);
        System.out.println("Student Name: " + name);
        System.out.println("Student GPA : " + gpa);
    }
```

```
}

// Manages serialization and deserialization of Student objects
class StudentFileManager {
    private static final String FILE_NAME = "student.ser";

    // Serialize student object
    public static void serializeStudent(Student student) {
        try (FileOutputStream fileOut = new FileOutputStream(FILE_NAME);
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            out.writeObject(student);
            System.out.println("\n☑ Student object serialized successfully and saved to '" +
FILE_NAME + "'.");
        } catch (IOException e) {
            System.out.println("Error during serialization: " + e.getMessage());
        }
    }
    // Deserialize student object
    public static Student deserializeStudent() {
        File file = new File(FILE_NAME);
        if (!file.exists()) {
            System.out.println(" No serialized student data found.");
            return null;
        }

        try (FileInputStream fileIn = new FileInputStream(FILE_NAME);
             ObjectInputStream in = new ObjectInputStream(fileIn)) {
            return (Student) in.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(" Error during deserialization: " + e.getMessage());
        }
        return null;
    }
}
// Main class to manage user interactions
public class StudentSerialization {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
boolean exit = false;
while (!exit) {
    // Display menu
    System.out.println("\n===== Student Serialization Program =====");
    System.out.println("1. Create and Serialize a Student");
    System.out.println("2. Deserialize and Display Student Details");
    System.out.println("3. Exit");
    System.out.print("Enter your choice (1-3): ");

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

    switch (choice) {
        case 1:
            // Input 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 and serialize student
            Student student = new Student(id, name, gpa);
            StudentFileManager.serializeStudent(student);
            break;

        case 2:
            // Deserialize and display student details
            Student deserializedStudent = StudentFileManager.deserializeStudent();
            if (deserializedStudent != null) {
                deserializedStudent.displayDetails();
            }
            break;

        case 3:
            // Exit the program
```

```
                System.out.println("Exiting the program. Goodbye!");
                exit = true;
                break;

            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }

    scanner.close();
  }
}
```

4. Output:

```
===== Student Serialization Program =====
1. Create and Serialize a Student
2. Deserialize and Display Student Details
3. Exit
Enter your choice (1-3): 1
Enter Student ID: 1
Enter Student Name: Nisha
Enter Student GPA: 9.2

☑ Student object serialized successfully and saved to 'student.ser'.

===== Student Serialization Program =====
1. Create and Serialize a Student
2. Deserialize and Display Student Details
3. Exit
Enter your choice (1-3): 2

--- Student Details ---
Student ID  : 1
Student Name: Nisha
Student GPA : 9.2

===== Student Serialization Program =====
1. Create and Serialize a Student
2. Deserialize and Display Student Details
3. Exit
Enter your choice (1-3):
```

Fig 2.1. Student details

5. Learning Outcomes:

- Understanding the concepts of serialization and deserialization in Java.
- Learning to save and retrieve object data from files using streams.
- Implement exception handling for common file I/O errors.
- Developing the ability to manage persistent object data efficiently.
- Gain hands-on experience with Java's ObjectOutputStream and ObjectInputStream classes.

Problem 3

1. Aim:

To develop a menu-based Java application for managing employee records, including adding new employees, displaying all employee details, and storing data persistently using file handling.

2. Objective:

- To gather and store employee details (name, ID, designation, salary) in a file using file handling.
- To display all stored employee records from the file.
- To implement exception handling for robust file operations.
- To develop a user-friendly and efficient console-based interface for employee management..

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; // Ensures compatibility during serialization
    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 displayEmployee() {
        System.out.println("\n-------------------------");
        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("-------------------------");
    }
}
// File manager class to handle employee data storage and retrieval
class EmployeeFileManager {
    private static final String FILE_NAME = "employees.dat";
    // Save employee data to file (Serialization)
    public static void saveEmployees(List<Employee> employees) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
            System.out.println("\n☑ Employee data saved successfully!");
        } catch (IOException e) {
            System.out.println("Error saving employee data: " + e.getMessage());
        }
    }
    // Load employee data from file (Deserialization)
    @SuppressWarnings("unchecked")
    public static List<Employee> loadEmployees() {
        File file = new File(FILE_NAME);
        if (!file.exists()) {
            return new ArrayList<>(); // Return empty list if no file exists
        }

        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            return (List<Employee>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(" Error loading employee data: " + e.getMessage());
            return new ArrayList<>();
        }
    }
}

public class EmployeeManagementSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        List<Employee> employees = EmployeeFileManager.loadEmployees(); // Load employees
at startup
        boolean exit = false;

        while (!exit) {
            // Display menu
            System.out.println("\n===== 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 (1-3): ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline character
            switch (choice) {
                case 1:
                    // Add employee details
                    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();

                    Employee emp = new Employee(id, name, designation, salary); // Create employee object
                    employees.add(emp); // add it to the list
                    EmployeeFileManager.saveEmployees(employees); // Save updated employee list
                    break;

                case 2:
                    if (employees.isEmpty()) {                  // Display all employees
                        System.out.println("\nNo employee records found!");
                    } else {
                        System.out.println("\n===== Employee Records =====");
                        for (Employee e : employees) {
```

```
                    e.displayEmployee();
                }
            }
            break;
        case 3:
            // Exit program
            System.out.println("\nExiting the application. Goodbye!");
            exit = true;
            break;
        default:
            System.out.println("Invalid choice. Please select a valid option.");
      }
    }
    scanner.close();
  }
}
```

4. Output:

```
<terminated> EmployeeManagementSystem [Java Application] C:\Users\ASUS\Oracle_JDK-22\bin\javaw.exe  (24 Feb 2025, 8:13:36 pm – 8:14:13 pm elapsed: 0:00:36.530) [pid: 6128]

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

Enter Employee ID: 1
Enter Employee Name: Nisha
Enter Designation: Engineer
Enter Salary: 60000

☑ Employee data saved successfully!

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

===== Employee Records =====

--------------------------
Employee ID    : 1
Employee Name  : Nisha
Designation    : Engineer
Salary         : $60000.0
--------------------------

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

Exiting the application. Goodbye!
```

Fig 3.1. Employee Management System

5. Learning Outcomes:

- Implement file handling operations (read/write) in Java.
- Store and retrieve employee data from a file using object serialization.
- Develop a menu-driven application for managing employee data.
- Apply exception handling techniques to manage runtime errors effectively.
- Develop modular Java code using classes and methods for better maintainability.