Experiment 5

Student Name: Akshita Sharma **UID: 22BCS15804**
Branch: BE/CSE **Section/Group: IOT_618/B**
Semester: 6th **Date of Performance: 28/02/25**
Subject Name: Project based learning in Java
Subject Code: 22CSH- 359

Easy Level

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:** This program demonstrates autoboxing (automatic conversion of primitives to wrapper classes) and unboxing (conversion of wrapper objects back to primitives). It takes a list of integers as strings, converts them into Integer objects using Integer.parseInt(), and computes their sum.

3. **Implementation/Code:**

```
import java.util.ArrayList;
import java.util.List;
public class WrapperClassExample {
    public static void main(String[] args) {
        // Sample string numbers
        String[] numberStrings = {"10", "20", "30", "40", "50"};
```

```
        // Convert string numbers to Integer objects using parseInt
        List<Integer> numbers = new ArrayList<>();
        for (String numStr : numberStrings) {
            numbers.add(Integer.parseInt(numStr)); // Autoboxing
        }
        // Calculate the sum of the numbers using unboxing
        int sum = calculateSum(numbers);
        // Display the result
        System.out.println("Sum of the numbers: " + sum);
    }
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Unboxing happens here
        }
        return sum;
    } }
```

4. Output:

```
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> cd
assExample }
Sum of the numbers: 150
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6>
```

5. Learning Outcomes:

- Learnt about Autoboxing and Unboxing.
- Efficient Data Handling
- Learn to handle user input from the command line.
- Looping and Computation.
- Understanding Java Wrapper Class.

Medium Level

1. Aim: Create a Java program to serialize and deserialize a Student object.

2. Objective: This program serializes a Student object (converts it into a byte stream and writes to a file) and deserializes it (retrieves the object from the file).

3. Implementation/Code:

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
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 Details:");
```

```
        System.out.println("ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}
public class WrapperClassExampe{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Getting numbers from the user
        System.out.println("Enter 5 numbers (space-separated):");
        String[] numberStrings = scanner.nextLine().split(" ");
        List<Integer> numbers = new ArrayList<>();
        for (String numStr : numberStrings) {
            numbers.add(Integer.parseInt(numStr));
        }
        int sum = calculateSum(numbers);
        System.out.println("Sum of the numbers: " + sum);
        // Getting student details from the user
        System.out.println("Enter Student ID:");
        int id = scanner.nextInt();
        scanner.nextLine();  // Consume newline
        System.out.println("Enter Student Name:");
        String name = scanner.nextLine();
        System.out.println("Enter Student GPA:");
        double gpa = scanner.nextDouble();

        Student student = new Student(id, name, gpa);
```

```
        String filename = "student.ser";

        // Serialization
            try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filename))) {
            out.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (IOException e) {
            System.err.println("Error during serialization: " + e.getMessage());
        }

        // Deserialization
            try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(filename))) {
            Student deserializedStudent = (Student) in.readObject();
            System.out.println("Deserialized Student details:");
            deserializedStudent.displayStudent();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error during deserialization: " + e.getMessage());
        }
        scanner.close();
    }

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

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

4. Output:

```
Enter 5 numbers (space-separated):
10 20 30 40 50
Sum of the numbers: 150
Enter Student ID:
15804
Enter Student Name:
Akshita Sharma
Enter Student GPA:
8.5
Student object serialized successfully.
Deserialized Student details:
Student Details:
ID: 15804, Name: Akshita Sharma, GPA: 8.5
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6>
```

5. Learning Outcomes:

- Understanding Java collections.
- Implement key-value storage.
- Add and retrieve elements dynamically without predefined limits.
- Use Scanner to take user input and process it efficiently.

Hard Level

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:** The objective of this Java program is to create a menu-based Java application.

3. **Implementation/Code:**

```
import java.io.*;
import java.util.*;
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    int id;
    String name, designation;
    double salary;
    Employee(int id, String name, String designation, double salary) {
        this.id = id;
        this.name = name;
```

```
        this.designation = designation;
        this.salary = salary;
    }
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Designation: " + designation +
", Salary: " + salary;
    }
}
public class EmployeeManage1 {
    static final String FILE = "employees.ser";
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("\nChoose an option:");
            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 = sc.nextInt();
```

```
            switch (choice) {
                case 1 -> addEmployee(sc);
                case 2 -> displayEmployees();
                case 3 -> {
                    System.out.println("Exiting program. Goodbye!");
                    sc.close();
                    return;
                }
                  default -> System.out.println("Invalid choice. Please enter a number
between 1 and 3.");
            }
        }
    }
    static void addEmployee(Scanner sc) {
        try {
            System.out.print("Enter Employee ID: ");
            int id = sc.nextInt();
            sc.nextLine(); // Consume newline
```

```
            System.out.print("Enter Employee Name: ");
            String name = sc.nextLine();
            System.out.print("Enter Employee Designation: ");
            String des = sc.nextLine();
            System.out.print("Enter Employee Salary: ");
            double salary = sc.nextDouble();
            List<Employee> list = loadEmployees();
            list.add(new Employee(id, name, des, salary));
                try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE))) {
                oos.writeObject(list);
            }
            System.out.println("Employee added successfully!");
        } catch (Exception e) {
            System.out.println("Error adding employee: " + e.getMessage());
        }
    }
    static void displayEmployees() {
        List<Employee> employees = loadEmployees();
```

```
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            System.out.println("\nEmployee List:");
            for (Employee e : employees) {
                System.out.println(e);
            }
        }
    }
    static List<Employee> loadEmployees() {
            try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE))) {
            return (List<Employee>) ois.readObject();
        } catch (Exception e) {
            return new ArrayList<>();
        }
    }
}
```

4. Output:

```
Choose an option:
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 15804
Enter Employee Name: Akshita Sharma
Enter Employee Designation: General Manager
Enter Employee Salary: 150000
Employee added successfully!
```

```
ID: 15804, Name: Akshita Sharma, Designation: General Manager, Salary: 150000.0

Choose an option:
```

5. Learning Outcomes:

- How multiple threads interact when accessing shared data. Implement **encapsulation** by using private fields and constructors in the Employee class.
- Handling race conditions in a multi-threaded environment.
- Ensuring that only one thread modifies the shared resource (salary) at a time.
- Taking user input for dynamic seat selection and priority assignment.