Experiment 5

Student Name: Harshit Mishra **UID: 22BCS11592**

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

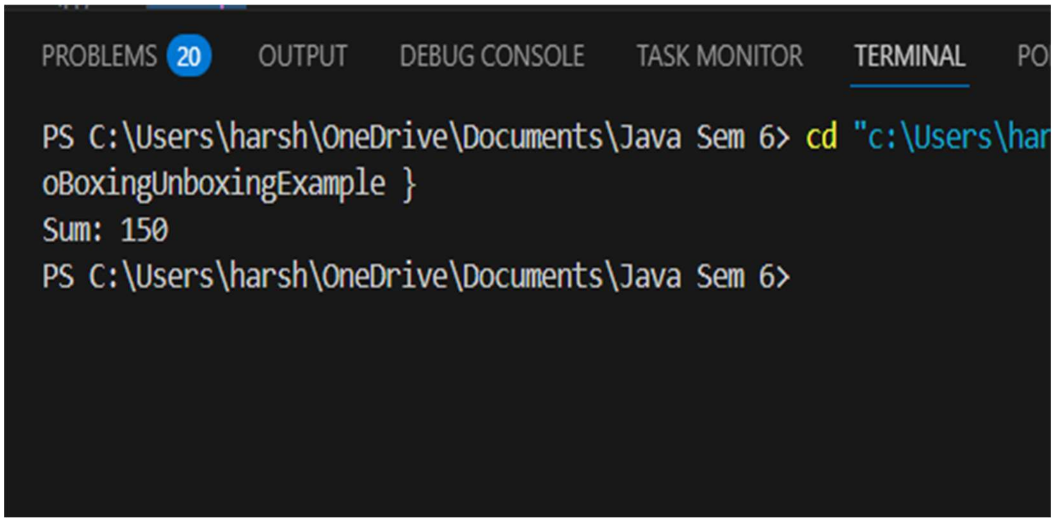
1. **Aim:** Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

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 AutoBoxingUnboxingExample {
    public static int calculateSum(List<String> numbers) {
        List<Integer> intList = new ArrayList<>();
        for (String num : numbers) {
            intList.add(Integer.parseInt(num)); // Autoboxing
        }
        int sum = 0;
        for (Integer num : intList) {
            sum += num; // Unboxing
        }
        return sum;
    }
    public static void main(String[] args) {
        List<String> numbers = List.of("10", "20", "30", "40", "50");
        System.out.println("Sum: " + calculateSum(numbers));
    }
}
```

4. Output:



```
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> cd "c:\Users\har
oBoxingUnboxingExample }
Sum: 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.

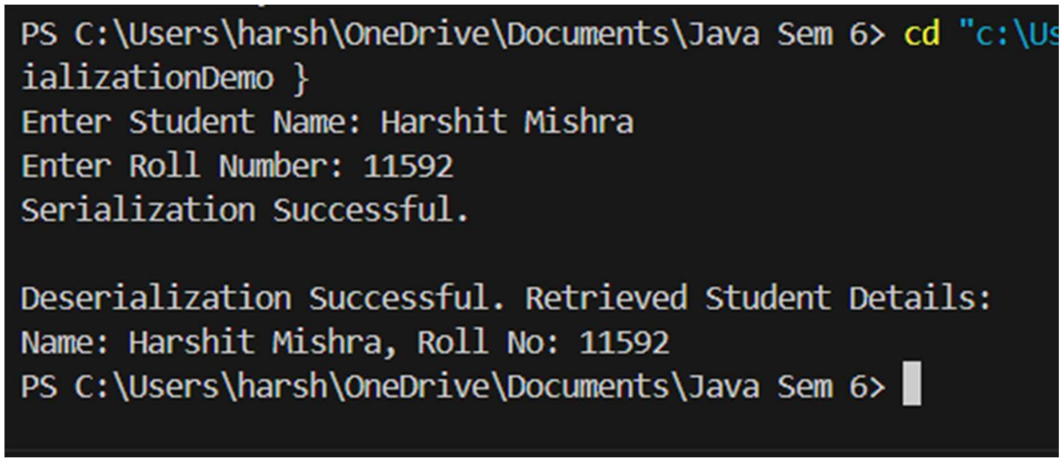
2. **Objective:** This program serializes a Student object (converts it into a byte stream and writes to a file) and deserializes it (retrieves the object from the file).

3. **Implementation/Code:**

```
import java.io.*;
import java.util.Scanner;
class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private int rollNo;
    public Student(String name, int rollNo) {
        this.name = name;
        this.rollNo = rollNo;
    }
    public void display() {
        System.out.println("Name: " + name + ", Roll No: " + rollNo);
    }
}
public class UserSerializationDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String filename = "student.ser";
        // Taking user input
        System.out.print("Enter Student Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Roll Number: ");
        int rollNo = scanner.nextInt();
        Student student = new Student(name, rollNo);
        // Serialization
            try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filename))) {
            out.writeObject(student);
            System.out.println("Serialization Successful.");
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Deserialization
             try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(filename))) {
            Student deserializedStudent = (Student) in.readObject();
                System.out.println("\nDeserialization Successful. Retrieved Student
Details:");
            deserializedStudent.display();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
```

```
            scanner.close();
        }
    }
```

4. Output:



```
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> cd "c:\Us
ializationDemo }
Enter Student Name: Harshit Mishra
Enter Roll Number: 11592
Serialization Successful.

Deserialization Successful. Retrieved Student Details:
Name: Harshit Mishra, Roll No: 11592
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6>
```

5. Learning Outcomes:

- Understanding Java collections.
- Implement key-value storage for categorizing playing cards.
- 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:**

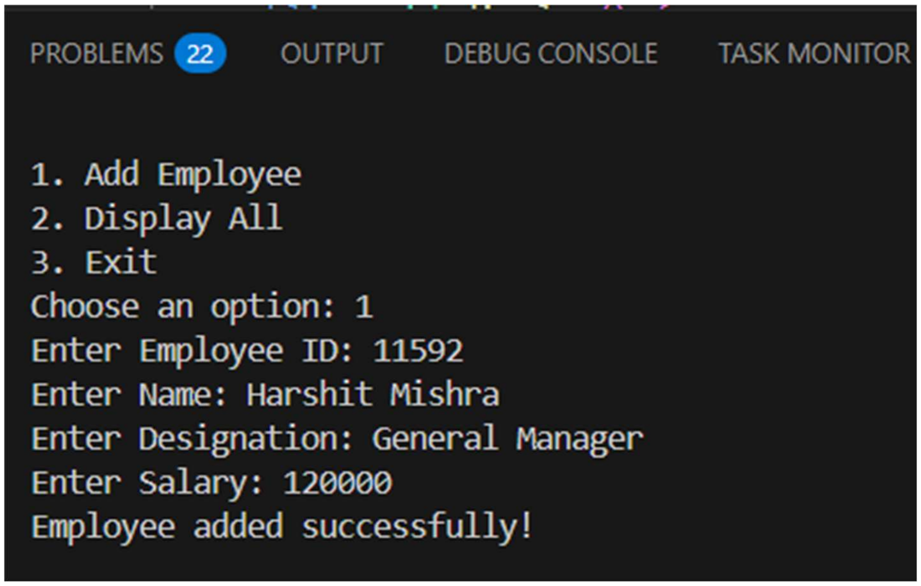
```
import java.io.*;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Employee implements Serializable {

    private static final long serialVersionUID = 1L;

    private int id;

    private String name, 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 EmployeeManagementApp {

    private static final String FILE_NAME = "employees.ser";

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        List<Employee> employees = loadEmployees();

        while (true) {

            System.out.println("\n1. Add Employee\n2. Display All\n3. Exit");

            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
```

```
scanner.nextLine();
switch (choice) {
    case 1:
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter 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));
        saveEmployees(employees);
        System.out.println("Employee added successfully!");
        break;
    case 2:
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
```

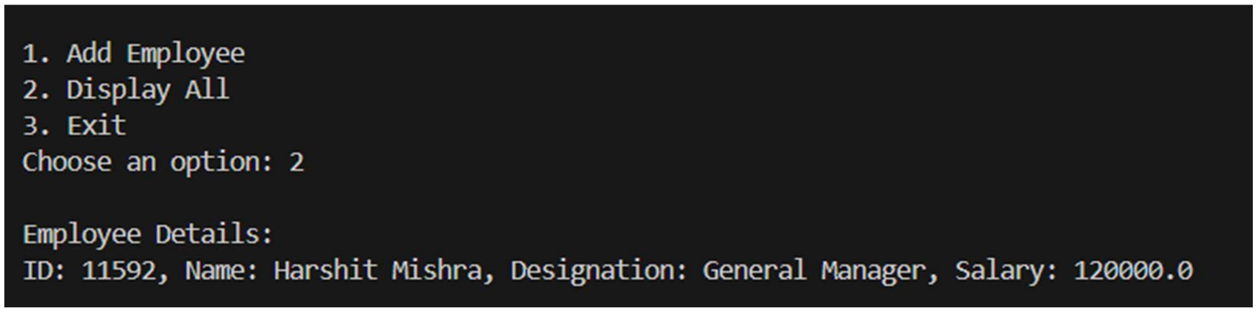
```
                } else {
                    System.out.println("\nEmployee Details:");
                    for (Employee emp : employees) emp.display();
                }
                break;
            case 3:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice! Try again.");
        }
    }
}
private static List<Employee> loadEmployees() {
    File file = new File(FILE_NAME);
    if (!file.exists()) return new ArrayList<>();
```

```
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            return (List<Employee>) in.readObject();
        } catch (IOException | ClassNotFoundException e) {
            return new ArrayList<>();
        }
    }
    private static void saveEmployees(List<Employee> employees) {
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
            out.writeObject(employees);
        } catch (IOException e) {
            System.out.println("Error saving employees.");
        }
    }
}
```

4. Output:



```
1. Add Employee
2. Display All
3. Exit
Choose an option: 1
Enter Employee ID: 11592
Enter Name: Harshit Mishra
Enter Designation: General Manager
Enter Salary: 120000
Employee added successfully!
```



```
1. Add Employee
2. Display All
3. Exit
Choose an option: 2

Employee Details:
ID: 11592, Name: Harshit Mishra, Designation: General Manager, Salary: 120000.0
```

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.