Experiment-5

Student Name: Shubham Sharma **UID:** 22BET10358
Branch: BE-IT **Section:**22BET_IOT-703(A)
Semester: 6th **DOP:** 21/02/2025
Subject: Project Based Learning in JAVA with Lab **Subject Code:** 22ITH-359

PROBLEM 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()).

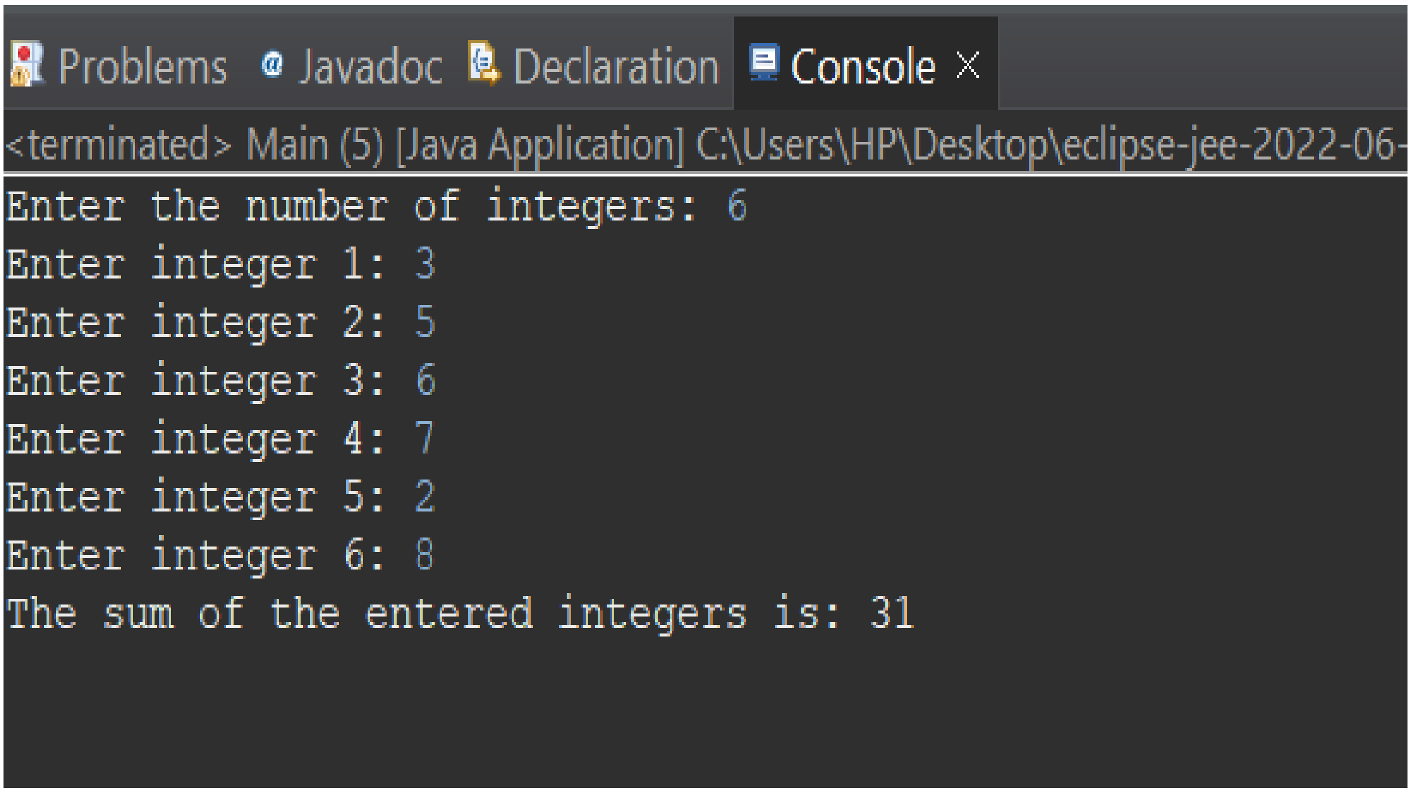
Objective :

- Demonstrate autoboxing and unboxing in Java.
- Use wrapper classes to parse strings into integers.
- Compute the sum of a list of integers efficiently.

Implementation/Code :

```
package AutoboxingSum;

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

public class Main {

    public static Integer parseInteger(String str) {
        try {
            return Integer.parseInt(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format: " + str);
            return null;
        }
    }

    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num;
        }
        return sum;
    }

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

        System.out.print("Enter the number of integers: ");
        int count = scanner.nextInt();
        scanner.nextLine();

        for (int i = 0; i < count; i++) {
            System.out.print("Enter integer " + (i + 1) + ": ");
            String input = scanner.nextLine();
            Integer num = parseInteger(input);
            if (num != null) {
                numbers.add(num);
            } else {
                System.out.println("Invalid input, skipping...");
            }
        }

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

        scanner.close();
    }
}
```

Output :



```
<terminated> Main (5) [Java Application] C:\Users\HP\Desktop\eclipse-jee-2022-06-
Enter the number of integers: 6
Enter integer 1: 3
Enter integer 2: 5
Enter integer 3: 6
Enter integer 4: 7
Enter integer 5: 2
Enter integer 6: 8
The sum of the entered integers is: 31
```

Learning Outcomes :

- Understand autoboxing (automatic conversion of int to Integer).
- Understand unboxing (automatic conversion of Integer to int).
- Learn to use wrapper classes (Integer.parseInt()) for type conversion.
- Work with lists of wrapper class objects (List<Integer>).
- Implement loop-based summation while handling wrapper classes.
- Improve understanding of Java collections and type handling.

PROBLEM 2

Aim :

Create a Java program to serialize and deserialize a Student object.

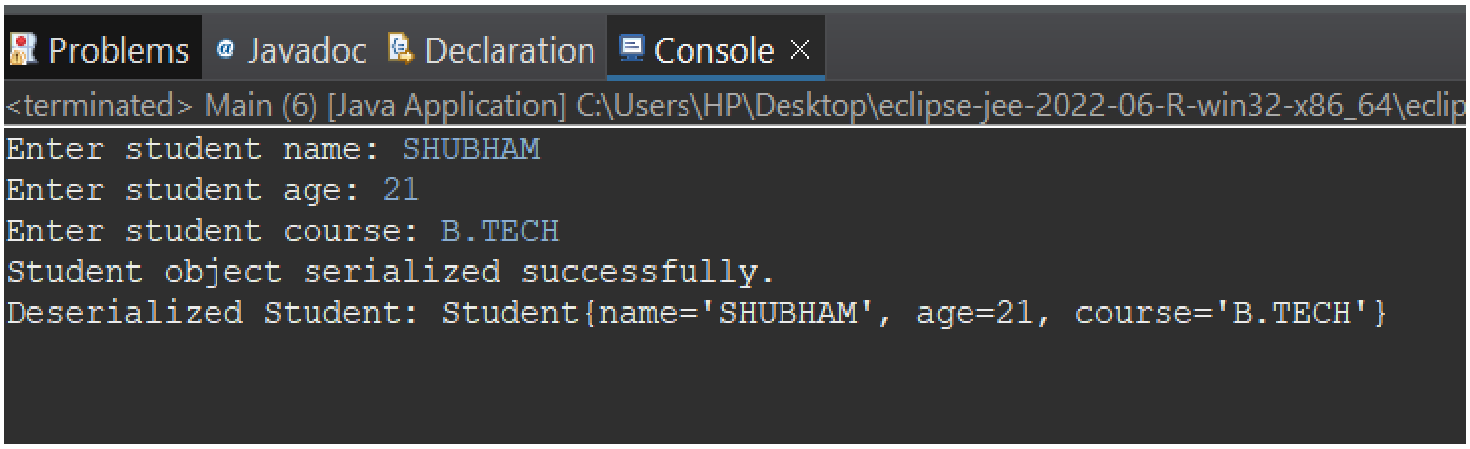
Objective :

- Demonstrate serialization and deserialization of a Java object.
- Use ObjectOutputStream and ObjectInputStream for file operations.
- Implement the Serializable interface in the Student class.

Implementation/Code :

```
package SerializationExample;

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

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private int age;
    private String course;

    public Student(String name, int age, String course) {
        this.name = name;
        this.age = age;
        this.course = course;
    }

    @Override
    public String toString() {
        return "Student{name='" + name + "', age=" + age + ", course='" + course + "'}";
    }
}

public class Main {
    public static void serializeStudent(Student student, String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static Student deserializeStudent(String filename) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {
            return (Student) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return null;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter student name: ");
        String name = scanner.nextLine();
        System.out.print("Enter student age: ");
        int age = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter student course: ");
        String course = scanner.nextLine();

        String filename = "student.ser";

        Student student1 = new Student(name, age, course);

        serializeStudent(student1, filename);

        Student deserializedStudent = deserializeStudent(filename);
        System.out.println("Deserialized Student: " + deserializedStudent);

        scanner.close();
    }
}
```

Output :



```
<terminated> Main (6) [Java Application] C:\Users\HP\Desktop\eclipse-jee-2022-06-R-win32-x86_64\eclip
Enter student name: SHUBHAM
Enter student age: 21
Enter student course: B.TECH
Student object serialized successfully.
Deserialized Student: Student{name='SHUBHAM', age=21, course='B.TECH'}
```

Learning Outcomes :

- Understand serialization and deserialization in Java.
- Learn to use ObjectOutputStream for writing objects to a file.
- Learn to use ObjectInputStream for reading objects from a file.
- Implement the Serializable interface for object persistence.
- Handle IOException and ClassNotFoundException during file operations.
- Learn the importance of serialVersionUID in object version control.

PROBLEM 3

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.

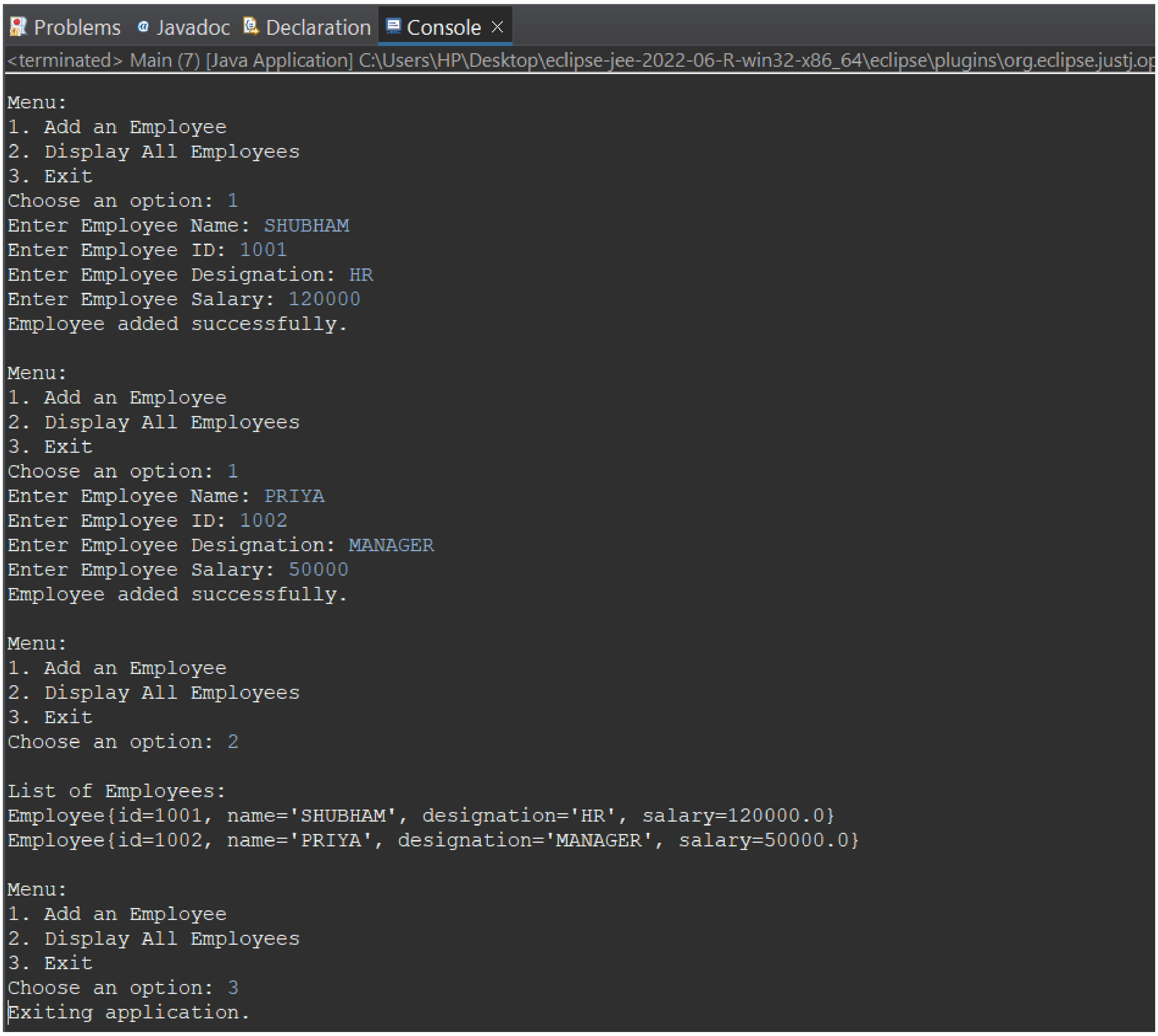
Objective :

- Create a menu-driven Java application for employee management.
- Implement file handling to store and retrieve employee details.
- Use serialization and deserialization to manage employee records.

Implementation/Code :

```
package EmployeeManagement;

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 String name;
    private int id;
    private String designation;
    private double salary;

    public Employee(String name, int id, String designation, double salary) {
        this.name = name;
        this.id = id;
        this.designation = designation;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee{id=" + id + ", name='" + name + "', designation='" + designation + "', salary=" +
        salary + "}";
    }
}

public class Main {
    private static final String FILENAME = "employees.ser";

    public static void addEmployee() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter Employee Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();

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


        List<Employee> employees = readEmployees();


        employees.add(employee);


        writeEmployees(employees);

        System.out.println("Employee added successfully.");
    }

    public static void displayEmployees() {
        List<Employee> employees = readEmployees();
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            System.out.println("\nList of Employees:");
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }

    private static List<Employee> readEmployees() {
        File file = new File(FILENAME);


        if (!file.exists() || file.length() == 0) {
            return new ArrayList<>();
        }
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILENAME))) {
            return (List<Employee>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error reading employee data. Resetting to a new list.");
            file.delete();
            return new ArrayList<>();
        }
    }

    private static void writeEmployees(List<Employee> employees) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILENAME, false)))
    {
            oos.writeObject(employees);
        } catch (IOException e) {
            System.out.println("Error writing employee data.");
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        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("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayEmployees();
                    break;
                case 3:
                    System.out.println("Exiting application.");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

Output :



```
<terminated> Main (7) [Java Application] C:\Users\HP\Desktop\eclipse-jee-2022-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.op

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee Name: SHUBHAM
Enter Employee ID: 1001
Enter Employee Designation: HR
Enter Employee Salary: 120000
Employee added successfully.

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee Name: PRIYA
Enter Employee ID: 1002
Enter Employee Designation: MANAGER
Enter Employee Salary: 50000
Employee added successfully.

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

List of Employees:
Employee{id=1001, name='SHUBHAM', designation='HR', salary=120000.0}
Employee{id=1002, name='PRIYA', designation='MANAGER', salary=50000.0}

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 3
Exiting application.
```

Learning Outcomes :

- Understand how to create a menu-driven Java application.
- Learn file handling using serialization and deserialization.
- Implement object persistence using ObjectOutputStream and ObjectInputStream.
- Work with ArrayList to store multiple employee records.
- Handle user input efficiently using Scanner.
- Improve programming logic for real-world applications.