Experiment 05

Student Name: Rajni Gandha
UID: 22BET10080
Branch: BE-IT
Section/Group: IOT_701/A
Semester: 06th
Date of Performance: 18-02-2025
Subject Name: PBLJ
Subject Code: 22ITH-259

Problem 01

1. **Aim:**
   The aim of the experiment is to calculate the sum of a list of integers using autoboxing and unboxing by parsing strings to their wrapper classes.

2. **Objective:**
   - The objective of this experiment is to write code which calculates the sum of a list of integers using autoboxing and unboxing.
   - To include methods to parse strings into their respective wrapper classes.

3. **Implementation:**

```
package com.studyopedia;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Exp5{
   public static List<Integer> parseStringToIntegerList(String[] numbers) {
      List<Integer> integerList = new ArrayList<>();

      for (String num : numbers) {
         integerList.add(Integer.parseInt(num));
      }
      return integerList;
   }

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

   public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter numbers separated by spaces:");
        String input = scanner.nextLine();

        String[] numberStrings = input.split("\\s+");

        List<Integer> numbers = parseStringToIntegerList(numberStrings);

        int sum = calculateSum(numbers);

        System.out.println("The sum of the numbers is: " + sum);

        scanner.close();
    }
}
```

4. **Output:**

5. **Learning Outcomes:**
   - We learned the concept of auto boxing and unboxing which is a conversion between the primitive data types and their wrapper classes.
   - We demonstrated parsing a string array, computed the sum, and displayed the result.

Problem 02

1. **Aim:**

   The aim of the experiment is to create a program to serialize and deserialize a Student object which contains id, name, and GPA of the student and save this in a file.

2. **Objective:**

   - The objective of this experiment is to serialize a Student object which contains id, name, and GPA of the student and then saving it in a file.
   - To deserialize the object from the file and display the details of the student.
   - To handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

3. **Implementation:**

```
package com.studyopedia;
import java.io.*;

class Student implements Serializable {
 private static final long serialVersionUID = 1L;
 int id;
 String name;
 double gpa;

 public Student(int id, String name, double gpa) {
    this.id = id;
    this.name = name;
    this.gpa = gpa;
 }
 public void display() {
    System.out.println("ID: " + id + ", Name: " + name + ", GPA: " + gpa);
 }
}

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

    Student student = new Student(101, "XYZ", 8.8);

    try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(fileName))) {
        out.writeObject(student);
```

```
            System.out.println("Student serialization is successful!");
        }
        catch (IOException e) {
            System.out.println("Serialization error: " + e.getMessage());
        }

        try (ObjectInputStream in = new ObjectInputStream(new
    FileInputStream(fileName))) {
            Student deserializedStudent = (Student) in.readObject();
            System.out.println("Student Details:");
            deserializedStudent.display();
        }
        catch (IOException | ClassNotFoundException e) {
            System.out.println("Deserialization error: " + e.getMessage());
        }
    }
}
```

4. **Output:**

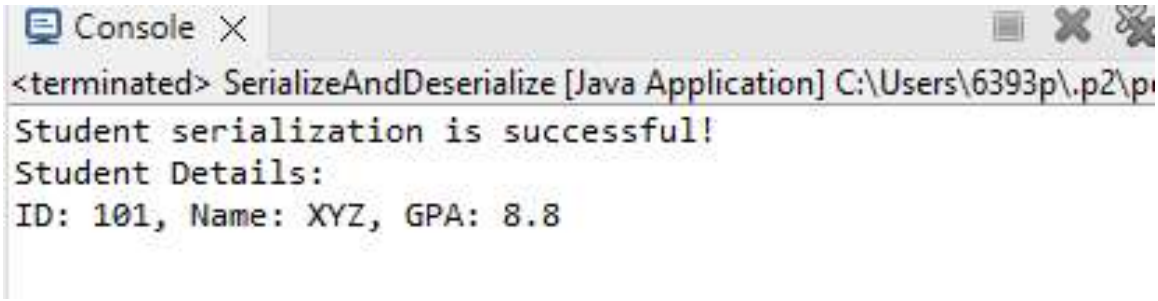

Fig 02: Output of the experiment

5. **Learning Outcomes:**
   - We learned to demonstrate serialization and deserialization of a student object with id, name, and GPA.
   - We learned to manage file and serialization errors effectively.
   - We ensured that the **S**tudent object can be stored and retrieved reliably, demonstrating core Java serialization concepts.

Problem 03

1. **Aim:**

   The aim of the experiment is to create a menu-based Java application with various options which manages the data of the employees.

2. **Objective:**

   - The objective of this experiment is to collect details and save them to a file using serialization.
   - To read and display all the data of employees from the file.
   - To use Integer, Long, and Boolean for automatic boxing and unboxing.

3. **Implementation:**

```
package com.studyopedia;
import java.io.*;
import java.util.*;

class Employee implements Serializable {
   private static final long serialVersionUID = 1L;
   private Integer id;
   private String name;
   private String designation;
   private Long salary;
   private Boolean active;

   public Employee(Integer id, String name, String designation, Long salary, Boolean
active) {
      this.id = id;
      this.name = name;
      this.designation = designation;
      this.salary = salary;
      this.active = active;
   }

   public void display() {
      System.out.printf("ID: %d | Name: %s | Designation: %s | Salary: %d | Active:
%b%n",
            id, name, designation, salary, active);
   }
}

public class EmployeeManagementApp {
```

```
private static final String FILE_NAME = "employees.dat";

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 -> addEmployee(scanner, employees);
            case 2 -> displayEmployees(employees);
            case 3 -> {
                saveEmployees(employees);
                System.out.println("Exiting application...");
                return;
            }
            default -> System.out.println("Invalid choice. Try again.");
        }
    }
}

private static void addEmployee(Scanner scanner, List<Employee> employees) {
    System.out.print("Enter Employee ID: ");
    Integer 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: ");
    Long salary = scanner.nextLong();

    System.out.print("Is Active (true/false): ");
    Boolean active = scanner.nextBoolean();

    Employee emp = new Employee(id, name, designation, salary, active);
    employees.add(emp);
```

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

    private static void displayEmployees(List<Employee> employees) {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
            return;
        }

        System.out.println("\nAll Employees:");
        for (Employee emp : employees) {
            emp.display();
        }
    }

    private static void saveEmployees(List<Employee> employees) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
            System.out.println("Employee data saved.");
        } catch (IOException e) {
            System.err.println("Error saving employees: " + e.getMessage());
        }
    }
    private static List<Employee> loadEmployees() {
        File file = new File(FILE_NAME);
        if (!file.exists()) return new ArrayList<>();

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

4. **Output:**

Fig 03: Output of the experiment

5. **Learning Outcome:**
   - We learned to use Serialization & Deserialization.
   - We demonstrated Autoboxing & Unboxing in our program.
   - We designed a fully Functional Menu System which manages the employee details.