Experiment-5

Name: Kuldeep **UID:** 22BET10168
Branch: BE-IT **Section/Group:** 22BET_IOT-702/A
Semester: 6th **Date of Performance:** 19/02/25
Subject Name: Project Based Learning in Java **Subject Code:** 22ITP-351

Problem-1

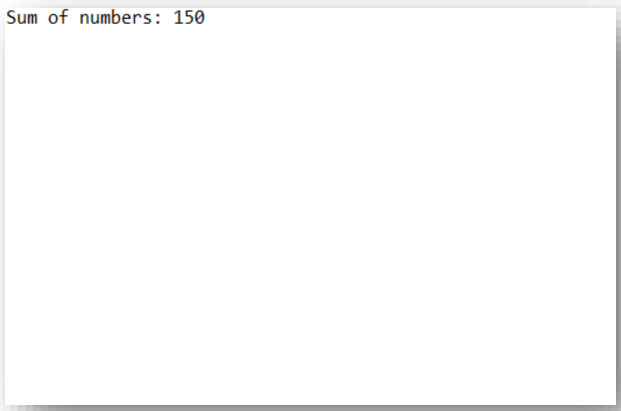
1. Aim:
Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

2. Objective:
The objective of this problem is to demonstrate the concepts of autoboxing and unboxing in Java while working with a list of integers. The program should:
1. Parse Strings to Integer Objects
2. Utilize Autoboxing and Unboxing

3. Code:

```
import java.util.*;

public class AutoboxingUnboxingExample {

    public static void main(String[] args) {

        List<String> stringNumbers = Arrays.asList("10", "20", "30", "40", "50");



        List<Integer> intList = parseStringsToIntegers(stringNumbers);



        int sum = calculateSum(intList);



        System.out.println("Sum of numbers: " + sum);

    }

    public static List<Integer> parseStringsToIntegers(List<String> stringList) {
        List<Integer> intList = new ArrayList<>();

        for (String str : stringList) {

            intList.add(Integer.parseInt(str));

        }

        return intList;

    }

    public static int calculateSum(List<Integer> numbers) {

        int sum = 0;

        for (Integer num : numbers) {

            sum += num;

        }

        return sum;

    }

}
```

4. Output:



```
Sum of numbers: 150
```

Problem-2

1. Aim:

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

2. Objective:

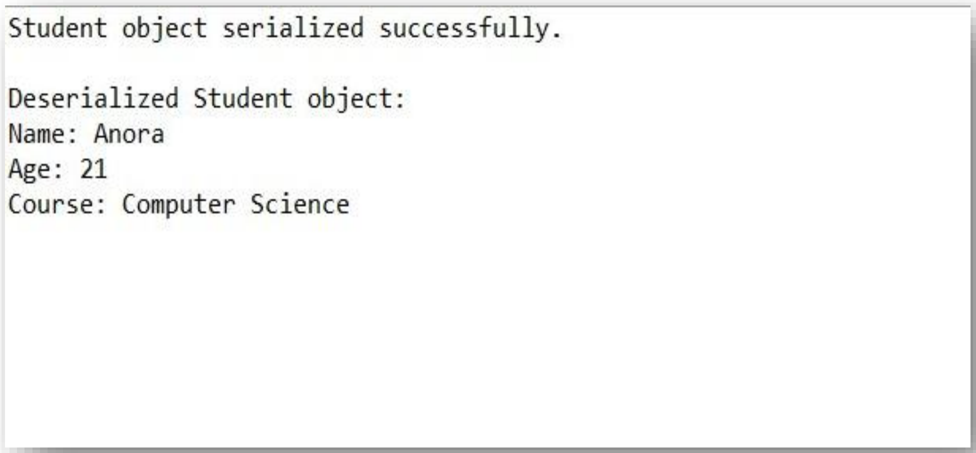
The objective of this program is to demonstrate serialization and deserialization in Java using the Serializable interface. The program should:

1. Create a Student class that implements Serializable, allowing its objects to be saved and restored.
2. Serialize a Student object by writing it to a file using ObjectOutputStream.

3. Code:

```
import java.io.*;

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;

 }

public void display() {

     System.out.println("Name: " + name);

     System.out.println("Age: " + age);

     System.out.println("Course: " + course);

 }
}

public class StudentSerialization {
 public static void main(String[] args) {
     String filename = "student.ser";
     Student student = new Student("Anora", 21, "Computer Science");
     try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filename))) {
         oos.writeObject(student);
         System.out.println("Student object serialized successfully.");
     } catch (IOException e) {
         e.printStackTrace();
     }
     try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {
         Student deserializedStudent = (Student) ois.readObject();
         System.out.println("\nDeserialized Student object:");
         deserializedStudent.display();
     } catch (IOException | ClassNotFoundException e) {
         e.printStackTrace();
     }
 }
}
```

4. Output:



```
Student object serialized successfully.

Deserialized Student object:
Name: Anora
Age: 21
Course: Computer Science
```

Problem-3

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 program is to develop a menu-driven Java application that performs the following tasks:

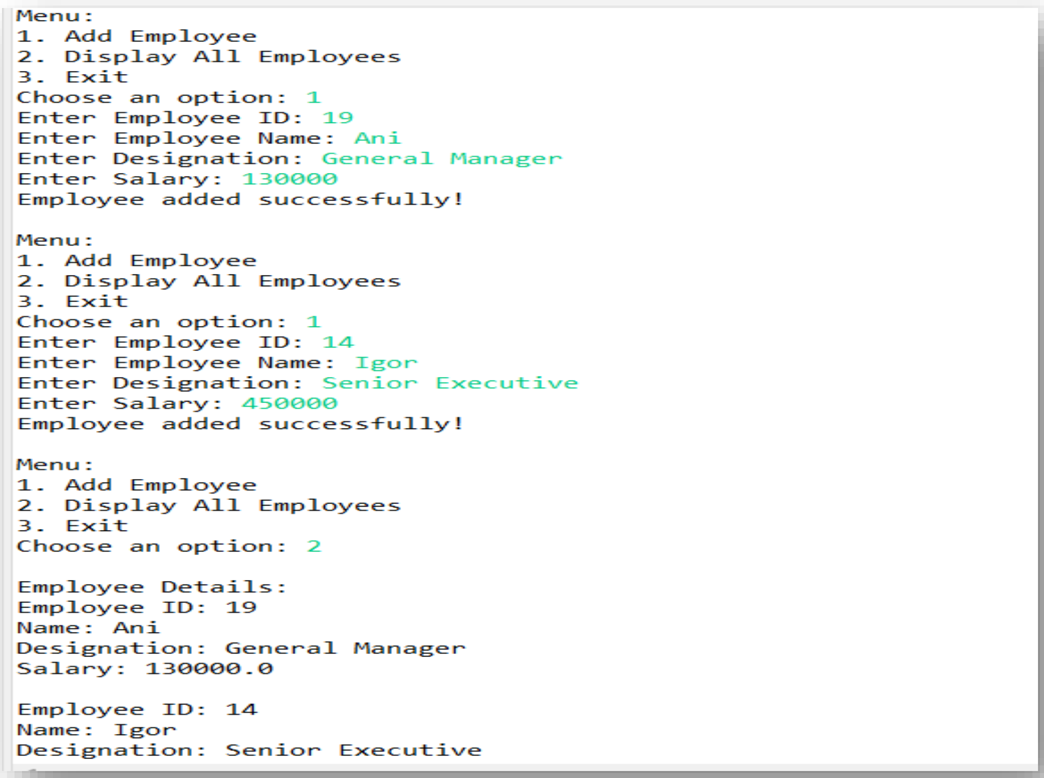
1. Add an Employee – Collect employee details (name, ID, designation, salary) and store them in a file using serialization.
2. Display All Employees – Read the stored employee details from the file and display them.
3. Exit – Terminate the application when the user selects the exit option.

3. 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 empId;

    private String name;

    private String designation;

    private double salary;

    public Employee(int empId, String name, String designation, double salary) {

        this.empId = empId;

        this.name = name;

        this.designation = designation;
        this.salary = salary;

    }

    @Override

    public String toString() {

        return "Employee ID: " + empId + "\nName: " + name + "\nDesignation: " + designation +
         "\nSalary: " + salary + "\n";

    }

}



public class EmployeeManagement {

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

    public static void addEmployee() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter Employee ID: ");

        int empId = 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(empId, name, designation, salary);

        List<Employee> employees = readEmployees(); // Read existing employees

        employees.add(emp); // Add new employee
        try (ObjectOutputStream oos = new ObjectOutputStream(new
         FileOutputStream(FILE_NAME))) {

            oos.writeObject(employees);

            System.out.println("Employee added successfully!\n");

        } catch (IOException e) {

            System.out.println("Error saving employee data.");

            e.printStackTrace();

        }

    }

    public static void displayEmployees() {

        List<Employee> employees = readEmployees();

        if (employees.isEmpty()) {

            System.out.println("No employees found.\n");

        } else {

            System.out.println("\nEmployee Details:");

            for (Employee emp : employees) {

                System.out.println(emp);

            }

        }

    }

    private static List<Employee> readEmployees() {

        List<Employee> employees = new ArrayList<>();

        File file = new File(FILE_NAME);

        if (file.exists()) {

            try (ObjectInputStream ois = new ObjectInputStream(new
         FileInputStream(FILE_NAME))) {
                employees = (List<Employee>) ois.readObject();
            } catch (IOException | ClassNotFoundException e) {
                System.out.println("Error reading employee data.");
            }
        }
        return employees;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("Menu:");
            System.out.println("1. Add Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayEmployees();
                    break;
                case 3:
                    System.out.println("Exiting the program.");
```

```
                        scanner.close();

                        System.exit(0);

                        break;

                    default:

                        System.out.println("Invalid choice! Please try again.");

                }

            }

        }

    }
```

4. Output:



```
Menu:
1. Add Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 19
Enter Employee Name: Ani
Enter Designation: General Manager
Enter Salary: 130000
Employee added successfully!

Menu:
1. Add Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 14
Enter Employee Name: Igor
Enter Designation: Senior Executive
Enter Salary: 450000
Employee added successfully!

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

Employee Details:
Employee ID: 19
Name: Ani
Designation: General Manager
Salary: 130000.0

Employee ID: 14
Name: Igor
Designation: Senior Executive
```