Experiment 5.1

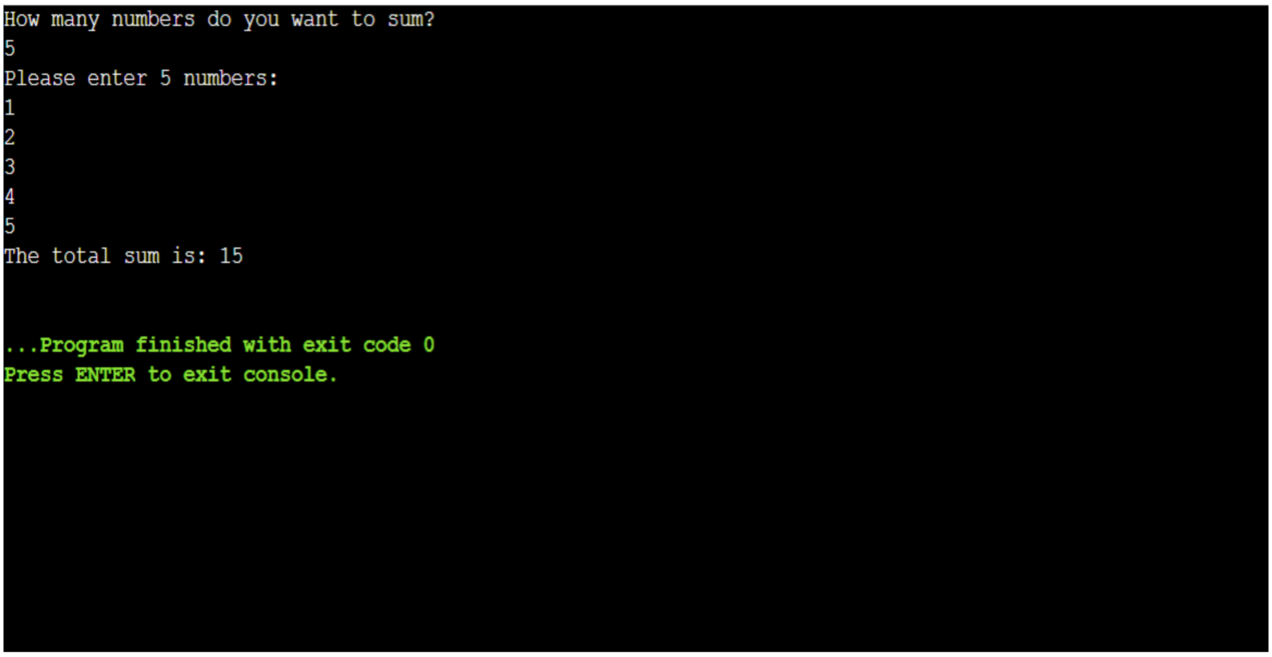
Student Name: SAHIL	**UID: 22BCS14873**
Branch: CSE	**Section: 22BCS_IOT-642-A**
Semester: 6th	**DOP: 04/03/25**
Subject: PBLJ	**Subject Code:22CSH-359**

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: The goal of this Java program is to demonstrate autoboxing and unboxing while calculating the sum of a list of integers.

Code:
```
import java.util.*;

public class NumberSumCalculator {

    public static List<Integer> convertStringsToIntegers(List<String> stringNumbers) {
        List<Integer> integerNumbers = new ArrayList<>();
        for (String str : stringNumbers) {
            integerNumbers.add(Integer.valueOf(str));     // Using Integer.valueOf instead of
parseInt
        }
        return integerNumbers;
    }

    public static int computeSum(List<Integer> nums) {
        int total = 0;
        for (Integer num : nums) {
            total += num;  // Simplified the addition statement
        }
        return total;
    }

    public static void main(String[] args) {
        Scanner inputScanner = new Scanner(System.in);

        System.out.println("How many numbers do you want to sum?");
        int count = inputScanner.nextInt();
        inputScanner.nextLine();  // Consume the newline character after the number

        List<String> inputStrings = new ArrayList<>();
        System.out.println("Please enter " + count + " numbers:");

        for (int i = 0; i < count; i++) {
            inputStrings.add(inputScanner.nextLine());
        }
        List<Integer> integerList = convertStringsToIntegers(inputStrings);

        int totalSum = computeSum(integerList);

        System.out.println("The total sum is: " + totalSum);

        inputScanner.close();
    }
}
```

Output:



```
How many numbers do you want to sum?
5
Please enter 5 numbers:
1
2
3
4
5
The total sum is: 15


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

- Understand the concept of autoboxing and unboxing in Java and how primitive types are automatically converted to their wrapper classes and vice versa.
- Learn how to convert string values into Integer objects using Integer.parseInt() and store them in a list.
- Gain experience in working with ArrayLists to store and manipulate a collection of numbers dynamically.
- Develop proficiency in iterating through collections and performing arithmetic operations like summation.

Experiment 5.2

1. **Aim:** Create a Java program to serialize and deserialize a Student object. The program should: Serialize a Student object (containing id, name, and GPA) and save it to a file. Deserialize the object from the file and display the student details. Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

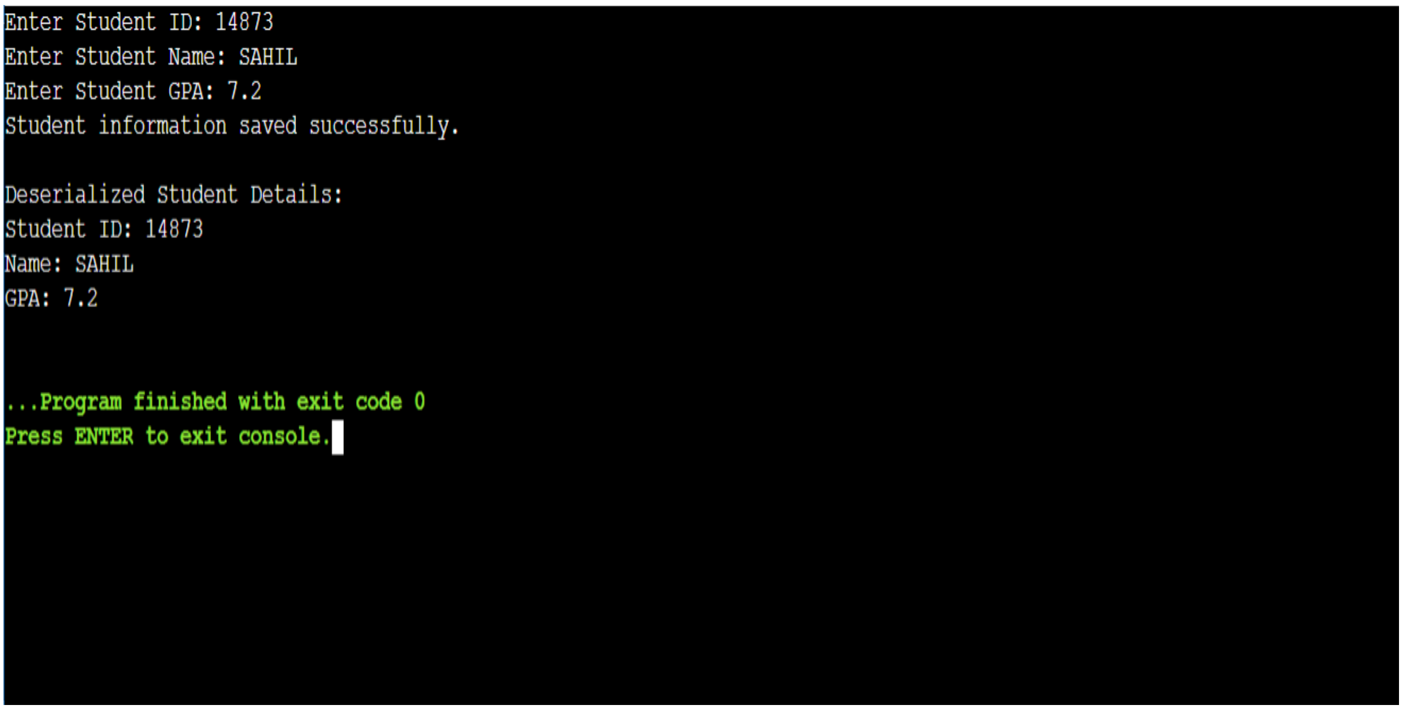
2. **Objective:** The objective is to serialize and deserialize a Student object, store and retrieve its id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

3. Implementation Code:

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

class Learner implements Serializable {
    static final long serialVersionUID = 2L;  // Updated serialVersionUID
    int studentId;
    String fullName;
    double gradePointAverage;

    public Learner(int studentId, String fullName, double gradePointAverage) {
        this.studentId = studentId;
        this.fullName = fullName;
        this.gradePointAverage = gradePointAverage;
    }

    public void showDetails() {
        System.out.println("Student ID: " + studentId);
        System.out.println("Name: " + fullName);
        System.out.println("GPA: " + gradePointAverage);
    }
}

public class StudentFileSerialization {

    public static void saveStudent(Learner learner, String fileName) {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            outputStream.writeObject(learner);
            System.out.println("Student information saved successfully.");
        } catch (IOException e) {
            System.err.println("Serialization error: " + e.getMessage());
        }
    }

    public static Learner loadStudent(String fileName) {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(fileName))) {
```

```
            return (Learner) inputStream.readObject();
        } catch (FileNotFoundException e) {
            System.err.println("The file doesn't exist: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Error during deserialization: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Class not found during deserialization: " + e.getMessage());
        }
        return null;
    }

    public static void main(String[] args) {
        Scanner inputScanner = new Scanner(System.in);

        System.out.print("Enter Student ID: ");
        int id = inputScanner.nextInt();
        inputScanner.nextLine();  // Consume newline

        System.out.print("Enter Student Name: ");
        String name = inputScanner.nextLine();

        System.out.print("Enter Student GPA: ");
        double gpa = inputScanner.nextDouble();

        Learner learner = new Learner(id, name, gpa);

        String file = "studentData.ser";

        saveStudent(learner, file);

        Learner loadedLearner = loadStudent(file);

        if (loadedLearner != null) {
            System.out.println("\nDeserialized Student Details:");
            loadedLearner.showDetails();
        }

        inputScanner.close();
    }
}
```

4. Output



```
Enter Student ID: 14873
Enter Student Name: SAHIL
Enter Student GPA: 7.2
Student information saved successfully.

Deserialized Student Details:
Student ID: 14873
Name: SAHIL
GPA: 7.2


...Program finished with exit code 0
Press ENTER to exit console.
```

5. Learning Outcomes:

- Understand object serialization and deserialization in Java.
- Learn how to use ObjectOutputStream and ObjectInputStream for file operations.
- Implement exception handling for FileNotFoundException, IOException, and ClassNotFoundException.
- Gain hands-on experience in storing and retrieving objects from a file.
- Develop skills in data persistence and file management using Java.

Experiment 5.3

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.

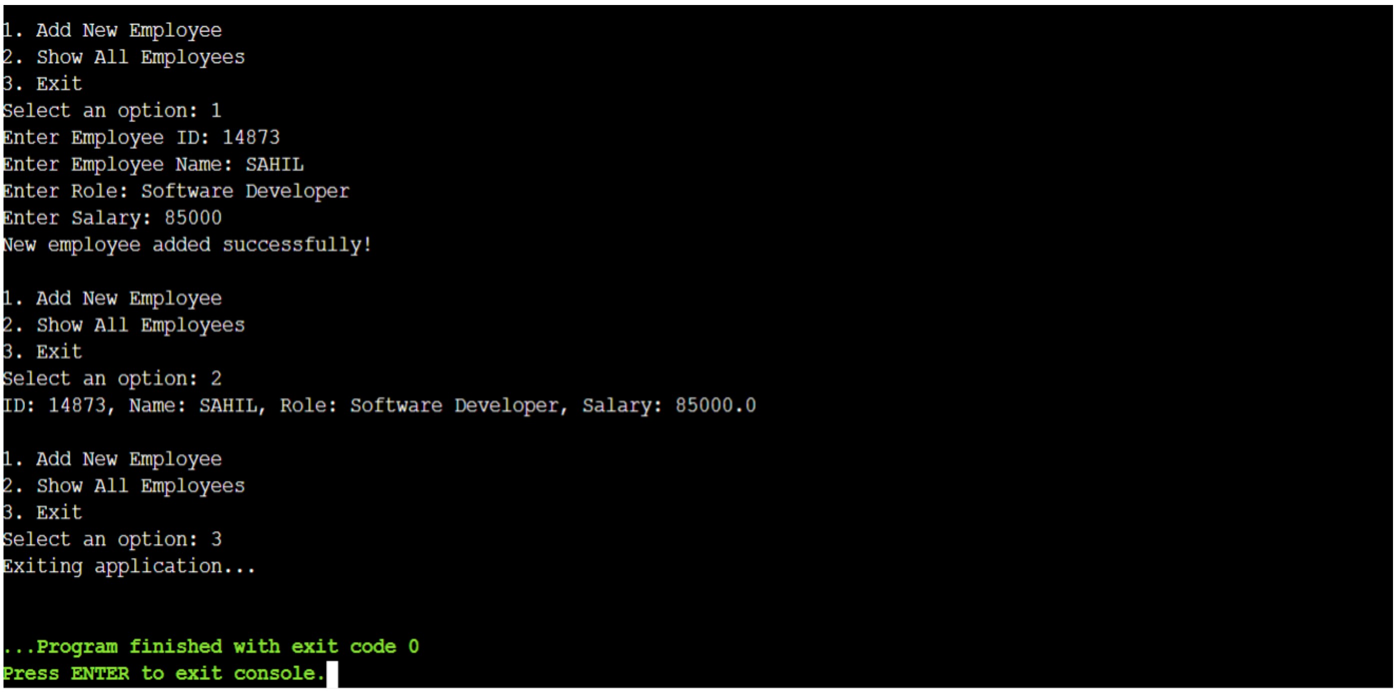
2. **Objective:** The objective of this Java application is to create a **simple** menu-driven employee management **system** using file handling for data persistence.

3. **Implementation Code:**

```
import java.io.*;
import java.util.*;

class StaffMember {
    int employeeId;
    String fullName;
    String role;
    double annualSalary;

    public StaffMember(int employeeId, String fullName, String role, double annualSalary) {
        this.employeeId = employeeId;
        this.fullName = fullName;
        this.role = role;
        this.annualSalary = annualSalary;
    }

    @Override
    public String toString() {
        return employeeId + "," + fullName + "," + role + "," + annualSalary;
    }

    public static StaffMember fromString(String data) {
        String[] fields = data.split(",");
        return new StaffMember(Integer.parseInt(fields[0]), fields[1], fields[2],
Double.parseDouble(fields[3]));
    }
}

public class EmployeeSystem {
    static final String EMPLOYEE_FILE = "employeeRecords.txt";

    public static void addNewEmployee() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee ID: ");
        int employeeId = scanner.nextInt();
        scanner.nextLine();  // Consume the newline
        System.out.print("Enter Employee Name: ");
        String fullName = scanner.nextLine();
        System.out.print("Enter Role: ");
        String role = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double annualSalary = scanner.nextDouble();

        StaffMember newEmployee = new StaffMember(employeeId, fullName, role, annualSalary);

        try (FileWriter fw = new FileWriter(EMPLOYEE_FILE, true);
             BufferedWriter bw = new BufferedWriter(fw);
             PrintWriter pw = new PrintWriter(bw)) {
            pw.println(newEmployee);
        } catch (IOException e) {
            System.err.println("Error saving employee information: " + e.getMessage());
        }

        System.out.println("New employee added successfully!");
    }

    public static void showAllEmployees() {
        File employeeFile = new File(EMPLOYEE_FILE);
        if (!employeeFile.exists()) {
            System.out.println("No employee records found.");
            return;
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(EMPLOYEE_FILE))) {
            String line;
            while ((line = reader.readLine()) != null) {
                StaffMember employee = StaffMember.fromString(line);
                System.out.println("ID: " + employee.employeeId + ", Name: " + employee.fullName +
                        ", Role: " + employee.role + ", Salary: " + employee.annualSalary);
            }
        } catch (IOException e) {
            System.err.println("Error reading employee records: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n1. Add New Employee");
            System.out.println("2. Show All Employees");
            System.out.println("3. Exit");
            System.out.print("Select an option: ");
            int option = scanner.nextInt();

            switch (option) {
                case 1:
                    addNewEmployee();
                    break;
                case 2:
                    showAllEmployees();
```

```
                        break;
                    case 3:
                        System.out.println("Exiting application...");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Invalid option. Please try again.");
                }
            }
        }
    }
```

4. Output:



```
1. Add New Employee
2. Show All Employees
3. Exit
Select an option: 1
Enter Employee ID: 14873
Enter Employee Name: SAHIL
Enter Role: Software Developer
Enter Salary: 85000
New employee added successfully!

1. Add New Employee
2. Show All Employees
3. Exit
Select an option: 2
ID: 14873, Name: SAHIL, Role: Software Developer, Salary: 85000.0

1. Add New Employee
2. Show All Employees
3. Exit
Select an option: 3
Exiting application...


...Program finished with exit code 0
Press ENTER to exit console.
```

5. Learning Outcomes:

- Understand file handling and serialization in Java to store and retrieve objects persistently.
- Learn how to implement a menu-driven console application using loops and conditional statements.
- Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.
- Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
- Develop skills in list manipulation and user input handling using ArrayList and Scanner.