Experiment 5.1

Student Name: Aashna deep	**UID: 22BCS10833**
Branch: CSE	**Section:643-B**
Semester: 6th	**DOP: 24/2/25**
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()).(Easy)

Objective: To develop a Java program that takes space-separated integers as input from the user, processes them into a list of integers, calculates their sum, and displays the result.

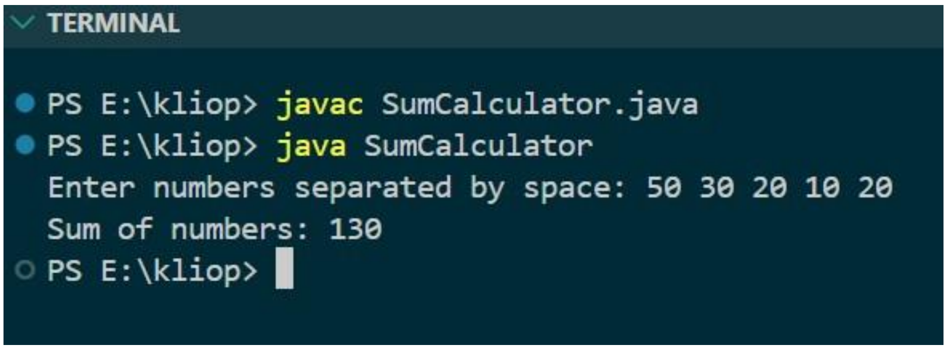
Algorithm:

1. Start
2. Create a Scanner object to read user input.
3. Prompt the user to enter numbers separated by spaces.
4. Read the input as a string.
5. Split the string into an array of number strings using .split(" ").
6. Convert the array of strings into a List<Integer> using a helper method (parseNumbers).
7. Iterate through the string array.
8. Convert each string to an integer and store it in the list.
9. Calculate the sum of all integers in the list using the calculateSum method.
10. Iterate through the list.
11. Add each integer to a sum variable.
12. Print the calculated sum.
13. Close the scanner.
14. **End**

CODE:

```
import java.util.*; public class SumCalculator {     public
  static int calculateSum(List<Integer> numbers) {         int
  sum = 0;

        for (Integer num : numbers) { // Auto-unboxing
  sum += num;
        }
        return sum;
    }
    public static List<Integer> parseNumbers(String[] numberStrings) {
  List<Integer> numbers = new ArrayList<>();         for (String str :
  numberStrings) {             numbers.add(Integer.parseInt(str)); //
  Autoboxing
        }
        return numbers;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter numbers separated by space: ");
        String input = scanner.nextLine();
        String[] numberStrings = input.split(" ");
        List<Integer> numbers = parseNumbers(numberStrings);
  int sum = calculateSum(numbers);
        System.out.println("Sum of numbers: " + sum);
        scanner.close();
    }
}
```

OUTPUT



```
TERMINAL
● PS E:\kliop> javac SumCalculator.java
● PS E:\kliop> java SumCalculator
  Enter numbers separated by space: 50 30 20 10 20
  Sum of numbers: 130
○ PS E:\kliop> █
```

Learning Outcomes:

1. **Understanding Autoboxing and Auto-unboxing** ○ Learn how Java automatically converts between primitive types (int) and wrapper classes (Integer).
2. **Working with Lists in Java** ○ Learn how to create and manipulate List<Integer> using ArrayList.
3. **String Handling and Parsing** ○ Gain experience in using split() to process user input.
   ○ Learn how to convert strings to integers using Integer.parseInt().
4. **Looping and Iteration** ○ Understand how to iterate over lists using enhanced for loops.
5. **Basic Input Handling** ○ Learn how to take user input using Scanner and process it efficiently.

Experiment 5.2

Student Name: Aashna deep **UID: 22BCS10833**
Branch: CSE **Section:643-B**
Semester: 6th **DOP: 24/2/25**
Subject: PBLJ **Subject Code: 22CSH-359**

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.(Medium)

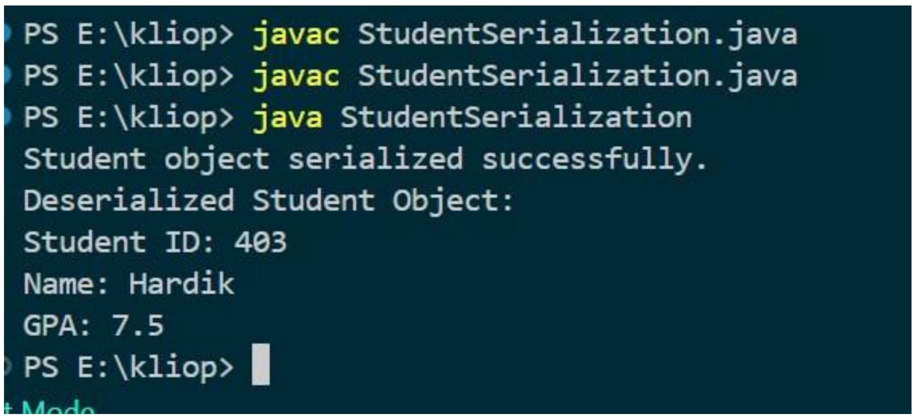
CODE:

```
 import java.io.*;


 // Student class implementing Serializable class

 Student implements Serializable {     private

 static final long serialVersionUID = 1L;

    private int id;

 private String name;

 private double gpa;


    public Student(int id, String name, double gpa) {

        this.id = id;

        this.name = name;

 this.gpa = gpa;

    }

    @Override

    public String toString() {        return "Student ID: " + id + "\nName: "

 + name + "\nGPA: " + gpa;
```

```
    }
}

// Serialization and Deserialization class public class
StudentSerialization {     private static final String
FILE_NAME = "student.ser";

    // Method to serialize Student object     public static
void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (FileNotFoundException e) {
            System.err.println("Error: File not found.");
        } catch (IOException e) {
            System.err.println("Error: Unable to write object to file.");
        }
    }

    // Method to deserialize Student object
public static Student deserializeStudent() {
try (ObjectInputStream ois = new
ObjectInputStream(new
FileInputStream(FILE_NAME))) {

            return (Student) ois.readObject();
        } catch (FileNotFoundException e) {
            System.err.println("Error: File not found.");
        } catch (IOException e) {
            System.err.println("Error: Unable to read object from file.");
        } catch (ClassNotFoundException e) {
            System.err.println("Error: Class not found.");
        }
        return null;
    }


    public static void main(String[] args) {
        // Creating a Student object
        Student student = new Student(403, "Hardik", 7.5);


        // Serializing the Student object
serializeStudent(student);


        // Deserializing the Student object
        Student deserializedStudent = deserializeStudent();
if (deserializedStudent != null) {
```

```
            System.out.println("Deserialized Student Object:");

            System.out.println(deserializedStudent);

         }

      }

   }
```

OUTPUT



```
PS E:\kliop> javac StudentSerialization.java
PS E:\kliop> javac StudentSerialization.java
PS E:\kliop> java StudentSerialization
Student object serialized successfully.
Deserialized Student Object:
Student ID: 403
Name: Hardik
GPA: 7.5
PS E:\kliop>
```

Learning Outcomes:

1. **Understanding Serialization and Deserialization:**
   - Learn how to persist Java objects into files and retrieve them.
2. **Working with Streams (ObjectOutputStream, ObjectInputStream):**
   - Understand how Java handles object writing and reading.
3. **Exception Handling in File Operations:**
   - Learn to handle IOException, FileNotFoundException, and ClassNotFoundException.
4. **Implementing the Serializable Interface:**
   - Understand why serialVersionUID is required for version compatibility.
5. **Working with File Handling:**
   - Gain experience in handling file input/output operations.

Experiment 5.3

Student Name: Aashna deep	UID: 22BCS10833
Branch: CSE	Section:643-B
Semester: 6th	DOP: 24/2/25
Subject: PBLJ	Subject Code: 22CSH-359

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.(Hard)

CODE:

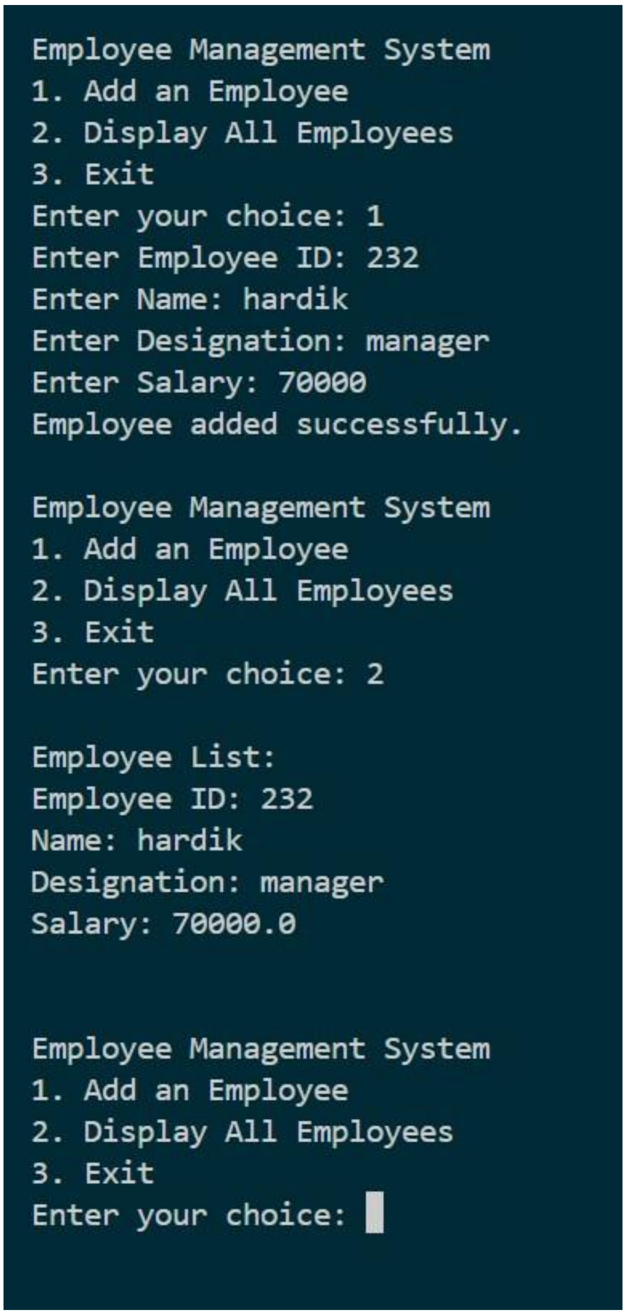
```
import java.io.*; import

java.util.*;


// Employee class implementing Serializable 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";

    }

}


// Main Application Class public class EmployeeManagement

{     private static final String FILE_NAME =

"employees.ser";


    // Method to add an employee     public static void

addEmployee(Employee employee) {

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

employees.add(employee); // Add new employee


        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {

            oos.writeObject(employees);

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

        } catch (IOException e) {

            System.err.println("Error: Unable to write employee data.");
        }
    }

    // Method to get all employees
    @SuppressWarnings("unchecked")
    public static List<Employee> getAllEmployees() {
File file = new File(FILE_NAME);
        if (!file.exists()) {
            return new ArrayList<>(); // Return empty list if file does not exist
        }

        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            return (List<Employee>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error: Unable to read employee data.");
        }
        return new ArrayList<>();
    }

    // Main Menu    public static void
main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\nEmployee Management System");
            System.out.println("1. Add an Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter Employee ID: ");
                    int id = scanner.nextInt();
scanner.nextLine(); // Consume newline

                    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();
```

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

        case 2:
          List<Employee> employees = getAllEmployees();
          if (employees.isEmpty()) {
            System.out.println("No employee records found.");
          } else {
            System.out.println("\nEmployee List:");
for (Employee emp : employees) {
              System.out.println(emp);
            }
}
          break;

        case 3:
          System.out.println("Exiting program...");
          scanner.close();
          System.exit(0);

        default:
          System.out.println("Invalid choice. Please enter a valid option.");
```

```
            }
        }
    }
}
```

OUTPUT



```
Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 232
Enter Name: hardik
Enter Designation: manager
Enter Salary: 70000
Employee added successfully.

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2

Employee List:
Employee ID: 232
Name: hardik
Designation: manager
Salary: 70000.0


Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 
```

Learning Outcomes:

1. Object-Oriented Programming (OOP)

- **Encapsulation:** Employee details are stored in a separate Employee class.
- **Serialization:** The Serializable interface is used to save objects to a file.
- **Abstraction:** The main logic is handled in separate methods (addEmployee, getAllEmployees), making the code modular.

2. File Handling in Java

- Using FileOutputStream and ObjectOutputStream to **write** objects to a file.
- Using FileInputStream and ObjectInputStream to **read** objects from a file.
- Handling file-related exceptions (IOException, FileNotFoundException).