Experiment 5.1

Student Name: Om Mishra **UID:22BCS16609**
Branch: CSE **Section:901/A**
Semester: 6th **DOP:**
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: Demonstrate **autoboxing** and **unboxing** in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

Algorithm:

Step 1: Initialize the Program

1. Start the program.
2. Import ArrayList and List classes.
3. Define the AutoboxingExample class.

Step 2: Convert String Array to Integer List

1. Define the method parseStringArrayToIntegers(String[] strings).
2. Create an empty ArrayList<Integer>.
3. Iterate through the string array:
    - Convert each string to an Integer using Integer.parseInt(str).
    - Add the integer to the list (**autoboxing** happens here).
4. Return the list of integers.

Step 3: Calculate the Sum of Integers

1. Define the method calculateSum(List<Integer> numbers).
2. Initialize a variable sum to 0.
3. Iterate through the list:
    - Extract each integer (**unboxing** happens here).
    - Add it to sum.
4. Return the total sum.

Step 4: Execute Main Function

1. Define main(String[] args).
2. Create a string array with numeric values.
3. Call parseStringArrayToIntegers() to convert it into a list of integers.
4. Call calculateSum() to compute the sum.
5. Print the result.

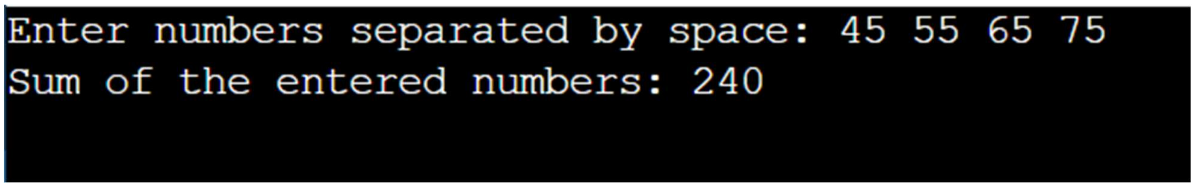
Step 5: Terminate the Program

1. End the execution.

Code:

```
import java.util.*;

public class Main {

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

            Integer num = Integer.parseInt(s);
            intList.add(num);
        }
        return intList;
    }

    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);

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

        String[] parts = input.trim().split("\\s+");
        List<String> stringList = Arrays.asList(parts);

        List<Integer> numbers = parseStringsToIntegers(stringList);

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

        scanner.close();
    }
}
```

Output:



```
Enter numbers separated by space: 45 55 65 75
Sum of the entered numbers: 240
```

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.

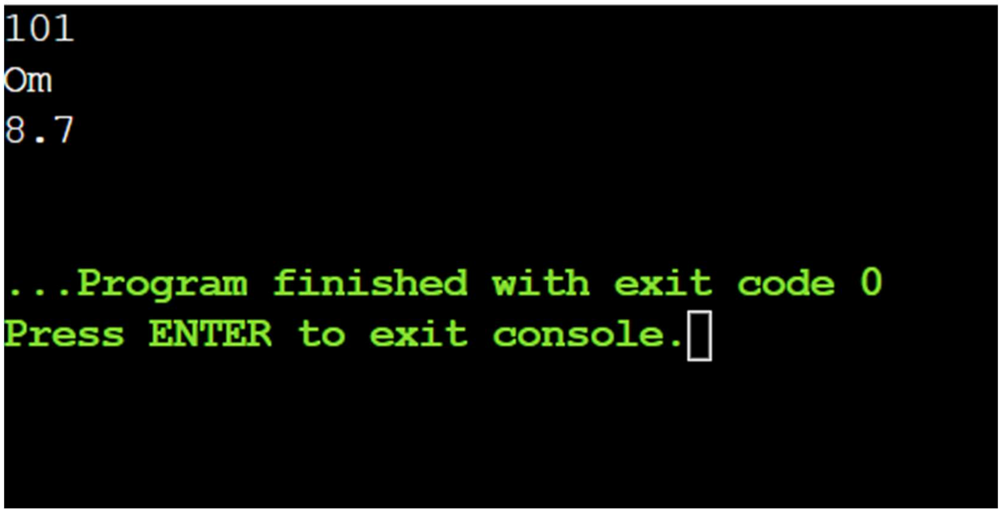
3. Algorithm:

```
Step 1: Initialize the Program
    1. Start the program.
    2. Import the necessary classes (java.io.*).
    3. Define a Student class implementing Serializable.
    4. Declare attributes:
        o id (int) o    name (String) o    gpa (double)
    5. Define a constructor to initialize Student objects.
    6. Override toString() to display student details.
Step 2: Define the Serialization Method
    2. Create serializeStudent(Student student).
    3. Use a try-with-resources block to create an ObjectOutputStream:
        o Open a FileOutputStream to write to student.ser.
        o Write the Student object to the file using writeObject().
    4. Handle exceptions:
        o FileNotFoundException → Print error message.
        o IOException → Print error message.
    5. Print a success message if serialization is successful.
Step 3: Define the Deserialization Method
    1. Create deserializeStudent().
    2. Use a try-with-resources block to create an ObjectInputStream:
        o Open a FileInputStream to read student.ser.
        o Read the Student object using readObject().
    3. Handle exceptions:
        o FileNotFoundException → Print error message.
        o IOException → Print error message.
        o ClassNotFoundException → Print error message.
    4. Print the deserialized student details.
Step 4: Execute Main Function
    1. Define main(String[] args).
    2. Create a Student object with sample data.
    3. Call serializeStudent() to save the object.
    4. Call deserializeStudent() to read and display the object.
Step 5: Terminate the Program
    1. End execution.
```

4. Implementation Code:

```
import java.io.*;

class Student implements Serializable {
    int id;
    String name;
    double gpa;

    Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }
}

public class SerializeDemo {

    public static void main(String[] args) {
        Student s1 = new Student(101, "Om", 8.7);

        try {
            FileOutputStream fileOut = new FileOutputStream("student.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(s1);
            out.close();
            fileOut.close();
        } catch (IOException e) {
            System.out.println("IOException: " + e.getMessage());
        }

        Student deserializedStudent = null;

        try {
            FileInputStream fileIn = new FileInputStream("student.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            deserializedStudent = (Student) in.readObject();
            in.close();
            fileIn.close();
        } catch (FileNotFoundException e) {
            System.out.println("FileNotFoundException: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IOException: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println("ClassNotFoundException: " + e.getMessage());
        }

        if (deserializedStudent != null) {
            System.out.println(deserializedStudent.id);
            System.out.println(deserializedStudent.name);
            System.out.println(deserializedStudent.gpa);
        }
      }
    }
```

5.Output



```
101
Om
8.7


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

6.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 is to develop a menu-based Java application that allows users to **add employee details**, **store them in a file**, and **display all stored employee records**, with an option to exit the program.

3. **Algorithm:**

Step 1: Initialize the Program

1. Start the program.
2. Import java.util.* and java.util.concurrent.* for thread handling.
3. Define a class TicketBookingSystem with:
   - A List<Boolean> representing seat availability (true for available, false for booked).
   - A synchronized method bookSeat(int seatNumber, String passengerName) to ensure thread safety.

Step 2: Implement Seat Booking Logic

1. Define bookSeat(int seatNumber, String passengerName):
   - If the seat is available (true), mark it as booked (false). o Print confirmation: "Seat X booked successfully by Y".
   - If already booked, print: "Seat X is already booked."

Step 3: Define Booking Threads

1. Create a class PassengerThread extending Thread:
   - Store passenger name, seat number, and booking system reference.
   - Implement run() method to call bookSeat().

Step 4: Assign Thread Priorities

1. Create VIP and Regular passenger threads.
2. Set higher priority for VIP passengers using setPriority(Thread.MAX_PRIORITY).
3. Set default priority for regular passengers.

Step 5: Handle User Input & Simulate Booking

1. In main(), create an instance of TicketBookingSystem.
2. Accept number of seats and bookings from the user.
3. Create multiple PassengerThread instances for VIP and regular passengers.
4. Start all threads using start().

Step 6: Synchronization & Preventing Double Booking

1. Use the synchronized keyword in bookSeat() to ensure only one thread accesses it at a time.
2. Ensure thread execution order by assigning higher priority to VIP threads.

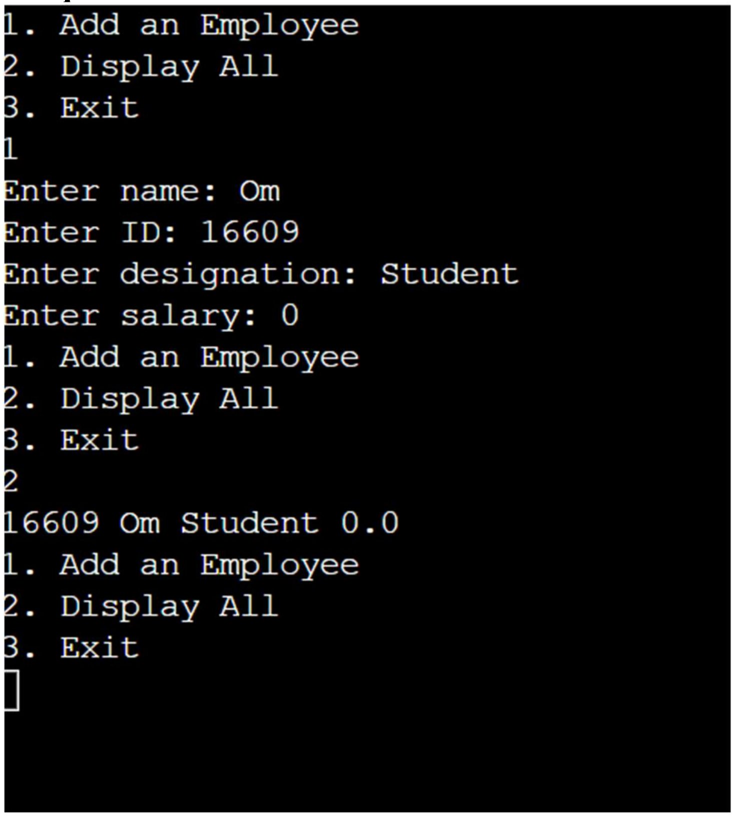
Step 7: Display Final Booking Status

1. After all threads finish execution, display the list of booked seats.
2. End the program with a message: "All bookings completed successfully."

4. **Implementation Code:**

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

class Employee implements Serializable {
    String name;
    int id;
    String designation;
    double salary;

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

    public String toString() {
        return id + " " + name + " " + designation + " " + salary;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Employee> employees = new ArrayList<>();

        try {
            File f = new File("employees.ser");
            if (f.exists()) {
                ObjectInputStream in = new ObjectInputStream(new FileInputStream(f));
                employees = (ArrayList<Employee>) in.readObject();
                in.close();
            }
        } catch (Exception e) {
        }

        while (true) {
            System.out.println("1. Add an Employee\n2. Display All\n3. Exit");
            int choice = sc.nextInt();
            sc.nextLine();

            if (choice == 1) {
                System.out.print("Enter name: ");
                String name = sc.nextLine();
                System.out.print("Enter ID: ");
                int id = sc.nextInt();
                sc.nextLine();
                System.out.print("Enter designation: ");
                String designation = sc.nextLine();
                System.out.print("Enter salary: ");
                double salary = sc.nextDouble();
                employees.add(new Employee(name, id, designation, salary));

                try {
                    ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("employees.ser"));
                    out.writeObject(employees);
                    out.close();
                } catch (IOException e) {
                    System.out.println("Error saving employee");
                }

            } else if (choice == 2) {
                for (Employee emp : employees)
                    System.out.println(emp);
            } else if (choice == 3) {
                break;
            } else {
                System.out.println("Invalid Option");
            }
        }

        sc.close();
    }
}
```

5. Output:



```
1. Add an Employee
2. Display All
3. Exit
1
Enter name: Om
Enter ID: 16609
Enter designation: Student
Enter salary: 0
1. Add an Employee
2. Display All
3. Exit
2
16609 Om Student 0.0
1. Add an Employee
2. Display All
3. Exit
```

6. 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.