Experiment 5

Student Name: Ananya Akhouri **UID: 22BCS10289**
Branch: CSE **Section: 641 A**
Semester: 6th **DOP:24/02/25**
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: 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.ArrayList;
import java.util.List;

public class AutoboxingExample {
    public static void main(String[] args) {
        String[] numberStrings = {"10", "20", "30", "40", "50"};

        List<Integer> numbers = parseStringArrayToIntegers(numberStrings);

        int sum = calculateSum(numbers);

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

    public static List<Integer> parseStringArrayToIntegers(String[] strings) {
        List<Integer> integerList = new ArrayList<>();
        for (String str : strings) {
            integerList.add(Integer.parseInt(str));
        }
        return integerList;
    }

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

Output:

```
The sum of the numbers is: 150


...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. 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:
    - id (int)
    - name (String)
    - gpa (double)
5. Define a constructor to initialize Student objects.
6. Override toString() to display student details.

Step 2: Define the Serialization Method

1. Create serializeStudent(Student student).
2. Use a try-with-resources block to create an ObjectOutputStream:
    - Open a FileOutputStream to write to student.ser.
    - Write the Student object to the file using writeObject().
3. Handle exceptions:
    - FileNotFoundException → Print error message.
    - IOException → Print error message.
4. 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:
    - Open a FileInputStream to read student.ser.
    - Read the Student object using readObject().
3. Handle exceptions:
    - FileNotFoundException → Print error message.
    - IOException → Print error message.
    - 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 {
    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 + ", name='" + name + "', gpa=" + gpa + "}";
    }
}

public class StudentSerialization {
    private static final String FILE_NAME = "student.ser";

    public static void main(String[] args) {
        Student student = new Student(1, "Anwar", 7.8);
        serializeStudent(student);
        deserializeStudent();
    }

    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("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException occurred: " + e.getMessage());
        }
    }

    public static void deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME)))
{
            Student student = (Student) ois.readObject();
            System.out.println("Deserialized Student: " + student);
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException occurred: " + e.getMessage());
        } catch (ClassNotFoundException e) {
```

```
            System.err.println("Class not found: " + e.getMessage());
        }
    }
}
```

5. Output

```
Student object serialized successfully.
Deserialized Student: Student{id=1, name='Anwar', gpa=7.8}


...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

Aim: Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

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.

Code: import java.util.concurrent.locks.*;

```
class TicketBookingSystem {
    private int availableSeats;
    private final Lock lock = new ReentrantLock();

    public TicketBookingSystem(int seats) {
        this.availableSeats = seats;
    }

    public void bookTicket(String user) {
        lock.lock();
        try {
            if (availableSeats > 0) {
                System.out.println(user + " successfully booked a seat.");
                availableSeats--;
            } else {
```

```
            System.out.println(user + " failed to book. No seats available.");
        }
    } finally {
        lock.unlock();
    }
  }
}

class BookingThread extends Thread {
  private TicketBookingSystem system;
  private String user;

  public BookingThread(TicketBookingSystem system, String user, int priority) {
    this.system = system;
    this.user = user;
    setPriority(priority);
  }

  public void run() {
    system.bookTicket(user);
  }
}

public class TicketBooking {
  public static void main(String[] args) {
    TicketBookingSystem system = new TicketBookingSystem(5);

    BookingThread vip1 = new BookingThread(system, "VIP-User1", Thread.MAX_PRIORITY);
    BookingThread vip2 = new BookingThread(system, "VIP-User2", Thread.MAX_PRIORITY);
    BookingThread user1 = new BookingThread(system, "User1", Thread.NORM_PRIORITY);
    BookingThread user2 = new BookingThread(system, "User2", Thread.NORM_PRIORITY);
    BookingThread user3 = new BookingThread(system, "User3", Thread.NORM_PRIORITY);

    vip1.start();
    vip2.start();
    user1.start();
    user2.start();
    user3.start();
  }
}
```

Output:

```
VIP-User1 successfully booked a seat.
VIP-User2 successfully booked a seat.
User1 successfully booked a seat.
User2 successfully booked a seat.
User3 successfully booked a seat.


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

Learning Outcome:

- 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.