Experiment 5.1

Student Name: **Rohan** UID: **22BET10169**
Branch: **BE - IT** Section/Group: **22BET-IOT-701 A**
Semester: **6th** Date of Performance: **18/02/25**
Subject Name: **PBLJ** Subject Code: **22ITH-359**

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 program is to demonstrate **autoboxing and unboxing** in Java while computing the sum of a list of integers. It allows users to input numbers as strings, converts them into Integer objects (autoboxing), and then performs arithmetic operations using unboxed int values. This showcases how Java automatically handles conversions between primitive types and their corresponding wrapper classes.

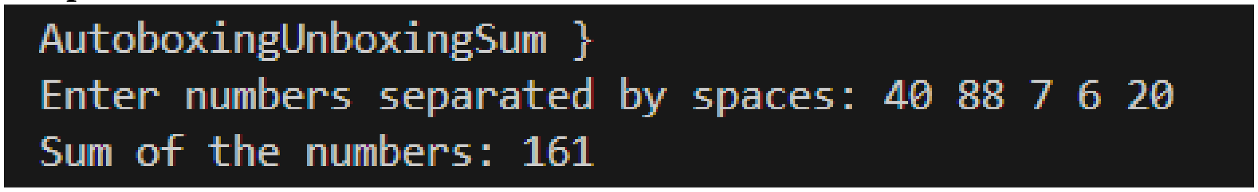
3. Algorithm –

1. Start
2. Take user input: Read a line of space-separated numbers as a string.
3. Split the input: Convert the string into an array of substrings (each representing a number).
4. Initialize an empty list: Create a list to store Integer objects.
5. Convert each substring to Integer:
    - For each number string in the array:
        - Convert it to an int using Integer.parseInt().
        - Autobox the int value into an Integer object and add it to the list.
6. Initialize sum to 0
7. Calculate sum:
    - For each Integer in the list:
        - Unbox it to int and add it to the sum.
8. Display the sum
9. End

4. Implementation/Code:

```java
import java.util.*;

public class AutoboxingUnboxingSum {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter numbers separated by spaces: ");
        String input = scanner.nextLine();
        scanner.close();

        List<String> numberStrings = Arrays.asList(input.split(" "));
        List<Integer> numbers = parseIntegerList(numberStrings);
        int sum = calculateSum(numbers);
        System.out.println("Sum of the numbers: " + sum);
    }

    public static List<Integer> parseIntegerList(List<String> strList) {
        List<Integer> intList = new ArrayList<>();
        for (String str : strList) {
            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;
    }
  }
```

5. Output –



```
AutoboxingUnboxingSum }
Enter numbers separated by spaces: 40 88 7 6 20
Sum of the numbers: 161
```

Experiment 5.2

Student Name: **BIJAYINI BEHERA** UID: **22BCS10780**
Branch: **BE - CSE** Section/Group: **22bcs-IOT-639 A**
Semester: **6th** Date of Performance: **21/2/25**
Subject Name: **PBLJ** Subject Code: **22CSH-359**

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 of this program is to demonstrate object serialization and deserialization in Java. It allows storing a Student object in a file and retrieving it later while ensuring data persistence and handling exceptions efficiently.

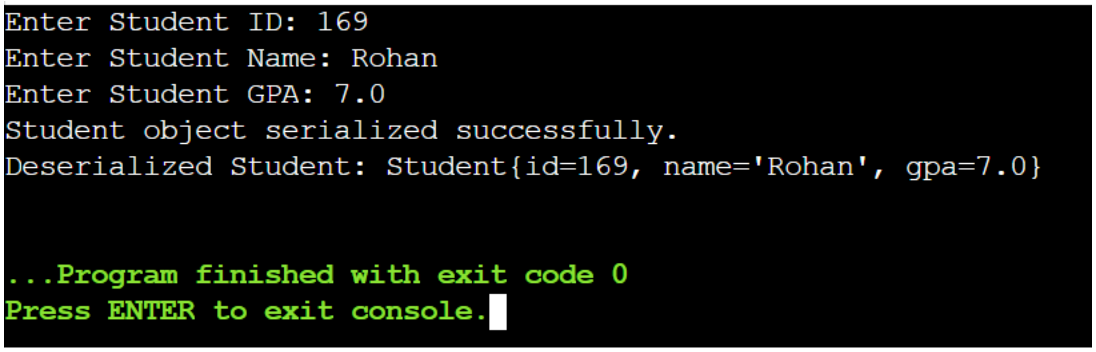
3. **Algorithm –**

- Start
- Define the Student class with attributes: id, name, and gpa, and implement Serializable.
- Create serializeStudent method
- Open a file output stream.
- Use ObjectOutputStream to write the Student object to a file.
- Handle exceptions (IOException).
- Create deserializeStudent method
- Open a file input stream.
- Use ObjectInputStream to read the Student object from the file.
- Handle exceptions (FileNotFoundException, IOException, ClassNotFoundException).
- In the main method
- Create a Student object with sample data.
- Call serializeStudent to save the object.
- Call deserializeStudent to retrieve and print the object.
- End

4. Implementation/Code:

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

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 serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (IOException e) {
            System.err.println("Error during serialization: " + e.getMessage());
        }
    }

    public static Student deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            return (Student) ois.readObject();
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Error during deserialization: " +
e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Class not found: " + e.getMessage());
        }
        return null;
    }

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

        System.out.print("Enter Student ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();

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

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

        scanner.close();

        Student student = new Student(id, name, gpa);
        serializeStudent(student);

        Student deserializedStudent = deserializeStudent();
        if (deserializedStudent != null) {
            System.out.println("Deserialized Student: " + deserializedStudent);
        }
    }
}
```

5. Output –



```
Enter Student ID: 169
Enter Student Name: Rohan
Enter Student GPA: 7.0
Student object serialized successfully.
Deserialized Student: Student{id=169, name='Rohan', gpa=7.0}


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

Experiment 5.3

Student Name: **BIJAYINI BEHERA** UID: **22BCS10780**
Branch: **BE - CSE** Section/Group: **22bcs-IOT-639 A**
Semester: **6th** Date of Performance: **21/2/25**
Subject Name: **PBLJ** Subject Code: **22CSH-359**

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:** to create a **menu-driven Employee Management System** that allows users to add employee details, store them persistently using serialization, and retrieve them when needed. The program ensures data persistence, provides a user-friendly interface, and handles exceptions for a smooth experience.

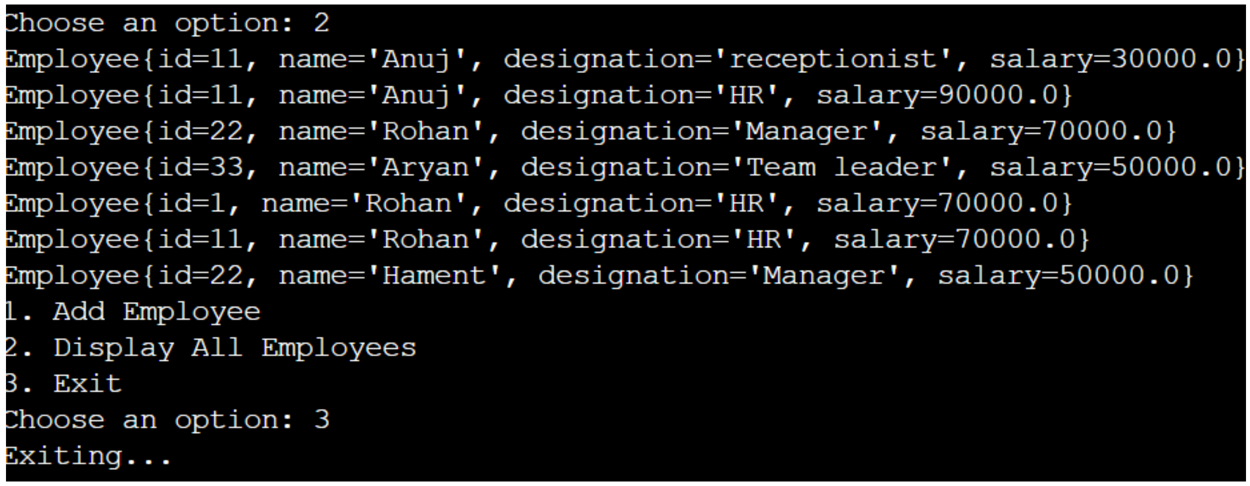
3. **Algorithm –**
   - Start
   - Define the Employee class
   - Implement Serializable to enable object serialization.
   - Declare attributes: id, name, designation, and salary.
   - Implement a constructor and toString() method.
   - Define addEmployee method
   - Read existing employees from the file.
   - Add the new employee to the list.
   - Serialize and save the list back to the file.
   - Define getAllEmployees method
   - Read and deserialize the employee list from the file.
   - Handle exceptions for missing or corrupted files.
   - Implement the main menu loop
   - Display menu options: Add Employee, Display All, Exit.
   - If "Add Employee" is selected:
   - Take user input for employee details.
   - Call addEmployee to store data.

- If "Display All" is selected:
- Retrieve and print all employee details from the file.
- If "Exit" is selected:
- Terminate the program.
- End

4. Implementation/Code:

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

class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

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

    @Override
    public String toString() {
       return "Employee{id=" + id + ", name='" + name + "', designation='" + designation
+ "', salary=" + salary + "}";
    }
}

public class EmployeeManagement {
    private static final String FILE_NAME = "employees.ser";

    public static void addEmployee(Employee employee) {
        List<Employee> employees = getAllEmployees();
        employees.add(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 saving employee: " + e.getMessage());
        }
    }

    public static List<Employee> getAllEmployees() {
                    try    (ObjectInputStream    ois    =    new    ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            return (List<Employee>) ois.readObject();
        } catch (FileNotFoundException e) {
            return new ArrayList<>();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error loading employees: " + e.getMessage());
            return new ArrayList<>();
        }
    }

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

            switch (choice) {
                case 1:
                    System.out.print("Enter Employee ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    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();
                    scanner.nextLine();
                    Employee employee = new Employee(id, name, designation, salary);
                    addEmployee(employee);
                    break;
                case 2:
                    List<Employee> employees = getAllEmployees();
                    if (employees.isEmpty()) {
                        System.out.println("No employees found.");
                    } else {
                        for (Employee emp : employees) {
                            System.out.println(emp);
                        }
                    }
                    break;
                case 3:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Try again.");
            }
        }
    }
}
```

5. Output –



```
Choose an option: 2
Employee{id=11, name='Anuj', designation='receptionist', salary=30000.0}
Employee{id=11, name='Anuj', designation='HR', salary=90000.0}
Employee{id=22, name='Rohan', designation='Manager', salary=70000.0}
Employee{id=33, name='Aryan', designation='Team leader', salary=50000.0}
Employee{id=1, name='Rohan', designation='HR', salary=70000.0}
Employee{id=11, name='Rohan', designation='HR', salary=70000.0}
Employee{id=22, name='Hament', designation='Manager', salary=50000.0}
1. Add Employee
2. Display All Employees
3. Exit
Choose an option: 3
Exiting...
```

6. Learning Outcome-

1. **Autoboxing Sum Program** – Understand **autoboxing and unboxing**, use wrapper classes, parse strings to integers, and perform arithmetic operations.
2. **Student Serialization Program** – Learn **object serialization and deserialization**, handle file operations, and manage exceptions efficiently.
3. **Employee Management System** – Implement a **menu-driven application**, store and retrieve objects using serialization, and manage file handling with error handling.