Experiment 5

Student Name: Farhat **UID: 22BCS12854**
Branch: BE-CSE **Section/Group: IOT-642-B**
Semester: 6th **Date of Performance: 24th Feb, 25**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

Program 1: Wrapper Classes in Java

Aim: Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing, along with methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

Procedures:
1. Create a List of Integers: Initialize a List<Integer> to hold the integers.
2. Autoboxing: Use autoboxing to convert primitive int values to Integer objects automatically when adding to the list.
3. Unboxing: Use unboxing to convert Integer objects back to int for sum calculation.
4. Parse Strings: Create a utility method to parse strings to integers using Integer.parseInt().
5. Calculate the Sum: Use a loop or Java 8 streams to calculate the sum of the list.

Test cases:
Test Case 1:
Input: 10, 20, 30, "40", "50"
Expected Output: The sum of the list is: 150
Description: The list contains a mix of primitive integers and integers parsed from strings.

Test Case 2:
Input: "100", "200", "300"
Expected Output: The sum of the list is: 600
Description: All values are parsed from strings, and the sum is calculated.

Test Case 3:
Input: "50", "invalid", "70"
Expected Output:
Invalid number format: invalid
The sum of the list is: 120
Description: One of the inputs is not a valid integer, so it's skipped, and the sum of valid

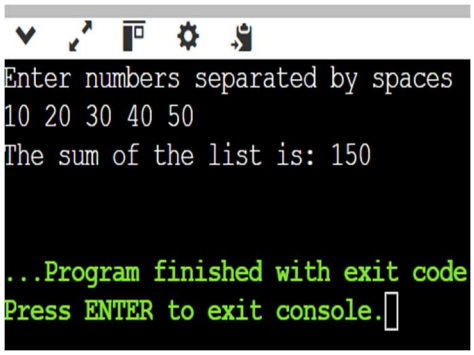
values is calculated.

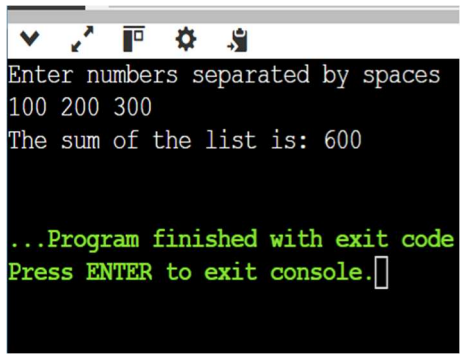
Program/Code:

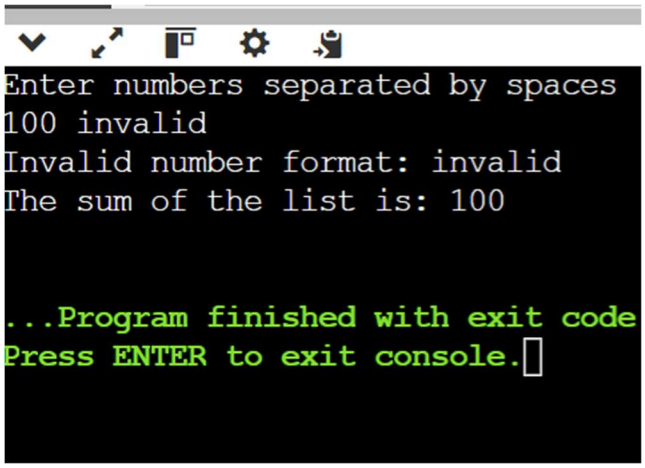
```
import java.util.*;

public class IntegerSumCalculator {
    public static Integer parseStringToInteger(String str) {
        try {
            return Integer.parseInt(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format: " + str);
            return null;
        }
    }

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Integer> numbers = new ArrayList<>();

        System.out.println("Enter numbers separated by spaces (non-numeric values will be
ignored):");
        String input = scanner.nextLine();
        String[] inputs = input.split(" ");

        for (String str : inputs) {
            Integer parsedNumber = parseStringToInteger(str);
            if (parsedNumber != null) {
                numbers.add(parsedNumber);
            }
        }
```

```
        int sum = calculateSum(numbers);
        System.out.println("The sum of the list is: " + sum);

        scanner.close();
    }
}
```

Output:







Program 2: Streams and Serialization

Aim: Write a Java program that serializes and deserializes a Student object. It saves the Student object to a file and then reads it back, displaying the student details.
The program handles exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

Procedure:

1. Create a Student class with id, name, and GPA.
2. Serialize the Student object: Convert the object to a byte stream and save it to a file.
3. Deserialize the Student object: Read the byte stream from the file and convert it back into an object.
4. Exception handling: Handle possible exceptions such as FileNotFoundException, IOException, and ClassNotFoundException.

Test cases:

Test Case 1: Serialize and Deserialize a valid student object.

Input: Student(1, "John Doe", 3.75)
Expected Output:
Student object has been serialized and saved to file.
Student object has been deserialized.
Deserialized Student Details:
Student ID: 1, Name: John Doe, GPA: 3.75

Test Case 2: Try to deserialize from a non-existent file.
Expected Output:
Error: File not found.

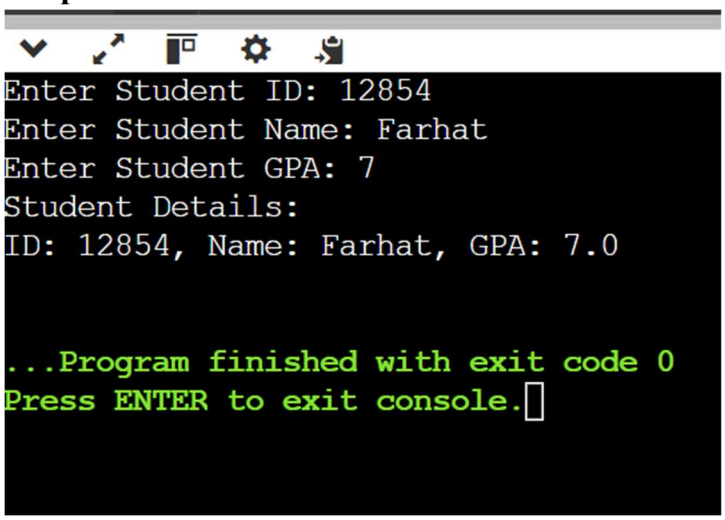
Test Case 3: Handle invalid class during deserialization.
Input: Manually modify the class file to simulate a ClassNotFoundException.
Expected Output:
Error: Class not found.

Program/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;
    }

    public void displayStudent() {
        System.out.println("Student Details:");
        System.out.println("ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}

public class StudentData {

    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();

        Student student = new Student(id, name, gpa);
        student.displayStudent();
```

```
        scanner.close();
    }
}
```

Output:



Program 3: Lambda Expressions and Functional Programming

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

Procedure:
1. Create an Employee class with fields like name, id, designation, and salary.
2. Create a menu with three options:
Add an Employee
Display All Employees
Exit
3. Store Employee Data in a File: Serialize the employee objects and store them in a file.
4. Read Employee Data from the File: Deserialize the employee objects from the file and display the details.
5. Handle Exceptions: Handle file I/O exceptions.

Test cases:
Test Case 1: Add a new employee and display all employees.
Steps: Select option 1 to add a new employee, then select option 2 to display all employees.
Input:
Employee Name: John Doe
Employee ID: 101
Designation: Software Engineer
Salary: 50000

Expected Output:
Employee added successfully!
Employee ID: 101, Name: John Doe, Designation: Software Engineer, Salary: 50000.0

Test Case 2: Try adding multiple employees and display all of them.
Steps: Add multiple employees (using option 1) and then display all employees (using option 2).
Expected Output:
Employee added successfully!

Employee ID: 101, Name: John Doe, Designation: Software Engineer, Salary: 50000.0
Employee added successfully!
Employee ID: 102, Name: Jane Smith, Designation: Manager, Salary: 75000.0

Program/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;
    }

    public void displayEmployee() {
        System.out.println("Employee ID: " + id + ", Name: " + name + ", Designation: " +
designation + ", Salary: " + salary);
    }
}

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

    public static void addEmployee() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee Designation: ");
        String designation = scanner.nextLine();
```

```
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();

        Employee employee = new Employee(id, name, designation, salary);
        saveEmployeeToFile(employee);
        System.out.println("Employee added successfully!");
    }

    private static void saveEmployeeToFile(Employee employee) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME, true))) {
            oos.writeObject(employee);
        } catch (IOException e) {
            System.out.println("Error saving employee: " + e.getMessage());
        }
    }

    public static void displayAllEmployees() {
        List<Employee> employees = readEmployeesFromFile();
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            for (Employee emp : employees) {
                emp.displayEmployee();
            }
        }
    }

    private static List<Employee> readEmployeesFromFile() {
        List<Employee> employees = new ArrayList<>();
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            while (true) {
                try {
                    Employee emp = (Employee) ois.readObject();
                    employees.add(emp);
                } catch (EOFException e) {
                    break;
                }
            }
        } catch (FileNotFoundException e) {
```

```
            System.out.println("Error: File not found.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error reading employees: " + e.getMessage());
        }
        return employees;
    }

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

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayAllEmployees();
                    break;
                case 3:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

Output:

```
Menu:
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 12854
Enter Employee Name: Farhat
Enter Employee Designation: R&D
Enter Employee Salary: 300000
Employee added successfully!
```

```
Menu:
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 3
Exiting...


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