Experiment 5

Name: Sagar Yadav **UID: 22BET10007**
Branch: BE-IT **Section/Group: 22BET_IOT_701/A**
Semester: 6th **Date of Performance: 18-02-25**
Subject Name: Project based Learning in Java **Subject Code: 22ITH-352**

Problem 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()).

Objective:

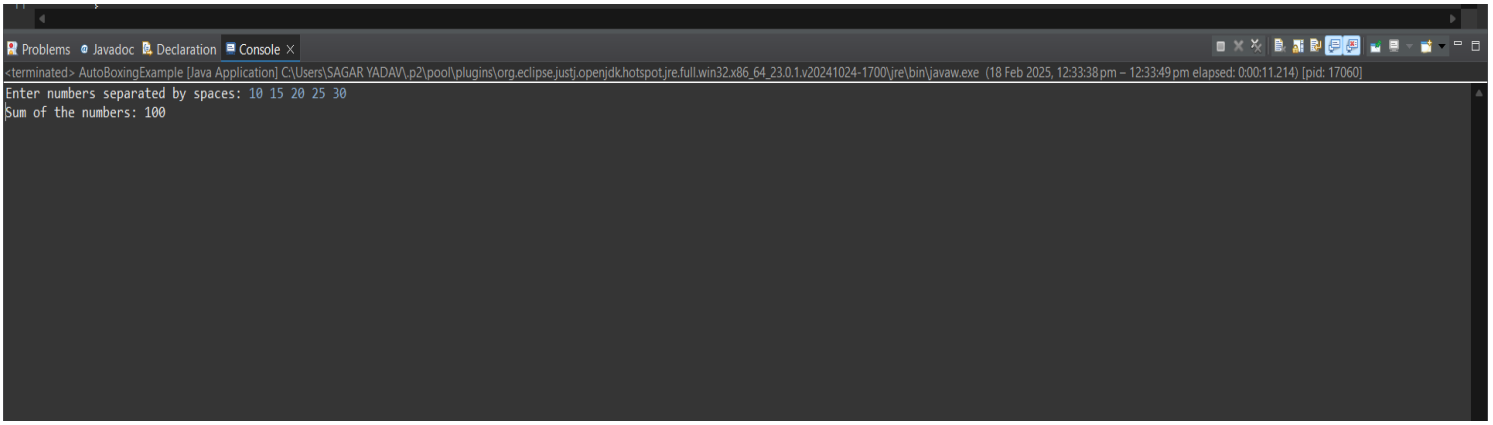
- To demonstrate how Java automatically converts between int (primitive) and Integer (wrapper class).
- To use an ArrayList to store and manage numbers dynamically.
- To convert string inputs into integers using Integer.parseInt().

Code:

```
import java.util.ArrayList;
import java.util.Scanner;

public class AutoBoxingExample {
    // Method to parse strings and convert them into Integer objects
    public static ArrayList<Integer> parseStringArray(String[] strNumbers) {
        ArrayList<Integer> numbers = new ArrayList<>();
        for (String str : strNumbers) {
            numbers.add(Integer.parseInt(str));  // Autoboxing (int → Integer)
        }
        return numbers;
    }
    // Method to calculate the sum of integers (demonstrating unboxing)
    public static int calculateSum(ArrayList<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num;  // Unboxing (Integer → int)
        }
        return sum;
    }
```

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

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

        // Splitting input into string array and parsing into Integer list
        String[] strNumbers = input.split(" ");
        ArrayList<Integer> numbers = parseStringArray(strNumbers);

        // Calculating and displaying the sum
        int sum = calculateSum(numbers);
        System.out.println("Sum of the numbers: " + sum);

        sc.close();
    }
}
```

Output :



Learning Outcomes:

- Learnt how Java automatically converts between **primitive types** (e.g., int) and **wrapper classes** (e.g., Integer) using autoboxing and unboxing.
- Gained experience in converting user input, which is typically in **String format**, to an **Integer** using the Integer.parseInt() method.
- Learnt how to use **ArrayList** to store a dynamic collection of **Integer** objects and manipulate them efficiently.

Problem 2 :

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.

Objective:

- To understand the concept of serialization (converting an object to a byte stream) and deserialization (restoring an object from the byte stream) in Java.
- To deserialize the object from the file using ObjectInputStream and display the student details.
- To manage FileNotFoundException, IOException, and ClassNotFoundException using proper exception handling mechanisms to ensure smooth file handling operations.

Code:

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

public class StudentFileIO {
    public static void main(String[] args) {
        String filename = "students.txt"; // File to store student details
        Scanner sc = new Scanner(System.in);

        // Writing student details to a file
        try (FileWriter writer = new FileWriter(filename)) {
            System.out.print("Enter number of students: ");
            int n = sc.nextInt();
            sc.nextLine(); // Consume newline
            for (int i = 0; i < n; i++) {
                System.out.println("\nEnter details for Student " + (i + 1) + ":");
                System.out.print("Enter Student ID: ");
                int id = sc.nextInt();
                sc.nextLine(); // Consume newline

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

                System.out.print("Enter Student GPA: ");
                double gpa = sc.nextDouble();
                sc.nextLine(); // Consume newline
```

```
                // Writing to file
                writer.write(id + "," + name + "," + gpa + "\n");
            }
            System.out.println("\nStudent data saved successfully in 'students.txt'!");
        } catch (IOException e) {
            System.out.println("Error: Unable to write to file.");
            e.printStackTrace();
        }

        // Reading and displaying student details from the file
        System.out.println("\nReading student data from file...");
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] details = line.split(",");
                System.out.println("Student ID: " + details[0]);
                System.out.println("Name: " + details[1]);
                System.out.println("GPA: " + details[2]);
                System.out.println("----------------------");
            }
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found.");
        } catch (IOException e) {
            System.out.println("Error: Unable to read from file.");
            e.printStackTrace();
        }

        sc.close();
    }
}
```

Output:

Learning Outcomes:

- Learnt how to serialize and deserialize custom objects (e.g., Student objects), which allows for storing complex data structures.
- Learned to handle common file-related errors like file not found and class not found using exception handling.
- Gained experience in handling FileNotFoundException, IOException, and ClassNotFoundException to prevent crashes.

Problem 3:

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.

Objective:

- To create a menu-driven application that allows users to interact with the program by choosing different options (Add, Display, Exit).
- To understand how to read and write data to files, and how to store and retrieve employee details using file I/O operations.
- To retrieve and **display employee details** stored in the file when selected by the user.

Code:

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


class Employee1 {
    private int id;
    private String name;
    private String designation;
    private double salary;

    // Constructor
    public Employee1(int id, String name, String designation, double salary) {
```

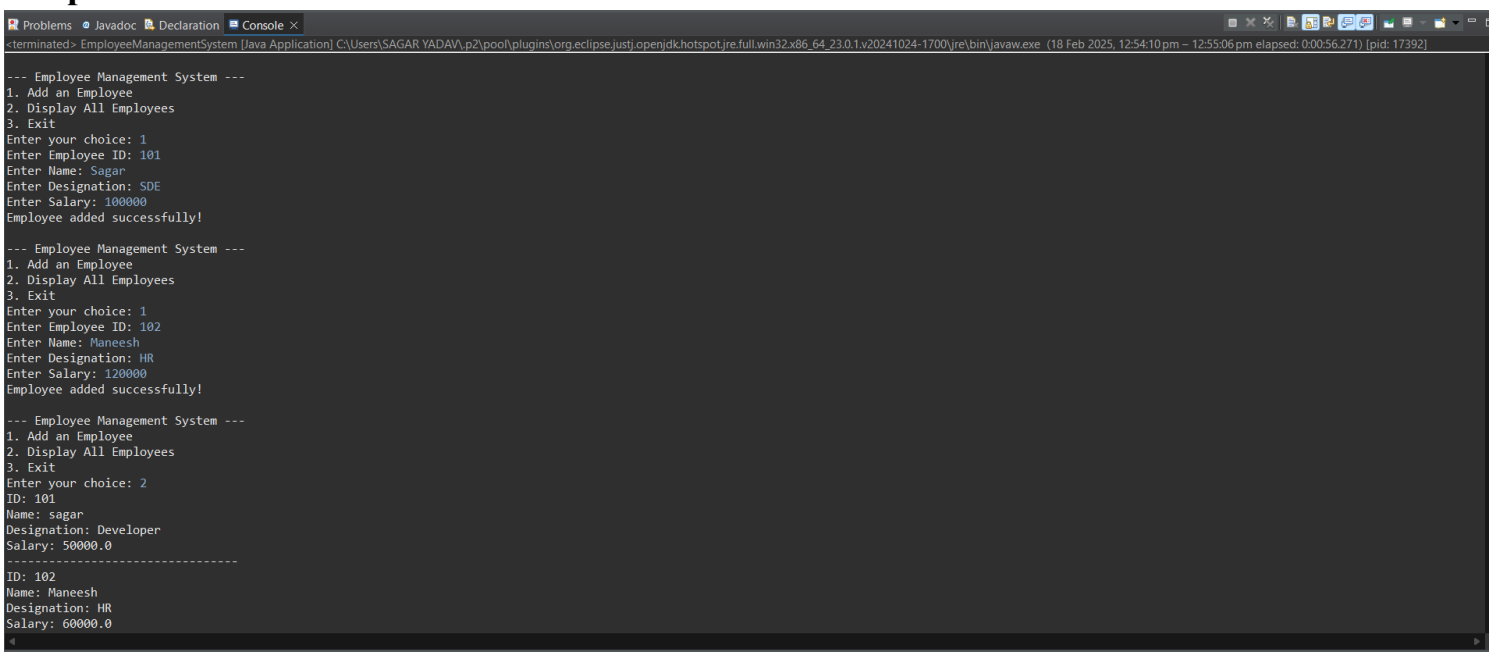
```
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    // Converts employee details into a string for file storage
    public String toFileFormat() {
        return id + "," + name + "," + designation + "," + salary;
    }

    // Displays employee details
    public void display() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Designation: " + designation);
        System.out.println("Salary: " + salary);
        System.out.println("---------------------------------");
    }

    // Parses an Employee object from a string (used for reading from file)
    public static Employee1 fromFileFormat(String line) {
        String[] details = line.split(",");
        int id = Integer.parseInt(details[0]);
        String name = details[1];
        String designation = details[2];
        double salary = Double.parseDouble(details[3]);
        return new Employee1(id, name, designation, salary);
    }
}

public class EmployeeManagementSystem {
    private static final String FILE_NAME = "employees.txt";
```

```
private static Scanner sc = new Scanner(System.in);

public static void main(String[] args) {
    while (true) {
        System.out.println("\n--- Employee 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 = sc.nextInt();
        sc.nextLine(); // Consume newline

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                System.out.println("Exiting... Goodbyeeeee!!!");
                return;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    }
}

// Method to add an employee and store it in the file
private static void addEmployee() {
    try (FileWriter writer = new FileWriter(FILE_NAME, true)) {
        System.out.print("Enter Employee ID: ");
        int id = sc.nextInt();
```

```
            sc.nextLine(); // Consume newline
            System.out.print("Enter Name: ");
            String name = sc.nextLine();
            System.out.print("Enter Designation: ");
            String designation = sc.nextLine();
            System.out.print("Enter Salary: ");
            double salary = sc.nextDouble();

            Employee1 emp = new Employee1(id, name, designation, salary);
            writer.write(emp.toFileFormat() + "\n"); // Append to file
            System.out.println("Employee added successfully!");
        } catch (IOException e) {
            System.out.println("Error: Unable to save employee data.");
        }
    }

    // Method to display all employees from the file
    private static void displayEmployees() {
        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME))) {
            String line;
            boolean hasEmployees = false;
            while ((line = reader.readLine()) != null) {
                Employee1 emp = Employee1.fromFileFormat(line);
                emp.display();
                hasEmployees = true;
            }
            if (!hasEmployees) {
                System.out.println("No employees found.");
            }
        } catch (FileNotFoundException e) {
            System.out.println("No employee records found. Add employees first.");
        } catch (IOException e) {
            System.out.println("Error: Unable to read employee data.");
```

```
    }
  }
}
```

Output:



Learning Outcomes:

- Learned how to gather and store employee details (name, ID, designation, salary) in a structured manner, using classes and collections.
- Gained practical knowledge of basic Create, Read, and Display operations in a program through user interaction and file handling.
- Learned how to build a menu-driven interface where users can interact with the program by choosing various options.