Experiment 5

Student Name: Maneesh Sonkaria
UID: 22BET10058
Branch: BE- IT
Section/Group: 22BET-701/A
Semester: 6th
Date of Performance: 18/02/25
Subject Name: JAVA WITH LAB
Subject Code: 22ITH-359

PROBLEM 1:

1.Aim: Write a Java program to calculate the sum of a list of integers autoboxing and unboxing, include methods to parse strings into their respective wrapper class.

2. Objective:

- To develop a program that processes user input, performs calculations, and displays results.
- To implement string-to-integer conversion using Integer.parseInt().
- To understand the concepts of autoboxing and unboxing in Java.

3.Implementation/Code:

```
package Experiments;
import java.util.*;

public class Exp5_1 {
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) { // Auto-unboxing occurs here
            sum += num;
        }
        return sum;
    }

    public static List<Integer> parseStringToIntegers(List<String> strNumbers) {
        List<Integer> numbers = new ArrayList<>();
        for (String str : strNumbers) {
```

```
            numbers.add(Integer.parseInt(str)); // Autoboxing
        }
        return numbers;
    }

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

        List<String> strNumbers = Arrays.asList(input.split(" "));
        List<Integer> numbers = parseStringToIntegers(strNumbers);

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

4.Output:

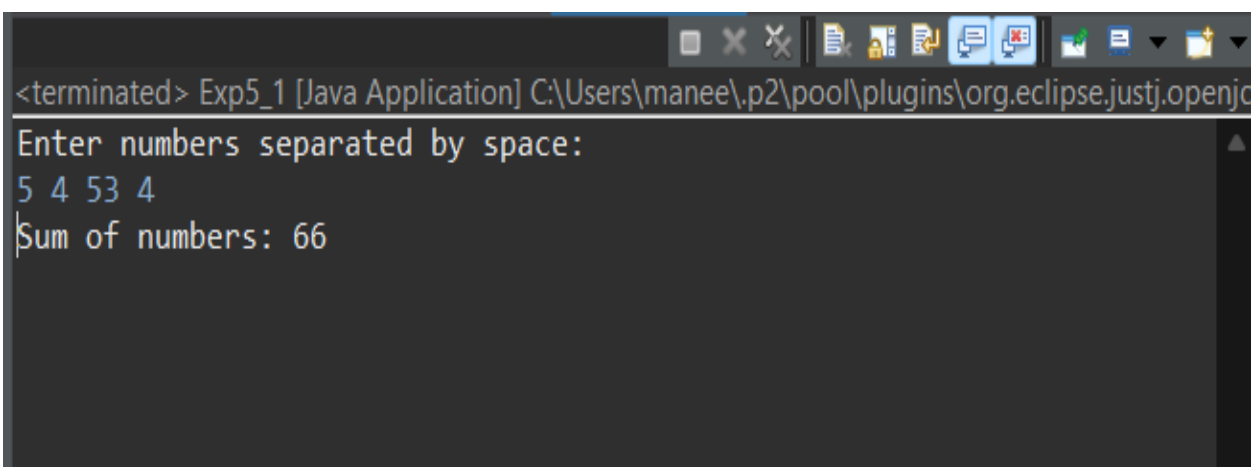

Figure 1: Output of code on ECLIPSE

5. Learning Outcomes:

a) Understood how autoboxing and unboxing work in Java when handling primitive and wrapper classes.

b) Implemented a Java program that reads, processes, and calculates the sum of user-inputted numbers.

c) Gained knowledge of using Scanner for input handling and Arrays.asList() for list conversion.

d) Developed an understanding of exception handling in integer parsing to prevent input errors.

PROBLEM 2:

1.Aim: To implement Java serialization and deserialization by creating a Student class, saving its object to a file, and retrieving it while ensuring data integrity.

2. Objective:

- To implement object persistence by writing and reading objects from a file.
- To understand the concept of Java serialization and deserialization using the Serializable interface.
- To develop error-handling mechanisms for file operations and object serialization.

3.Implementation/Code:

```
package Experiments;
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;
 }

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

public class Exp5_2 {
```

```
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 (FileNotFoundException e) {
        System.out.println("Error: File not found.");
    } catch (IOException e) {
        System.out.println("Error: Unable to serialize object.");
    }
}

public static void deserializeStudent() {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        Student student = (Student) ois.readObject();
        System.out.println("Deserialized Student Details:");
        student.display();
    } catch (FileNotFoundException e) {
        System.out.println("Error: File not found. Run serialization first.");
    } catch (IOException e) {
        System.out.println("Error: Unable to deserialize object.");
    } catch (ClassNotFoundException e) {
        System.out.println("Error: Student class not found.");
    }
}

public static void main(String[] args) {
    Student student = new Student(58, "Maneesh", 8.3);
    serializeStudent(student);
    deserializeStudent();
}
```

}

4.Ouput:

```
<terminated> Exp5_2 [Java Application] C:\Users\manee\.p2\pool\plugins\org.eclipse.justj.openjd
Student object serialized successfully.
Deserialized Student Details:
Student ID: 58
Student Name: Maneesh
Student GPA: 8.3
```

Figure 1: Output of code on ECLIPSE

5. Learning Outcomes:

a) Understood how serialization and deserialization work to store and retrieve Java objects.

b)Gained experience in handling IOException, FileNotFoundException, and ClassNotFoundException exceptions.

c) Developed a structured approach for object file handling using ObjectOutputStream and ObjectInputStream.

PROBLEM 3:

1.Aim: To create an Employee Management System in Java that allows users to add employee records and display them using file operations for persistent storage.

2. Objective:

- To understand how to store and retrieve employee records using file handling in Java.
- To implement a simple Employee Management System with options to add and display employees.
- To develop skills in handling user input and writing data persistently to a file.

3.Implementation/Code:

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

class Employee1 implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

    public Employee1(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 "ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " +
salary;
    }
}
```

```
public class Exp5_3 {
    private static final String FILE_NAME = "employees.txt";
    private static Scanner scanner = new Scanner(System.in);

    public static void addEmployee() {
        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 Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();

        Employee1 employee = new Employee1(id, name, designation, salary); // Changed to Employee1
        saveEmployeeToFile(employee);
        System.out.println("Employee added successfully!\n");
    }

    public static void saveEmployeeToFile(Employee1 employee) { // Changed to Employee1
        try (FileWriter fw = new FileWriter(FILE_NAME, true);
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter out = new PrintWriter(bw)) {

            out.println(employee);
        } catch (IOException e) {
            System.out.println("Error: Unable to save employee.");
        }
    }
```

```
public static void displayAllEmployees() {
    try (BufferedReader br = new BufferedReader(new FileReader(FILE_NAME))) {
        String line;
        System.out.println("\nEmployee Details:");
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        System.out.println();
    } catch (FileNotFoundException e) {
        System.out.println("No employee records found.\n");
    } catch (IOException e) {
        System.out.println("Error reading employee records.\n");
    }
}

public static void main(String[] args) {
    while (true) {
        System.out.println("1. Add an Employee");
        System.out.println("2. Display All Employees");
        System.out.println("3. Exit");
        System.out.print("Choose an option: ");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayAllEmployees();
                break;
            case 3:
                System.out.println("Exiting...");
                return;
```

```
            default:
                System.out.println("Invalid choice! Try again.\n");
        }
    }
  }
}
```

4.Output:

```
<terminated> Exp5_3 [Java Application] C:\Users\manee\.p2\pool\plugins\org.eclipse.justj.oper
1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 58
Enter Employee Name: Maneesh
Enter Designation: SDE
Enter Salary: 150000
Employee added successfully!

1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 7
Enter Employee Name: Sagar
Enter Designation: SDE
Enter Salary: 150000
Employee added successfully!

1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 48
Enter Employee Name: Manav Thakur
Enter Designation: Senior SDE
Enter Salary: 200000
Employee added successfully!

1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 2

Employee Details:
ID: 58, Name: Maneesh, Designation: SDE, Salary: 150000.0
ID: 7, Name: Sagar, Designation: SDE, Salary: 150000.0
ID: 48, Name: Manav Thakur, Designation: Senior SDE, Salary: 200000.0

1. Add an Employee
2. Display All Employees
3. Exit
Choose an option: 3
```

Figure 1: Output of code on ECLIPSE

5. Learning Outcomes:

a) Understood how to use file handling (FileWriter, BufferedReader, PrintWriter) for storing and retrieving data.

b) Implemented a menu-driven program to manage employee records dynamically.

c) Gained experience in handling exceptions like IOException and FileNotFoundException.