Experiment 5

Student Name: Sikander Singh Nanglu **UID: 22BET10031**
Branch: BE-IT **Section/Group: 22BET-IOT-701/A**
Semester: 6 **Date of Performance: 18/02/2025**
Subject Name: Java with Lab **Subject Code: 22ITH-359**

Problem No. 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:

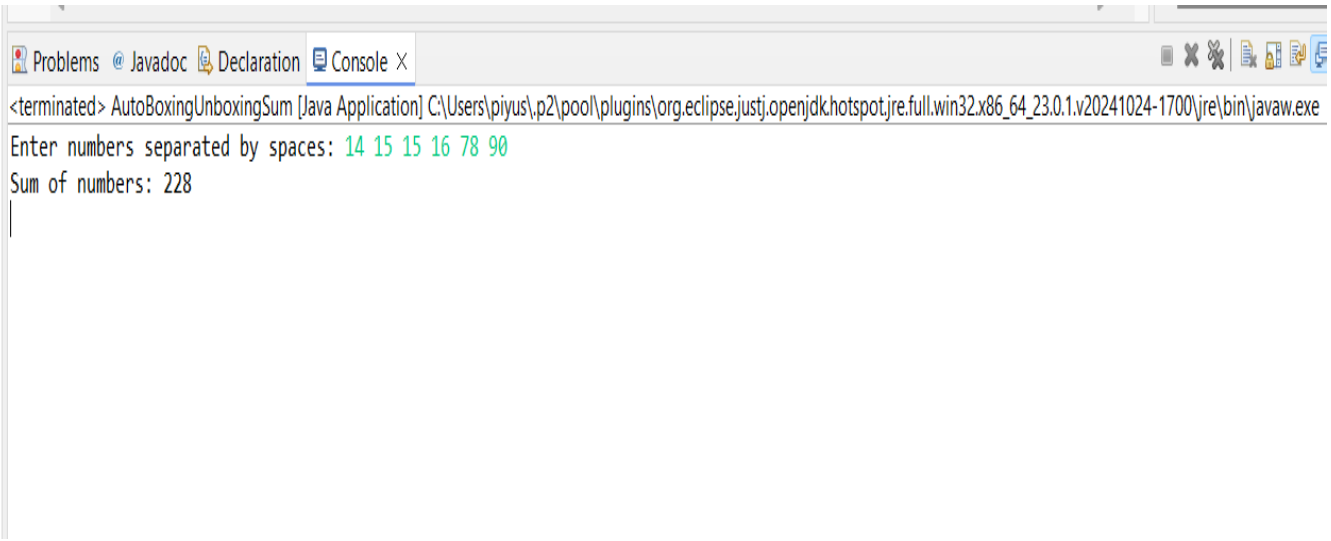
- To understand the concept of autoboxing and unboxing .
- To understand about wrapper classes.

Implementation/Code:

```
import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class AutoBoxingUnboxingSum {

public static List<Integer> parseStringsToIntegers(String[] strNumbers) {

List<Integer> intList = new ArrayList<>();

for (String str : strNumbers) {

intList.add(Integer.parseInt(str.trim()));

}

return intList;

}

public static int calculateSum(List<Integer> numbers) {

int sum = 0;

for (Integer num : numbers) {
```

```
sum += num;
}
return sum;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter numbers separated by spaces: ");
String input = scanner.nextLine();
String[] strNumbers = input.split("\\s+");
List<Integer> numbers = parseStringsToIntegers(strNumbers);
int sum = calculateSum(numbers);
System.out.println("Sum of numbers: " + sum);
scanner.close();
}
}
```

Output:



Learning Outcome:

a) Learners will understand how Java automatically converts primitive int to Integer (autoboxing) and Integer to int (unboxing).

b) Learn how to use Integer.parseInt() to convert string inputs into integer values.

c) Learn how to store and manipulate a collection of integers using ArrayList<Integer>.

d) Learn how to split a string into parts using split("\\s+") and remove unwanted spaces using .trim().

e) Gain experience in taking user input using Scanner and processing it efficiently.

Problem No. 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. HandleFileNotFoundException, IOException, and ClassNotFoundException using exception handling.

Objective:

- To understand the concept of Exception handling.
- To understand the concept of serialize and deserialize.

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

```
}
}
public class StudentSerialization {
public static void serializeStudent(Student student, String filename) {
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
oos.writeObject(student);
System.out.println("Serialized successfully!");
} catch (FileNotFoundException e) {
System.out.println("Error: File not found.");
} catch (IOException e) {
System.out.println("Error: Unable to write to file.");
}
}
public static Student deserializeStudent(String filename) {
Student student = null;
try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename))) {
student = (Student) ois.readObject();
System.out.println("Deserialized successfully!");
} catch (FileNotFoundException e) {
System.out.println("Error: File not found.");
} catch (IOException e) {
System.out.println("Error: Unable to read from file.");
} catch (ClassNotFoundException e) {
System.out.println("Error: Class not found.");
}
return student;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
 String filename = "student.ser";
 while (true) {
 System.out.println("\n1. Serialize Student");
 System.out.println("2. Deserialize Student");
 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 Student ID: ");
   int id = scanner.nextInt();
```

```
scanner.nextLine();
System.out.print("Enter Student Name: ");
String name = scanner.nextLine();
System.out.print("Enter GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(id, name, gpa);
serializeStudent(student, filename);
break;
case 2:
Student deserializedStudent = deserializeStudent(filename);
if (deserializedStudent != null) {
deserializedStudent.display();
}
break;
case 3:
System.out.println("Exiting program...");
scanner.close();
return;
default:
System.out.println("Invalid choice! Please enter 1, 2, or 3.");
}
}
}
}
```

Output:

```
<terminated> StudentSerialization [Java Application] C:\Users\piyus\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.1.v20241024-1700\jre
1. Serialize Student
2. Deserialize Student
3. Exit
Choose an option: 1
Enter Student ID: 10031
Enter Student Name: Sikander Singh
Enter GPA: 7.5
Serialized successfully!

1. Serialize Student
2. Deserialize Student
3. Exit
Choose an option: 2
Deserialized successfully!
Student ID: 10031
Name: Sikander Singh
GPA: 7.5

1. Serialize Student
2. Deserialize Student
3. Exit
Choose an option: 3
Exiting program...
```

Learning Outcomes:

- Learn how to save and retrieve Java objects using ObjectOutputStream and ObjectInputStream.
- Understand how to read from and write to files using FileOutputStream and FileInputStream.
- Understand how to use Scanner to read different types of user input like integers, strings, and doubles.
- Understand why Java requires objects to implement Serializable for object persistence.
- Learn how Java handles int, String, and double data types in serialization and deserialization.

Problem No. 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 understand the concept of scanner classes.
- To understand the concept of switch case and inheritance etc.

Implementation/Code:

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
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 display() {
System.out.println("Employee ID: " + id);
System.out.println("Name: " + name);
System.out.println("Designation: " + designation);
System.out.println("Salary: " + salary);
System.out.println("------------------------");
}
}
public class EmployeeManagement {
private static final String FILE_NAME = "employees.dat";
public static void addEmployee(Employee emp) {
List<Employee> employees = readEmployees();
employees.add(emp);
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
oos.writeObject(employees);
System.out.println("Employee added successfully!");
} catch (IOException e) {
System.out.println("Error: Unable to save employee details.");
}
}
public static List<Employee> readEmployees() {
List<Employee> employees = new ArrayList<>();
try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
employees = (List<Employee>) ois.readObject();
} catch (FileNotFoundException e) {
} catch (IOException | ClassNotFoundException e) {
System.out.println("Error: Unable to read employee details.");
}
return employees;
}
public static void displayEmployees() {
List<Employee> employees = readEmployees();
if (employees.isEmpty()) {
System.out.println("No employees found.");
} else {
```

```
System.out.println("\nEmployee List:");
System.out.println("------------------------");
for (Employee emp : employees) {
emp.display();
}
}
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
while (true) {
System.out.println("\n1. 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();
Employee newEmployee = new Employee(id, name, designation, salary);
addEmployee(newEmployee);
break;
case 2:
displayEmployees();
break;
case 3:
System.out.println("Exiting the application...");
scanner.close();
return;
default:
System.out.println("Invalid choice! Please enter 1, 2, or 3.");
}
}
}
```

```
}
```

Output:

```
<terminated> EmployeeManagement [Java Application] C:\Users\piyus\.p2\pool\plugins\org.eclipse.ju

1. Add Employee
2. Display All Employees
3. Exit
Choose an option: 1
Enter Employee ID: 10031
Enter Employee Name: Sikander Singh
Enter Designation: Software Engineer
Enter Salary: 100000
Employee added successfully!

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

Employee List:
------------------------
Employee ID: 10031
Name: Sikander Singh
Designation: Software Engineer
Salary: 100000.0
------------------------

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

Learning Outcomes:

- Learn how to store and retrieve Java objects using ObjectOutputStream and ObjectInputStream.
- Learn how to create an interactive program using a switch-case menu for better user experience.
- Understand how to create and manage a class (Employee) with attributes and methods.
- Learn to handle FileNotFoundException, IOException, and ClassNotFoundException to prevent crashes.
- Understand how to store multiple employee objects in a list and process them efficiently.