Experiment 5

Student Name: Khushi **UID: 22BCS15811**
Branch: BE-CSE **Section/Group: 618_B**
Semester: 6 **Date of Performance:21/2/25**
Subject Name:PBLJ **Subject Code: 22CSH-359**

Problem 1: 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()).

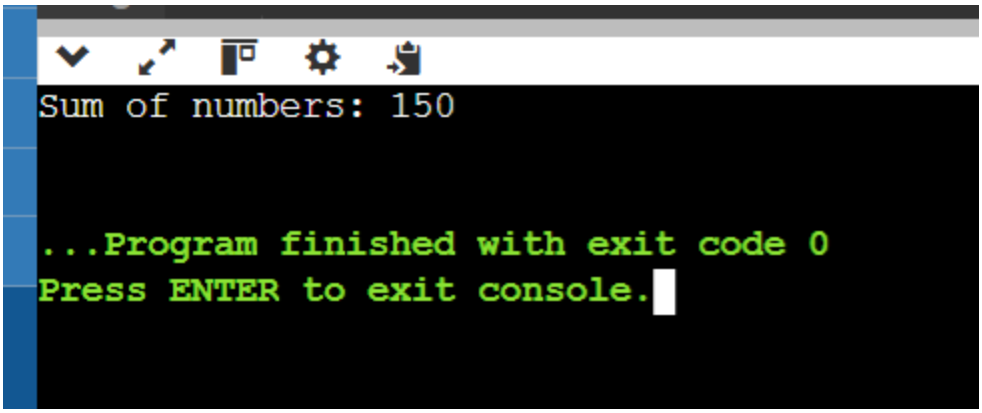
1. **Objective:** To demonstrate the concepts of autoboxing, unboxing, serialization, and deserialization in Java. The first part of the program converts string representations of numbers into Integer objects using Integer.parseInt(), performs arithmetic operations through autoboxing and unboxing, and calculates their sum.

2. **Implementation/Code:**

```
import java.util.ArrayList;
import java.util.List;

public class AutoBoxingUnboxing {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>();
        String[] numStrings = {"10", "20", "30", "40", "50"};


        for (String num : numStrings) {
            numbers.add(Integer.parseInt(num));
        }
```

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

    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Unboxing occurs here
        }
        return sum;
    }
}
```

OUTPUT:



Problem 2: Create a Java program to serialize and deserialize a Student object.

Objective: The objective of this Java program is to demonstrate **serialization and deserialization** of a Student object using Java's Serializable interface. It saves the object to a file using ObjectOutputStream and later retrieves it using ObjectInputStream

Code:

```
import java.io.*;


class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    String name;
    int age;
    String course;


    public Student(String name, int age, String course) {
        this.name = name;
        this.age = age;
        this.course = course;
    }
    public void display() {
        System.out.println("Name: " + name);
```

```
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
    }
}
public class StudentSerialization {
    public static void main(String[] args) {
        String filename = "student.ser";

        Student student1 = new Student("Khushi", 21, "Cybersecurity");
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(student1);
            System.out.println("Student object serialized successfully!");
        } catch (IOException e) {
            e.printStackTrace();
        }


        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filename))) {
            Student deserializedStudent = (Student) ois.readObject();
            System.out.println("\nDeserialized Student Object:");
            deserializedStudent.display();
        } catch (IOException | ClassNotFoundException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

```
Student object serialized successfully!

Deserialized Student Object:
Name: Khushi
Age: 21
Course: Cybersecurity


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

Learning Outcomes:

- Learn how to convert an object into a byte stream and retrieve it later while preserving its state.
- Gain hands-on experience using ObjectOutputStream and ObjectInputStream for storing and reading objects from files.
- Learn how serialization helps in saving object data for future use, making it useful in real-world applications like caching and data storage.

Problem 3: 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 : This Java application is to manage employee records using a menu-driven approach. It allows users to add employee details, store them in a file, and retrieve them when needed. The program ensures efficient data handling and provides a simple interface for employee management.

Code:

```
import java.io.*;

import java.util.*;


class Employee {

    private String empId;

    private String name;

    private String designation;

    private double salary;


    public Employee(String empId, String name, String designation, double salary) {

        this.empId = empId;

        this.name = name;
```

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

    @Override
    public String toString() {
        return empId + ", " + name + ", " + designation + ", " + salary;
    }
}

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

    public static void addEmployee() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee ID: ");
        String empId = 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 employee = new Employee(empId, name, designation, salary);
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME,
true))) {
            writer.write(employee.toString());
            writer.newLine();
            System.out.println("Employee added successfully!\n");
        } catch (IOException e) {
            System.out.println("Error saving employee data: " + e.getMessage());
        }
    }

    public static void displayEmployees() {
        try (BufferedReader reader = new BufferedReader(new
FileReader(FILE_NAME))) {
            String line;
            System.out.println("\nEmployee Details:");
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading employee data: " + e.getMessage());
        }
```

```
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("\nMenu:");
        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 = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                System.out.println("Exiting application.");
                scanner.close();
```

```
                System.exit(0);
            default:
                System.out.println("Invalid choice!.");
        }
    }
  }
}
```

Output:

```
input
Enter Employee Name: Khushi
Enter Designation: developer
Enter Salary: 200000
Employee added successfully!


Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 145
Enter Employee Name: Alice
Enter Designation: Teacher
Enter Salary: 12000
Employee added successfully!


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

Employee Details:
123, Khushi, developer, 200000.0
145, Alice, Teacher, 12000.0

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

Learning Outcomes:

- Understanding how to read and write data to files using BufferedWriter and BufferedReader.
- Implementing a user-friendly menu system using loops and conditional statements.
- Managing file-related errors using try-catch blocks to ensure program stability.