Experiment 5

Student Name: Aaryan Sharma   **UID:23BCS80058**
Branch: CSE   **Section/Group:903-A**
Semester: 6   **Date of Performance:3-03-25**
Subject Name: Java   **Subject Code: 22CSH-354**

1.Aim: Solve problems related to autoboxing , wrapper classes. 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()).

2.Objective: To solve the problems of autoboxing and corresponding wrapper classes.

3.Algorithm:
Step 1: Initialize Data
  1. Create a list to store integer values (List<Integer>).
  2. Define an array of strings representing integer numbers.
Step 2: Parse Strings to Integers (Autoboxing)
  3. Iterate through the string array.
  4. Convert each string to an Integer using Integer.parseInt().
  5. Add the parsed Integer to the list (Autoboxing happens here).
Step 3: Calculate Sum (Unboxing)
  6. Initialize a variable sum to store the total sum.
  7. Iterate through the list of Integer objects.
  8. Add each Integer value to sum (Unboxing happens here).
Step 4: Display Result
  9. Print the final sum.

4.Implementation/Code:

```java
import java.util.ArrayList;
import java.util.List;
public class AutoboxingUnboxingExample {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>();
        // Parsing strings to Integer objects (Autoboxing)
        String[] strNumbers = {"10", "20", "30", "40", "50"};
        for (String str : strNumbers) {
            numbers.add(parseStringToInteger(str)); // Autoboxing
        }
```
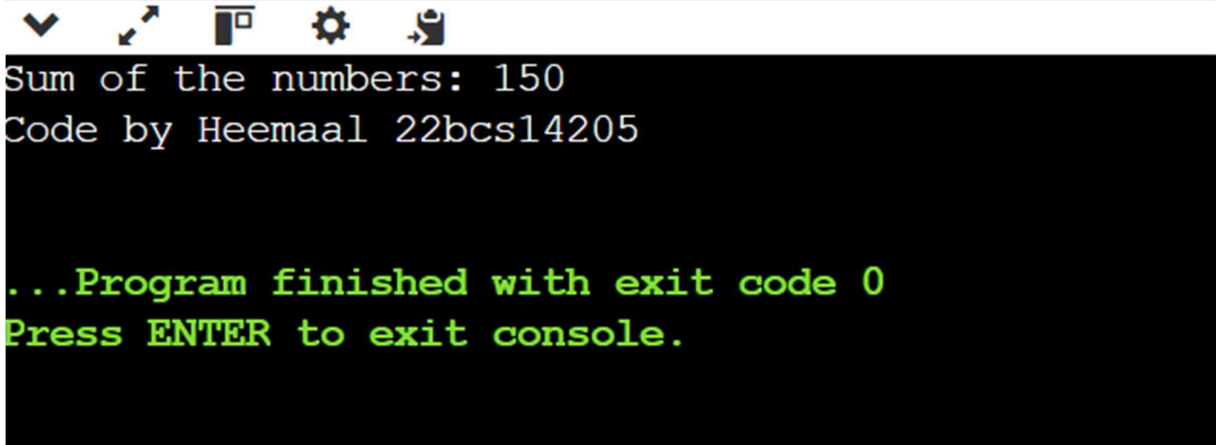
```java
        // Calculating the sum (Unboxing occurs here)
        int sum = calculateSum(numbers);
        System.out.println("Sum of the numbers: " + sum);
    }
    // Method to parse a string into Integer
    public static Integer parseStringToInteger(String str) {
        return Integer.parseInt(str); // Returns an Integer object (Autoboxing)
    }
    // Method to calculate sum
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Unboxing happens here
        }
        return sum;
    }
}
```

5. **Output:**


```
Sum of the numbers: 150
Code by Heemaal 22bcs14205


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

Problem-2:Serialization and deserialization

1.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.

2.Objective: performing object serialization and deserialization over entities.

3.Algorithm: Step 1: Define the Student Class

Create a Student class.

Implement the Serializable interface to enable object serialization.

Define attributes: id, name, and gpa.

Create a constructor to initialize values.

Implement a display() method to print student details.

Step 2: Serialization Process

Create a method serializeStudent(Student student):

Use FileOutputStream to create/open a file (student.ser).

Use ObjectOutputStream to write the Student object to the file.

Handle exceptions:

FileNotFoundException: If the file is not found.

IOException: If an input/output error occurs.

Step 3: Deserialization Process

Create a method deserializeStudent():

Use FileInputStream to open the file (student.ser).

Use ObjectInputStream to read the Student object from the file.

Handle exceptions:

FileNotFoundException: If the file does not exist.

IOException: If an input/output error occurs.

ClassNotFoundException: If the class definition is missing.

Step 4: Main Method Execution

Inside the main() method:

Create a Student object with sample data.

Call serializeStudent(student) to save the object.

Call deserializeStudent() to read the object from the file.

If deserialization is successful, call display() to print the student details.

4.Implementation code :

```java
import java.io.*;
// Step 1: Create a Student class implementing Serializable
class Student implements Serializable {
    private static final long serialVersionUID = 1L; // Recommended for serialization
    private int id;
    private String name;
    private double gpa;

    // Constructor
    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

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

// Step 2: Create a class to handle serialization and deserialization
public class StudentSerializationDemo {
    private static final String FILE_NAME = "student.ser";

    public static void main(String[] args) {
        // Creating a Student object
        Student student = new Student(101, "John Doe", 3.8);

        // Serialize the Student object
        serializeStudent(student);

        // Deserialize the Student object
        Student deserializedStudent = deserializeStudent();

        // Display the Student details
        if (deserializedStudent != null) {
            System.out.println("\nDeserialized Student Details:");
```

```java
            deserializedStudent.display();
        }
    }

    // Method to serialize a Student object
    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: IOException occurred while serializing.");
        }
    }

    // Method to deserialize a Student object
    public static Student deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            return (Student) ois.readObject();
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found! Cannot deserialize.");
        } catch (IOException e) {
            System.out.println("Error: IOException occurred while deserializing.");
        } catch (ClassNotFoundException e) {
            System.out.println("Error: Class not found!");
        }
        return null;
    }
}
```

5. **Output:**

```
Student object serialized successfully.

Deserialized Student Details:
Student ID: 101
Student Name: John Doe
Student GPA: 3.8
```

Problem 3: Menu Driven Program development

1.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.

2.Objective: Creating menu driven application as per the specified question.

3.Algorithm: **Step 1: Define Employee Class**
1. Create an Employee class.
2. Implement the Serializable interface for object storage.
3. Define attributes: empId, name, designation, and salary.
4. Create a constructor to initialize values.
5. Implement a display() method to print employee details.

Step 2: Main Menu Execution
6. Inside the main() method:
   - Display menu options:
     - 1. Add an Employee
     - 2. Display All Employees
     - 3. Exit
   - Read user choice and call the corresponding method.
   - Repeat menu until the user chooses to exit.

Step 3: Add Employee
7. If option 1 is selected:
   - Prompt user input for empId, name, designation, and salary.
   - Create a new Employee object.
   - Read existing employees from the file (if any).
   - Append the new employee to the list.
   - Write the updated list back to the file using serialization.
   - Print "Employee added successfully!".

Step 4: Display All Employees
8. If option 2 is selected:
   - Read employee list from the file using deserialization.
   - If list is empty, print "No employee records found."
   - Else, iterate through the list and call display() for each employee.

Step 5: File Handling Functions

9. Read Employees (readEmployees()):
   - Try reading employees from the file.
   - If the file does not exist, return an empty list.
   - Catch and handle FileNotFoundException, IOException, and ClassNotFoundException.
10. Write Employees (writeEmployees()):
- Open the file and write the employee list using serialization.
- Catch and handle IOException.

Step 6: Exit Application

11. If option 3 is selected, print "Exiting the application..." and terminate the program.

4.Implementation code :

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

// Step 1: Define Employee class implementing Serializable
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int empId;
    private String name;
    private String designation;
    private double salary;

    // Constructor
    public Employee(int empId, String name, String designation, double salary) {
        this.empId = empId;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    // Method to display employee details
    public void display() {
        System.out.println("\nEmployee ID: " + empId);
        System.out.println("Employee Name: " + name);
        System.out.println("Designation: " + designation);
        System.out.println("Salary: " + salary);
```

```java
        }
    }
}

// Step 2: Define Main Application Class
public class EmployeeManagementSystem {
    private static final String FILE_NAME = "employees.dat";
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            // Display menu
            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(); // Consume newline

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayAllEmployees();
                    break;
                case 3:
                    System.out.println("Exiting the application...");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }

    // Step 3: Method to add employee details
    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int empId = scanner.nextInt();
        scanner.nextLine(); // Consume newline
```

```java
        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();
        scanner.nextLine(); // Consume newline

        // Create an Employee object
        Employee employee = new Employee(empId, name, designation, salary);

        // Save to file
        List<Employee> employees = readEmployees();
        employees.add(employee);
        writeEmployees(employees);

        System.out.println("Employee added successfully!");
    }

    // Step 4: Method to display all employees
    public static void displayAllEmployees() {
        List<Employee> employees = readEmployees();
        if (employees.isEmpty()) {
            System.out.println("No employee records found.");
        } else {
            System.out.println("\nEmployee Details:");
            for (Employee emp : employees) {
                emp.display();
            }
        }
    }

    // Step 5: Read employees from file
    private 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) {
```

```
            // File doesn't exist yet, return an empty list
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error reading employee data.");
        }
        return employees;
    }

    // Step 6: Write employees to file
    private static void writeEmployees(List<Employee> employees) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
        } catch (IOException e) {
            System.out.println("Error writing employee data.");
        }
    }
}
```

5.Output:

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

Employee Details:

Employee ID: 1001
Employee Name: heemaal
Designation: Product manager
Salary: 2000000.0
```

6.Learning Outcomes:
- Learned about implementation of java programs
- Learned about wrapper classes and their implementation.
- Learned about autoboxing.
- Learned about menu driven program and their built in cases.