**Experiment 5**

**Student Name: SHREYA UID: 22BET10172**

**Branch: IT Section/Group: 22BET\_IOT-702/A**

**Semester: 6th Date of Performance:18/02/25**

**Subject Name: JAVA Subject Code: 22ITH-359**

**Problem 1**

1. **Aim:** To develop a program that demonstrates the use of wrapper classes with a focus on autoboxing and unboxing by calculating the sum of a list of integers.
2. **Objective:**

* To demonstrate the use of wrapper classes like Integer in Java.
* To implement autoboxing and unboxing for seamless conversion between primitive types and their wrapper objects.
* To parse string inputs into integers using Integer.parseInt() for accurate data processing.
* To calculate the sum of a list of integers efficiently.

1. **Code:**

package mathoperations;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class SumUsingAutoboxing {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.println("Enter integers (type 'done' to finish):");

while (true) {

String input = scanner.nextLine();

if (input.equalsIgnoreCase("done")) break;

try {

numbers.add(Integer.parseInt(input));

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter an integer or 'done'.");

}

}

System.out.println("The sum of the entered integers is: " + calculateSum(numbers));

scanner.close();

}

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

int sum = 0;

for (int num : numbers) sum += num;

return sum;

}

}

1. **Output:**

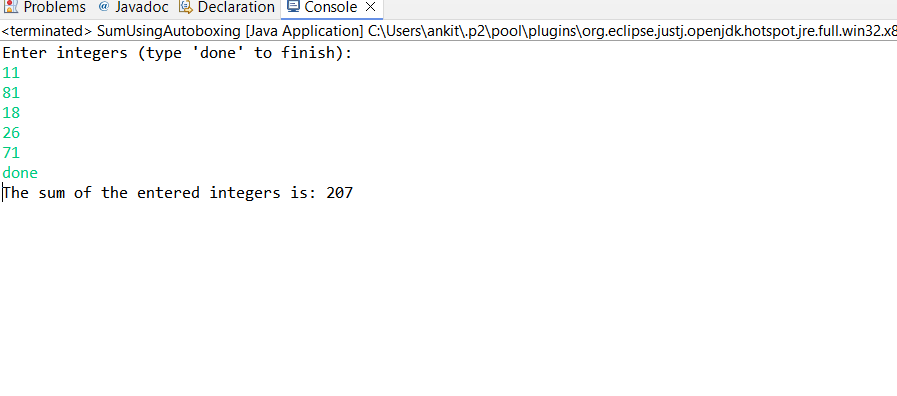


Fig 1. SUM OF INTEGER

1. **Learning Outcomes:**

* Understand the concept and usage of wrapper classes in Java.
* Implement autoboxing and unboxing for automatic type conversion between primitives and objects.
* Parse string inputs into numeric values using methods like Integer.parseInt().
* Develop skills in handling user input and managing exceptions effectively.
* Perform basic data processing tasks, such as calculating the sum of integers from user input.

**Problem 2**

**Aim:**

To develop a Java program that demonstrates **serialization** and **deserialization** of a Student object by saving its data (ID, name, and GPA) to a file and retrieving it later.

**Objective:**

* To implement serialization of a Student object and save it to a file.
* To perform deserialization and retrieve the object's data from the file.
* To handle file-related exceptions using proper exception handling mechanisms.
* To demonstrate persistent storage and retrieval of object data in Java.

**Code:**

package school;

import java.io.\*;

import java.util.Scanner;

class BeStudent implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

private double gpa;

public BeStudent(int id, String name, double gpa) {

this.id = id;

this.name = name;

this.gpa = gpa;

}

public void display() {

System.out.printf("\n--- Student Details ---\nID: %d\nName: %s\nGPA: %.2f\n", id, name, gpa);

}

}

class StudentFileManager {

private static final String FILE = "student.ser";

public static void save(BeStudent student) {

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE))) {

out.writeObject(student);

System.out.println("\n✅ Student saved.");

} catch (IOException e) {

System.out.println("❌ Save error: " + e.getMessage());

}

}

public static BeStudent load() {

if (!new File(FILE).exists()) return null;

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE))) {

return (BeStudent) in.readObject();

} catch (IOException | ClassNotFoundException e) {

System.out.println("❌ Load error: " + e.getMessage());

}

return null;

}

}

public class StudentSerialization {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("\n1. Save Student 2. Load Student 3. Exit\nChoice: ");

switch (sc.nextInt()) {

case 1 -> {

System.out.print("ID: "); int id = sc.nextInt();

sc.nextLine(); // Consume newline

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

System.out.print("GPA: "); double gpa = sc.nextDouble();

StudentFileManager.save(new BeStudent(id, name, gpa));

}

case 2 -> {

BeStudent s = StudentFileManager.load();

if (s != null) s.display();

else System.out.println("❌ No data found.");

}

case 3 -> {

System.out.println("Chalnikal!"); sc.close(); return;

}

default -> System.out.println("❌ Invalid choice.");

}

}

}

**Output:**

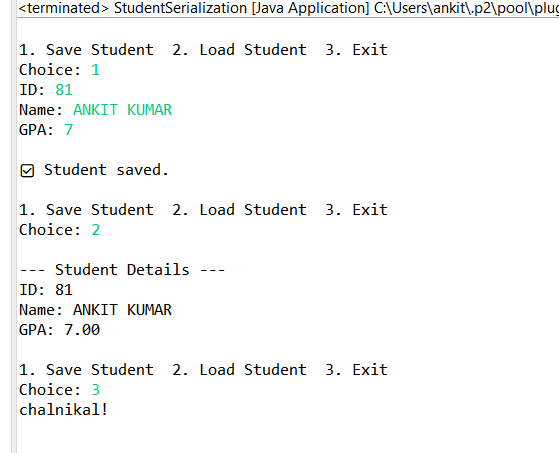
****

Fig 2.1. Student details

1. **Learning Outcomes:**

* Understanding the concepts of serialization and deserialization in Java.
* Learning to save and retrieve object data from files using streams.
* Implement exception handling for common file I/O errors.
* Developing the ability to manage persistent object data efficiently.
* Gain hands-on experience with Java’s ObjectOutputStream and ObjectInputStream classes.

**Problem 3**

1. **Aim:**

To develop a menu-based Java application for managing employee records, including adding new employees, displaying all employee details, and storing data persistently using file handling.

1. **Objective:**

* To gather and store employee details (name, ID, designation, salary) in a file using file handling.
* To display all stored employee records from the file.
* To implement exception handling for robust file operations.
* To develop a user-friendly and efficient console-based interface for employee management..

1. **Code:**

package employees;

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.printf("\nID: %d | Name: %s | Designation: %s | Salary: $%.2f\n", id, name, designation, salary);

}

}

class EmployeeFileManager {

private static final String FILE = "employees.dat";

public static void save(List<Employee> employees) {

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE))) {

out.writeObject(employees);

System.out.println("\n✅ Employee data saved!");

} catch (IOException e) {

System.out.println("❌ Save error: " + e.getMessage());

}

}

@SuppressWarnings("unchecked")

public static List<Employee> load() {

File file = new File(FILE);

if (!file.exists()) return new ArrayList<>();

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE))) {

return (List<Employee>) in.readObject();

} catch (IOException | ClassNotFoundException e) {

System.out.println("❌ Load error: " + e.getMessage());

}

return new ArrayList<>();

}

}

public class EmployeeManagementSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

List<Employee> employees = EmployeeFileManager.load();

while (true) {

System.out.print("\n1. Add Employee 2. Display All 3. Exit\nChoice: ");

switch (sc.nextInt()) {

case 1 -> {

System.out.print("ID: "); int id = sc.nextInt();

sc.nextLine();

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

System.out.print("Designation: "); String designation = sc.nextLine();

System.out.print("Salary: "); double salary = sc.nextDouble();

employees.add(new Employee(id, name, designation, salary));

EmployeeFileManager.save(employees);

}

case 2 -> {

if (employees.isEmpty()) System.out.println("\nNo employees found!");

else employees.forEach(Employee::display);

}

case 3 -> {

System.out.println("\nGoodbye!"); sc.close(); return;

}

default -> System.out.println("❌ Invalid choice.");

}

}

}

}

1. **Output:**

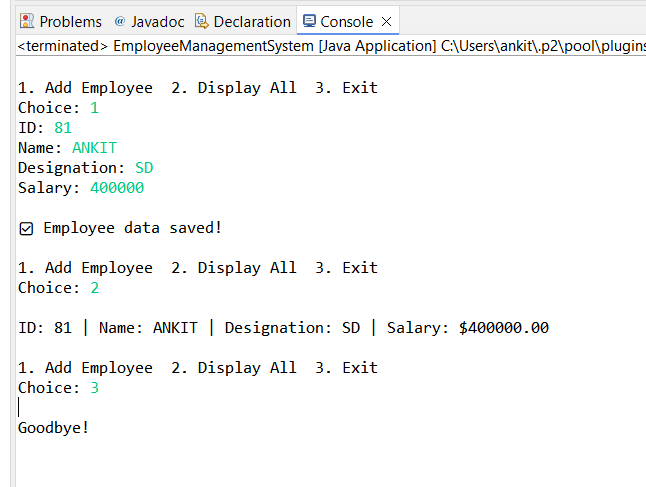
****

Fig 3.1. Employee Management System

1. **Learning Outcomes:**

* Implement file handling operations (read/write) in Java.
* Store and retrieve employee data from a file using object serialization.
* Develop a menu-driven application for managing employee data.
* Apply exception handling techniques to manage runtime errors effectively.
* Develop modular Java code using classes and methods for better maintainability.