# Experiment 5

**Student Name: Shivansh Singh** **UID: 22BET10105**

**Branch: IT** **Section/Group: 22BET_IOT-701/A**

**Semester: 6th** **Date of Performance:18.02.25**

**Subject Name: PBLJ Lab** **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 showcase the usage of wrapper classes such as Integer in Java.
- To implement autoboxing and unboxing for smooth conversion between primitive data types and their corresponding wrapper objects.
- To convert string inputs into integers using Integer.parseInt() for precise data handling.
- To efficiently compute the sum of a collection of integers.

3. **Code:**

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class IntegerSumCalculator {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      List<Integer> numberList = new ArrayList<>(); // List to store numbers (Autoboxing
occurs here)
      System.out.println("Enter integers (type 'done' to stop):");
      while (true) {
         String userInput = scanner.nextLine();
         if (userInput.equalsIgnoreCase("done")) {
            break;
         }
         try {
            int num = Integer.parseInt(userInput); // Convert string input to an integer
            numberList.add(num); // Autoboxing: int to Integer
```

```
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid integer or type 'done' to exit.");
            }
        }
        int totalSum = computeSum(numberList); // Unboxing occurs automatically
        System.out.println("Total sum of entered numbers: " + totalSum);
    }
    // Method to compute the sum of integers (Unboxing occurs here)
    public static int computeSum(List<Integer> numberList) {
        int total = 0;
        for (Integer num : numberList) {
            total += num; // Unboxing: Integer to int
        }
        return total;
    }
}
```

## 4. Output:

```
<terminated> SumUsingAutoboxing [Java Application] C:\Users\ASUS\Oracle_JDK-22\bin\javaw.exe  (24 Feb 2025, 8:29:29 pm – 8:29:48 pm elap
Enter integers (type 'done' to finish):
11
21
13
25
60
DONE
The sum of the entered integers is: 130
```

Fig 1. SUM OF INTEGER

## 5. Learning Outcomes:

- Develop a solid understanding of wrapper classes and their significance in Java.
- Utilize autoboxing and unboxing for effortless conversion between primitive data types and their corresponding wrapper objects.
- Transform string inputs into numerical values using methods like Integer.parseInt().
- Improve skills in processing user input and handling exceptions effectively.
- Perform essential data processing tasks, such as calculating the sum of user-entered integers.

# Problem 2

## 1. 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.

## 2. Objective:

- Implement serialization to store a Student object in a file.
- Perform deserialization to retrieve the object's data from the file.
- Manage file-related exceptions using appropriate exception handling techniques.
- Showcase persistent storage and retrieval of object data in Java.

## 3. Code:

```
import java.io.*;
import java.util.Scanner;
// Student class implementing Serializable interface
class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    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 displayDetails() {
        System.out.println("\n--- Student Details ---");
        System.out.println("ID   : " + id);
        System.out.println("Name : " + name);
        System.out.println("GPA  : " + gpa);
    }
}
// Handles serialization and deserialization of Student objects
```

```
class StudentDataManager {
    private static final String FILE_NAME = "student_data.ser";
    // Serialize a Student object and save it to a file
    public static void saveStudent(Student student) {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            out.writeObject(student);
            System.out.println("\n Student data successfully saved to '" + FILE_NAME + "'.");
        } catch (IOException e) {
            System.out.println(" Error saving student data: " + e.getMessage());
        }
    }
    // Deserialize a Student object from a file
    public static Student loadStudent() {
        File file = new File(FILE_NAME);
        if (!file.exists()) {
            System.out.println(" No saved student data found.");
            return null;
        }
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE_NAME)))
{
            return (Student) in.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(" Error loading student data: " + e.getMessage());
            return null;
        }
    }
}
// Main class for user interaction
public class StudentSerializationApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean running = true;
        while (running) {
            // Display menu
            System.out.println("\n===== Student Data Management =====");
            System.out.println("1. Create & Save Student");
            System.out.println("2. Load & Display Student");
```

```
System.out.println("3. Exit");
System.out.print("Choose an option (1-3): ");
int choice;
try {
    choice = Integer.parseInt(scanner.nextLine());
} catch (NumberFormatException e) {
    System.out.println("Invalid input. Please enter a number (1-3).");
    continue;
}
switch (choice) {
    case 1:
        // Input student details
        System.out.print("Enter Student ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Student Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Student GPA: ");
        double gpa = scanner.nextDouble();
        scanner.nextLine(); // Consume newline
        // Create and save student
        Student student = new Student(id, name, gpa);
        StudentDataManager.saveStudent(student);
        break;
    case 2:
        // Load and display student data
        Student loadedStudent = StudentDataManager.loadStudent();
        if (loadedStudent != null) {
            loadedStudent.displayDetails();
        }
        break;
    case 3:
        System.out.println("Exiting program. Goodbye!");
        running = false;
        break;
    default:
        System.out.println(" Invalid option. Please select 1, 2, or 3.");
}
```

```
            }
            scanner.close();
        }
    }
```

## 4. Output:

```
===== Student Serialization Program =====
1. Create and Serialize a Student
2. Deserialize and Display Student Details
3. Exit
Enter your choice (1-3): 1
Enter Student ID: 1
Enter Student Name: SHIVANSH SINGH
Enter Student GPA: 7.5

☑ Student object serialized successfully and saved to 'student.ser'.

===== Student Serialization Program =====
1. Create and Serialize a Student
2. Deserialize and Display Student Details
3. Exit
Enter your choice (1-3): 2

--- Student Details ---
Student ID  : 1
Student Name: SHIVANSH SINGH
Student GPA : 7.5

===== Student Serialization Program =====
1. Create and Serialize a Student
2. Deserialize and Display Student Details
3. Exit
Enter your choice (1-3): 3
Exiting the program. Goodbye!
```

Fig 2.1. Student details

## 5. Learning Outcomes:

- Comprehend the principles of serialization and deserialization in Java.
- Learn how to store and retrieve object data from files using streams.
- Implement proper exception handling for common file I/O errors.
- Enhance skills in managing persistent object data effectively.
- Gain practical 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.

## 2. Objective:

- Collect and store employee details (name, ID, designation, salary) in a file using file handling techniques.
- Retrieve and display all stored employee records from the file.
- Implement exception handling to ensure reliable and error-free file operations.
- Design a user-friendly and efficient console-based system for managing employee records.

## 3. Code:

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
// Employee class implementing Serializable
class Employee implements Serializable {
    private static final long serialVersionUID = 1L; // Ensures compatibility during serialization
    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;
    }
    // Method to display employee details
    public void display() {
        System.out.println("\n-------------------------");
        System.out.println("Employee ID    : " + id);
        System.out.println("Employee Name  : " + name);
```

```
        System.out.println("Designation    : " + designation);
        System.out.println("Salary         : $" + salary);
        System.out.println("--------------------------");
    }
}
// Handles file operations (saving and loading employee data)
class EmployeeFileManager {
    private static final String FILE_NAME = "employees.dat";
    // Save employee list to file (Serialization)
    public static void saveEmployees(List<Employee> employees) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
            System.out.println("\n Employee data successfully saved!");
        } catch (IOException e) {
            System.out.println("Error saving employee data: " + e.getMessage());
        }
    }
    // Load employee list from file (Deserialization)
    @SuppressWarnings("unchecked")
    public static List<Employee> loadEmployees() {
        File file = new File(FILE_NAME);
        if (!file.exists()) {
            return new ArrayList<>(); // Return empty list if file does not exist
        }
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
            return (List<Employee>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(" Error loading employee data: " + e.getMessage());
            return new ArrayList<>();
        }
    }
}
// Main class to manage employee records
public class EmployeeManagementSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        List<Employee> employees = EmployeeFileManager.loadEmployees(); // Load employees
at startup
        boolean running = true;
        while (running) {
            // Display menu
            System.out.println("\n===== Employee Management System =====");
            System.out.println("1. Add Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Enter your choice (1-3): ");
            int choice;
            try {
                choice = Integer.parseInt(scanner.nextLine()); // Read input as string and convert to
integer
            } catch (NumberFormatException e) {
                System.out.println(" Invalid input! Please enter a number between 1 and 3.");
                continue;
            }
            switch (choice) {
                case 1:
                    // Add new employee
                    System.out.print("\nEnter Employee ID: ");
                    int id = getValidInteger(scanner);
                    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 = getValidDouble(scanner);
                    Employee emp = new Employee(id, name, designation, salary);
                    employees.add(emp); // Add to list
                    EmployeeFileManager.saveEmployees(employees); // Save to file
                    break;
                case 2:
                    // Display all employees
                    if (employees.isEmpty()) {
                        System.out.println("\n No employee records found!");
                    } else {
```

```
                    System.out.println("\n===== Employee Records =====");
                    for (Employee e : employees) {
                        e.display();
                    }
                }
                break;
            case 3:
                // Exit application
                System.out.println("\nExiting the application. Goodbye! ");
                running = false;
                break;
            default:
                System.out.println(" Invalid choice! Please select a valid option.");
        }
    }
    scanner.close();
}
// Helper method to get valid integer input
private static int getValidInteger(Scanner scanner) {
    while (true) {
        try {
            return Integer.parseInt(scanner.nextLine());
        } catch (NumberFormatException e) {
            System.out.print(" Invalid input! Please enter a valid number: ");
        }
    }
}
// Helper method to get valid double input
private static double getValidDouble(Scanner scanner) {
    while (true) {
        try {
            return Double.parseDouble(scanner.nextLine());
        } catch (NumberFormatException e) {
            System.out.print(" Invalid input! Please enter a valid salary: ");
        }
    }
}
}
```

## 4. Output:

```
===== Employee Management System =====
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice (1-3): 1

Enter Employee ID: 1
Enter Employee Name: SHIVANSH SINGH
Enter Designation: SOFTWARE ENGINEER
Enter Salary: 100000

☑ Employee data saved successfully!

===== Employee Management System =====
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice (1-3): 2

===== Employee Records =====

--------------------------
Employee ID    : 1
Employee Name  : SHIVANSH SINGH
Designation    : SOFTWARE ENGINEER
Salary         : $100000.0
--------------------------

===== Employee Management System =====
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice (1-3):
```

Fig 3.1. Employee Management System

## 5. Learning Outcomes:

- Perform file handling operations in Java to read and write data efficiently.
- Utilize object serialization to store and retrieve employee records from a file.
- Design a menu-driven program for seamless employee data management.
- Implement exception handling to ensure robust and error-free execution.
- Structure the code using classes and methods for improved modularity and maintainability.