



## Experiment-5

**Student Name:** Kavikantraj Thakur

**Branch:** BE-CSE

**Semester:** 6th

**Subject Name:** Project Based Learning in Java with Lab

**UID:** 22BCS17022

**Section/Group:** IOT-642-B

**Date of Performance:** 24/02/25

**Subject Code:** 22CSH-359

### Problem 1

- **Aim:** writing a Java program to calculate the sum of a list of integers using autoboxing and unboxing, along with methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

#### **Steps to implement:**

1. Create a List of Integers: Initialize a List<Integer> to hold the integers.
2. Autoboxing: Use autoboxing to convert primitive int values to Integer objects automatically when adding to the list.
3. Unboxing: Use unboxing to convert Integer objects back to int for sum calculation.
4. Parse Strings: Create a utility method to parse strings to integers using Integer.parseInt().
5. Calculate the Sum: Use a loop or Java 8 streams to calculate the sum of the list

#### **Java Program:**

parseStringToInteger(): This method parses a string into an Integer. It catches any NumberFormatException if the string is not a valid number.

calculateSum(): This method calculates the sum of a list of integers. Java automatically performs unboxing when adding Integer values to sum (an int).

#### **Test Cases:**

Test Case 1:

Input: 10, 20, 30, "40", "50"

Expected Output: The sum of the list is: 150

Description: The list contains a mix of primitive integers and integers parsed from strings.

Test Case 2:

Input: "100", "200", "300"

Expected Output: The sum of the list is: 600

Description: All values are parsed from strings, and the sum is calculated.

Test Case 3:

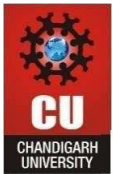
Input: "50", "invalid", "70"

Expected Output:

Invalid number format: invalid

The sum of the list is: 120

Description: One of the inputs is not a valid integer, so it's skipped, and the sum of valid values is calculated.



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

- **Implementation/Code:**

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

public class SumCalculator {

    public static Integer parseStringToInteger(String str) {
        try {
            return Integer.parseInt(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format: " + str);
            return null;
        }
    }

    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            if (num != null) {
                sum += num;
            }
        }
        return sum;
    }

    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<>();
        list1.add(10);
        list1.add(20);
        list1.add(30);
        list1.add(parseStringToInteger("40"));
        list1.add(parseStringToInteger("50"));

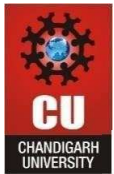
        System.out.println("The sum of the list is: " + calculateSum(list1));

        List<Integer> list2 = new ArrayList<>();
        list2.add(parseStringToInteger("100"));
        list2.add(parseStringToInteger("200"));
        list2.add(parseStringToInteger("300"));

        System.out.println("The sum of the list is: " + calculateSum(list2));

        List<Integer> list3 = new ArrayList<>();
        list3.add(parseStringToInteger("50"));
        list3.add(parseStringToInteger("invalid"));
        list3.add(parseStringToInteger("70"));

        System.out.println("The sum of the list is: " + calculateSum(list3));
    }
}
```



- **Output:**

```
The sum of the list is: 150
The sum of the list is: 600
Invalid number format: invalid
The sum of the list is: 120
```

### Problem 2

- **Aim:** Java program that serializes and deserializes a Student object. It saves the Student object to a file and then reads it back, displaying the student details.

The program handles exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

Steps:

1. Create a Student class with id, name, and GPA.
2. Serialize the Student object: Convert the object to a byte stream and save it to a file.
3. Deserialize the Student object: Read the byte stream from the file and convert it back into an object.
4. Exception handling: Handle possible exceptions such as FileNotFoundException, IOException, and ClassNotFoundException.

### **Implementation**

---Student Class: The Student class implements the Serializable interface, allowing it to be serialized. It has three fields: id, name, and gpa.

---serializeStudent(): This method serializes a Student object to a file using ObjectOutputStream. The object is written to a file named student.ser.

---deserializeStudent(): This method deserializes the Student object from the file using ObjectInputStream. If successful, it returns the deserialized Student object.

---Exception Handling: The program handles FileNotFoundException, IOException, and ClassNotFoundException during the serialization and deserialization processes.

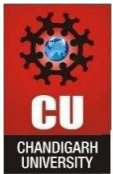
### **Test Cases:**

Test Case 1: Serialize and Deserialize a valid student object.

Input: Student(1, "John Doe", 3.75)

Expected Output:

Student object has been serialized and saved to file.



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Student object has been deserialized.

Deserialized Student Details:

Student ID: 1, Name: John Doe, GPA: 3.75

Test Case 2: Try to deserialize from a non-existent file.

Expected Output:

Error: File not found.

Test Case 3: Handle invalid class during deserialization.

Input: Manually modify the class file to simulate a ClassNotFoundException.

Expected Output:

Error: Class not found.

- **Implementation/Code:**

```
import java.io.*;

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 int getId() {

        return id;

    }

    public String getName() {

        return name;

    }

    public double getGpa() {
```



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return gpa;
    }}

    public class StudentSerialization {

        public static void serializeStudent(Student student) {

            try (ObjectOutputStream out = new ObjectOutputStream(new
                FileOutputStream("student.ser"))) {

                out.writeObject(student);

                System.out.println("Student object has been serialized and saved to file.");
            } catch (FileNotFoundException e) {

                System.out.println("Error: File not found.");
            } catch (IOException e) {

                System.out.println("Error: IOException occurred during serialization.");
            }
        }
    }

    public static Student deserializeStudent() {

        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("student.ser"))) {

            Student student = (Student) in.readObject();

            System.out.println("Student object has been deserialized.");

            return student;
        } catch (FileNotFoundException e) {

            System.out.println("Error: File not found.");
        } catch (IOException e) {

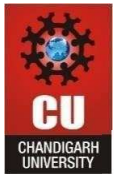
            System.out.println("Error: IOException occurred during deserialization.");
        } catch (ClassNotFoundException e) {

            System.out.println("Error: Class not found.");
        }

        return null;
    }

    public static void main(String[] args) {

        Student student1 = new Student(1, "John Doe", 3.75);
```



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
serializeStudent(student1);

Student deserializedStudent = deserializeStudent();

if (deserializedStudent != null) {

    System.out.println("Deserialized Student Details:");

    System.out.println("Student ID: " + deserializedStudent.getId());

    System.out.println("Student Name: " + deserializedStudent.getName());

    System.out.println("Student GPA: " + deserializedStudent.getGpa());

}

System.out.println("\nTest Case 2: Trying to deserialize from a non-existent file.");

File file = new File("non_existent_file.ser");

if (!file.exists()) {

    System.out.println("Error: File not found.");

}

System.out.println("\nTest Case 3: Handle invalid class during deserialization.");

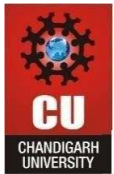
}}
```

- **Output:**

```
Student object has been serialized and saved to file.
Student object has been deserialized.
Deserialized Student Details:
Student ID: 1
Student Name: John Doe
Student GPA: 3.75

Test Case 2: Trying to deserialize from a non-existent file.
Error: File not found.

Test Case 3: Handle invalid class during deserialization.
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Problem 3

- **Aim:**

Menu-based Java application that allows you to add employee details, display all employees, and exit. The employee details will be stored in a file, and the program will read the file to display the stored employee information.

**Steps:**

1. Create an Employee class with fields like name, id, designation, and salary.

2. Create a menu with three options:

Add an Employee

Display All Employees

Exit

3. Store Employee Data in a File: Serialize the employee objects and store them in a file.

4. Read Employee Data from the File: Deserialize the employee objects from the file and display the details.

5. Handle Exceptions: Handle file I/O exceptions.

**Implementation**

**Employee Class:** This class contains details like name, id, designation, and salary. It implements Serializable to allow serialization of Employee objects.

**addEmployee():** This method takes input from the user for an employee's details, creates an Employee object, and saves it to a file using ObjectOutputStream.

**saveEmployeeToFile():** This method appends employee details to a file. The file is opened in append mode (true parameter in FileOutputStream).

**displayAllEmployees():** This method reads all employee objects from the file and prints their details.

**readEmployeesFromFile():** This method reads the employee objects from the file using ObjectInputStream and stores them in a list.

The loop continues until the end of the file is reached (IOException).

**Test Cases:**

Test Case 1: Add a new employee and display all employees.

Steps: Select option 1 to add a new employee, then select option 2 to display all employees.

Input:

Employee Name: John Doe

Employee ID: 101

Designation: Software Engineer



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Salary: 50000

Expected Output:

Employee added successfully!

Employee ID: 101, Name: John Doe, Designation: Software Engineer, Salary: 50000.0

Test Case 2: Try adding multiple employees and display all of them.

Steps: Add multiple employees (using option 1) and then display all employees (using option 2).

Expected Output:

Employee added successfully!

Employee ID: 101, Name: John Doe, Designation: Software Engineer, Salary: 50000.0

Employee added successfully!

Employee ID: 102, Name: Jane Smith, Designation: Manager, Salary: 75000.0

### • Implementation/Code:

```
import java.io.*;
import java.util.*;

class Employee implements Serializable {

    private static final long serialVersionUID = 1L;

    private String name;

    private int id;

    private String designation;

    private double salary;

    public Employee(String name, int id, String designation, double salary) {

        this.name = name;

        this.id = id;

        this.designation = designation;

        this.salary = salary;

    }

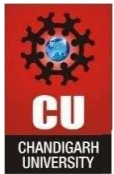
    public int getId() {

        return id;

    }

}
```

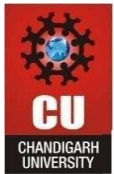




## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public String getName() {  
    return name;  
}  
  
public String getDesignation() {  
    return designation;  
}  
  
public double getSalary() {  
    return salary;  
}  
  
@Override  
public String toString() {  
    return "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " + salary;  
}  
  
public class EmployeeManagementSystem {  
    private static final String FILE_NAME = "employee_data.ser";  
    private static Scanner scanner = new Scanner(System.in);  
    public static void main(String[] args) {  
        while (true) {  
            System.out.println("Menu:");  
            System.out.println("1. 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:  
                    addEmployee();  
                case 2:  
                    displayAllEmployees();  
                case 3:  
                    exit();  
            }  
        }  
    }  
}
```



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        break;

    case 2:

        displayAllEmployees();

        break;

    case 3:

        System.out.println("Exiting...");

        System.exit(0);

        break;

    default:

        System.out.println("Invalid option, please try again.");

    } }

public static void addEmployee() {

    System.out.print("Enter Employee Name: ");

    String name = scanner.nextLine();

    System.out.print("Enter Employee ID: ");

    int id = scanner.nextInt();

    scanner.nextLine(); // consume newline

    System.out.print("Enter Designation: ");

    String designation = scanner.nextLine();

    System.out.print("Enter Salary: ");

    double salary = scanner.nextDouble();

    Employee employee = new Employee(name, id, designation, salary);

    saveEmployeeToFile(employee);

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

}

public static void saveEmployeeToFile(Employee employee) {

    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE_NAME, true)))

    {

        out.writeObject(employee);

    } catch (IOException e) {
```



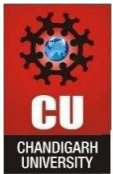
## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        System.out.println("Error occurred while saving employee data: " + e.getMessage());
    }
}

public static void displayAllEmployees() {
    List<Employee> employees = readEmployeesFromFile();
    if (employees.isEmpty()) {
        System.out.println("No employee data found.");
    } else {
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}

public static List<Employee> readEmployeesFromFile() {
    List<Employee> employees = new ArrayList<>();
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        while (true) {
            try {
                Employee employee = (Employee) in.readObject();
                employees.add(employee);
            } catch (EOFException e) {
                break;
            }
        }
    }
    catch (FileNotFoundException e) {
        System.out.println("No data file found, creating a new one.");
    }
    catch (IOException | ClassNotFoundException e) {
        System.out.println("Error occurred while reading employee data: " + e.getMessage());
    }
    return employees;
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

- **Output:**

```
StudentSerializatio... student.ser
1 -i sr Student D gpaI idL namet Ljava/lang/String;xp@ t John Doe
```

```
Student object has been serialized and saved to file.
Student object has been deserialized.
Deserialized Student Details:
Student ID: 1
Student Name: John Doe
Student GPA: 3.75

Test Case 2: Trying to deserialize from a non-existent file.
Error: File not found.

Test Case 3: Handle invalid class during deserialization.

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