



## Experiment 5

**Student Name:** Om Kumar Kushwaha  
**Branch:** CSE  
**Semester:** 6<sup>th</sup>  
**Subject:** Java

**UID:** 22BCS13906  
**Section:** IOT-642/B  
**DOP:** 28/2/025  
**Subject Code:** 22CSH-359

### Problem - 5.1

**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()).

#### Code:

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

public class SumCalculator {

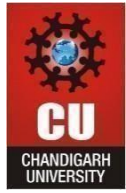
    public static Integer parseStringToInteger(String str) {
        try {
            return Integer.parseInt(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format: " + str);
            return null;
        }
    }

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

    public static void main(String[] args) {

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

        numbers.add(10);
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
numbers.add(20);
numbers.add(30);

// Parsing strings and adding them to the list
numbers.add(parseStringToInteger("40"));
numbers.add(parseStringToInteger("50"));

// Calculating the sum of the list
int sum = calculateSum(numbers);
System.out.println("The sum of the list is: " + sum);

// Test case with strings only
List<Integer> stringParsedNumbers = new ArrayList<>();
stringParsedNumbers.add(parseStringToInteger("100"));
stringParsedNumbers.add(parseStringToInteger("200"));
stringParsedNumbers.add(parseStringToInteger("300"));

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

// Test case with invalid input
List<Integer> invalidInputTest = new ArrayList<>();
invalidInputTest.add(parseStringToInteger("50"));
invalidInputTest.add(parseStringToInteger("invalid"));
invalidInputTest.add(parseStringToInteger("70"));

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

## 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

Process finished with exit code 0
```

## 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

## **Problem - 5.2**

**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.

### **Code :**

```
package javaSem6;

import java.io.*;

// Student class implementing Serializable
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;
    }
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
@Override
public String toString() {
    return "Student ID: " + id + ", Name: " + name + ", GPA: " + gpa;
}

}

public class StudentSerialization {
    private static final String FILE_NAME = "student.ser";

    // 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 has been serialized and saved to file.");
        } catch (IOException e) {
            System.err.println("Error during serialization: " + e.getMessage());
        }
    }

    // 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.err.println("Error: File not found.");
        } catch (IOException e) {
            System.err.println("Error during deserialization: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Error: Class not found.");
        }
        return null;
    }

    public static void main(String[] args) {
        // Test Case 1: Serialize and Deserialize a valid student object
        Student student = new Student(1, "John Doe", 3.75);
        serializeStudent(student);

        Student deserializedStudent = deserializeStudent();
        if (deserializedStudent != null) {
            System.out.println("Student object has been deserialized.");
            System.out.println("Deserialized Student Details:\n" + deserializedStudent);
        }
    }
}
```



### Output :

```
Student object has been serialized and saved to file.  
Student object has been deserialized.  
Deserialized Student Details:  
Student ID: 1, Name: John Doe, GPA: 3.75  
  
Process finished with exit code 0
```

### 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.

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.

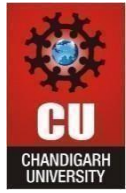
## **Problem - 5.3**

**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.

### Code:

```
package javaSem6;
```

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



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
import java.util.*;

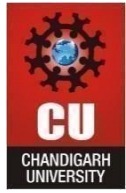
// Employees class implementing Serializable
class Employees implements Serializable
{
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

    public Employees(int id, String name, String designation, double salary)
    {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: "
+ salary;
    }
}

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

    // Method to add an employee
    public static void addEmployee()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();
    }
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
Employees employee = new Employees(id, name, designation, salary);
saveEmployeeToFile(employee);
System.out.println("Employee added successfully!\n");
}

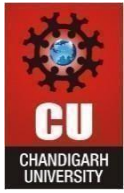
// Method to save an employee to a file
private static void saveEmployeeToFile(Employees employee)
{
    List<Employees> employees = readEmployeesFromFile(); // Read existing employees
    employees.add(employee); // Add new employee

    // Write entire list back to the file (overwriting)
    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME)))
    {
        oos.writeObject(employees);
    } catch (IOException e) {
        System.err.println("Error saving employee: " + e.getMessage());
    }
}

// Method to display all employees
public static void displayAllEmployees() {
    List<Employees> employees = readEmployeesFromFile();
    if (employees.isEmpty()) {
        System.out.println("No employees found.\n");
    } else {
        System.out.println("Employee List:");
        employees.forEach(System.out::println);
        System.out.println();
    }
}

// Method to read employees from file
private static List<Employees> readEmployeesFromFile() {
    List<Employees> employees = new ArrayList<>();
    if (!new File(FILE_NAME).exists()) {
        return employees; // Return empty list if file does not exist
    }

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        Object obj = ois.readObject();
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        if (obj instanceof List<?>) {
            employees = (List<Employees>) obj; // Cast safely
        }
    } catch (EOFException ignored) {
        // End of file reached
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Error reading employees: " + e.getMessage());
    }
    return employees;
}

public static void main(String[] args)
{
    Scanner scanner = new Scanner(System.in);
    while (true)
    {
        System.out.println("1. Add Employee");
        System.out.println("2. Display All Employees");
        System.out.println("3. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1 -> addEmployee();
            case 2 -> displayAllEmployees();
            case 3 -> {
                System.out.println("Exiting program.");
                System.exit(0);
            }
            default -> System.out.println("Invalid choice. Please try again.\n");
        }
    }
}
```





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Output :

```
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 17184
Enter Name: Dipendra
Enter Designation: Software Engineer
Enter Salary: 15000
Employee added successfully!

1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 13906
Enter Name: Om
Enter Designation: Web developer
Enter Salary: 1700
Employee added successfully!

1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 10790
Enter Name: Sanjay
Enter Designation: Full stack developer
Enter Salary: 16000
Employee added successfully!
```

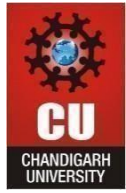
```
Employee ID: 17184, Name: Dipendra, Designation: Software Engineer, Salary: 15000.0
Employee ID: 13906, Name: Om, Designation: Web developer, Salary: 1700.0
Employee ID: 10790, Name: Sanjay, Designation: Full stack developer, Salary: 16000.0
Employee ID: 50206, Name: Sumantra, Designation: Software engineer, Salary: 15000.0
```

## Test Cases:

1. Add Employee
2. Display All Employees
3. Exit

Enter your choice: 1  
Enter Employee ID: 17184  
Enter Name: Dipendra  
Enter Designation: Software engineer  
Enter Salary: 15000  
Employee added successfully!

1. Add Employee
2. Display All Employees



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

3. Exit

Enter your choice: 1

Enter Employee ID: 13906

Enter Name: Om

Enter Designation: Web developer

Enter Salary: 17000

Employee added successfully!

1. Add Employee

2. Display All Employees

3. Exit

Enter your choice: 1

Enter Employee ID: 10790

Enter Name: Sanjay

Enter Designation: Full stack developer

Enter Salary: 16000

Employee added successfully!

1. Add Employee

2. Display All Employees

3. Exit

Enter your choice: 1

Enter Employee ID: 50206

Enter Name: Sumantra

Enter Designation: Software developer

Enter Salary: 17000

Employee added successfully!

1. Add Employee

2. Display All Employees

3. Exit

Enter your choice: 2

Employee List:

Employee ID: 17184, Name: Dipendra, Designation: Software engineer, Salary: 15000.0

Employee ID: 13906, Name: Om, Designation: Web developer, Salary: 17000.0

Employee ID: 10790, Name: Sanjay, Designation: Full stack developer, Salary: 16000.0

Employee ID: 50206, Name: Sumantra, Designation: Software developer, Salary: 17000.0