



## Experiment 5

**Student Name:** Sameer Gautam

**UID:** 22BCS10410

**Branch:** CSE

**Section:**618(B)

**Semester:** 6<sup>th</sup>

**DOP:**21/02/2025

**Subject:** Java

**Subject Code:** 22CSH-359

### Problem Statement 1

1. **Aim:** 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.
2. **Objective:** The objective of this experiment is to enhance understanding of Java's wrapper classes, I/O streams, and lambda expressions by implementing practical applications. This includes efficiently handling data conversions using autoboxing and unboxing, managing file operations through serialization and deserialization, and leveraging functional programming concepts to write concise and effective Java programs.
3. **Code:**

```
package College;
```

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

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

```
import java.util.Scanner;
```

```
public class SumCalculator {
```

```
    public static Integer parseStringToInteger(String str) {
```

```
        str = str.replaceAll("\\\"", "").trim();
```

```
        try {
```

```
            return Integer.parseInt(str);
```

```
        } catch (NumberFormatException e) {
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
System.out.println("Invalid number format: " + str);

return null; // Return null if invalid input

}

}

// Method to calculate sum

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

int sum = 0;

for (Integer num : numbers) {

if (num != null) { // Ignore null values

sum += num; }}

return sum;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.println("Enter numbers separated by spaces or commas:");

String input = scanner.nextLine();

// Split input by comma and space, then trim each value

String[] tokens = input.split("[,\\s]+"); // Handles spaces and commas

// Convert each input to an integer

for (String token : tokens) {

numbers.add(parseStringToInteger(token));

}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Calculate and display sum
```

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

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

#### 4. Output:

```
Enter numbers separated by spaces or commas:
"4" "3" "5" "s"
Invalid number format: s
The sum of the list is: 12
PS C:\Users\samro\Desktop\Codes\Java>
```

#### Problem Statement 2

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. Code:

```
package DSA;
```

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

```
class Student implements Serializable {
    private static final long serialVersionUID = 1L;
```

```
    private int id;
    private String name;
    private double gpa;
```

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

```
    // Getters to access private fields
    public int getId() { return id; }
    public String getName() { return name; }
    public double getGpa() { return gpa; }
}
```

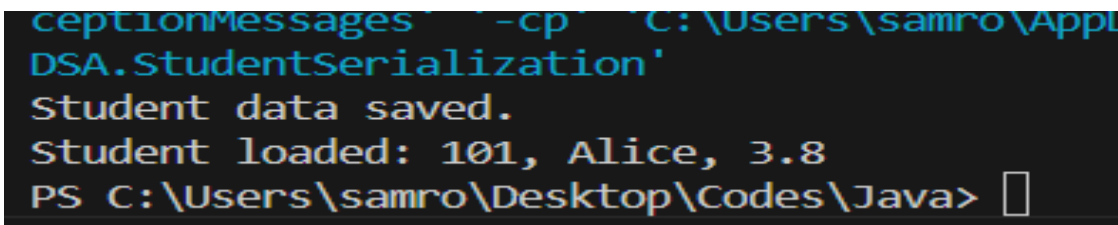
```
public class StudentSerialization {
```

```
public static void main(String[] args) {
    Student student = new Student(101, "Alice", 3.8);
    saveStudent(student);
    loadStudent();
}

static void saveStudent(Student student) {
    try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("student.ser"))) {
        out.writeObject(student);
        System.out.println("Student data saved.");
    } catch (IOException e) {
        System.out.println("Error saving student data: " + e.getMessage());
    }
}

static void loadStudent() {
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("student.ser"))) {
        Student student = (Student) in.readObject();
        System.out.println("Student loaded: " + student.getId() + ", " + student.getName() + ", " +
student.getGpa());
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error loading student data: " + e.getMessage());
    }
}
}
```

## 5. Output:



```
ceptionMessages' -cp C:\Users\samro\AppData\Local\Temp\
DSA.StudentSerialization'
Student data saved.
Student loaded: 101, Alice, 3.8
PS C:\Users\samro\Desktop\Codes\Java>
```

## Problem Statement 3

- **Aim:** Create a menu-based Java application with the following options. 1.Add an Employee
  - 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. If option 3 is selected the application should exit.

## 2. Code:

```
package DSA;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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

class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    String id;
    String name, designation;
    double salary;

    Employee(String id, String name, String designation, double salary) {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

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

public class EmployeeMenuApp {
    private static final String FILE_NAME = "employees.ser";
    private static List<Employee> employees = new ArrayList<>();

    public static void main(String[] args) {
        loadEmployees();
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\n1. Add Employee 2. Display All 3. Exit");
            System.out.print("Choose an option: ");
            int choice;
            try {
                choice = Integer.parseInt(scanner.nextLine()); // Handle non-integer input safely
            } catch (NumberFormatException e) {
                System.out.println("Invalid input! Please enter a number.");
                continue;
            }

            switch (choice) {
                case 1 -> addEmployee(scanner);
                case 2 -> displayEmployees();
                case 3 -> {
                    saveEmployees();
                    System.out.println("Exiting...");
                    scanner.close();
                    System.exit(0);
                }
                default -> System.out.println("Invalid choice! Please enter 1, 2, or 3.");
            }
        }
    }
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}
```

```
static void addEmployee(Scanner scanner) {  
    System.out.print("Enter ID: ");  
    String id = scanner.nextLine(); // Changed from nextInt() to nextLine()  
  
    // Check for duplicate ID  
    boolean idExists = employees.stream().anyMatch(emp -> emp.id.equals(id));  
    if (idExists) {  
        System.out.println("Employee with this ID already exists!");  
        return;  
    }  
}
```

```
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;  
try {  
    salary = Double.parseDouble(scanner.nextLine()); // Handle invalid salary input  
    if (salary < 0) {  
        System.out.println("Salary cannot be negative!");  
        return;  
    }  
} catch (NumberFormatException e) {  
    System.out.println("Invalid salary input! Please enter a valid number.");  
    return;  
}
```

```
employees.add(new Employee(id, name, designation, salary));  
System.out.println("Employee added successfully!");  
}
```

```
static void displayEmployees() {  
    if (employees.isEmpty()) {  
        System.out.println("No employees found.");  
    } else {  
        employees.forEach(System.out::println);  
    }  
}
```

```
static void saveEmployees() {  
    try (ObjectOutputStream oos = new ObjectOutputStream(new  
FileOutputStream(FILE_NAME))) {  
        oos.writeObject(employees);  
    } catch (IOException e) {  
        System.out.println("Error saving employee data.");  
    }  
}
```

```
static void loadEmployees() {
    File file = new File(FILE_NAME);
    if (!file.exists()) {
        employees = new ArrayList<>();
        return;
    }

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        employees = (List<Employee>) ois.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error loading employee data. Starting fresh.");
        employees = new ArrayList<>();
    }
}
```

### 3. Output:

```
Enter ID: 22BCS10410
Enter Name: Sameer Gautam
Enter Designation: HR
Enter Salary: 2000000
Employee added successfully!

1. Add Employee  2. Display All  3. Exit
Choose an option: 2
ID: 22BCS10410, Name: Sameer Gautam, Designation: HR, Salary: 2000000.0

1. Add Employee  2. Display All  3. Exit
Choose an option: █
```

### 4. Learning Outcomes

- Understand the concept of autoboxing and unboxing in Java and how primitive data types are automatically converted to their corresponding wrapper classes.
- Learn how to serialize and deserialize objects using Java's I/O streams, enabling efficient data storage and retrieval.
- Develop error-handling skills by managing exceptions like `FileNotFoundException`, `IOException`, and `ClassNotFoundException` during file operations.
- Gain experience in functional programming by implementing lambda expressions, method references, and functional interfaces for cleaner and more efficient code.



# **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.