



## Experiment 5.1

**Student Name:** Shirsha Rakshit

**Branch:** CSE

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

**Subject:** PBLJ

**UID:** 22BCS12279

**Section:** 22BCS\_IOT-618/A

**DOP:** 21/02/25

**Subject Code:** 22CSH-359

**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 (e.g., Integer.parseInt()).

**Objective:** Demonstrate **autoboxing** and **unboxing** in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

### **Algorithm:**

#### **Step 1: Initialize the Program**

1. Start the program.
2. Import ArrayList and List classes.
3. Define the AutoboxingExample class.

#### **Step 2: Convert String Array to Integer List**

1. Define the method parseStringArrayToIntegers(String[] strings).
2. Create an empty ArrayList<Integer>.
3. Iterate through the string array:
  - Convert each string to an Integer using Integer.parseInt(str).
  - Add the integer to the list (**autoboxing** happens here).
4. Return the list of integers.

#### **Step 3: Calculate the Sum of Integers**

1. Define the method calculateSum(List<Integer> numbers).
2. Initialize a variable sum to 0.
3. Iterate through the list:
  - Extract each integer (**unboxing** happens here).
  - Add it to sum.
4. Return the total sum.

#### **Step 4: Execute Main Function**

1. Define main(String[] args).
2. Create a string array with numeric values.
3. Call parseStringArrayToIntegers() to convert it into a list of integers.
4. Call calculateSum() to compute the sum.
5. Print the result.

#### **Step 5: Terminate the Program**

1. End the execution.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Code:

```
import java.util.ArrayList; // Importing ArrayList to store integers
import java.util.List;      // Importing List interface for flexibility
import java.util.Scanner;   // Importing Scanner for user input

public class IntegerSumCalculator {

    // Method to parse a string into an Integer
    public static Integer parseStringToInteger(String str) {
        try {
            return Integer.parseInt(str); // Converts string to Integer (Autoboxing happens here)
        } catch (NumberFormatException e) { // Handles invalid inputs that cannot be converted
            System.out.println("Invalid number format: " + str);
            return null; // Return null for invalid input
        }
    }

    // Method to calculate the sum of a list of Integers
    public static int calculateSum(List<Integer> numbers) {
        int sum = 0; // Variable to store the sum
        for (Integer num : numbers) { // Iterates through the list
            if (num != null) { // Ensures null values are ignored
                sum += num; // Adds the value to sum (Unboxing happens here)
            }
        }
        return sum; // Returns the total sum
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // Creating Scanner object to take input
        List<Integer> numbers = new ArrayList<>(); // List to store integer inputs

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

        while (true) { // Infinite loop until 'done' is entered
            String input = scanner.nextLine(); // Reads user input as a string
            if (input.equalsIgnoreCase("done")) { // Checks if user wants to stop
                break; // Exit loop
            }
            Integer number = parseStringToInteger(input); // Converts input to Integer
            if (number != null) { // Adds only valid numbers to the list
                numbers.add(number); // Autoboxing: Converts int to Integer automatically
            }
        }

        scanner.close(); // Closing the scanner to prevent memory leaks

        // Calculating and displaying the sum of valid numbers entered
        System.out.println("The sum of the list is: " + calculateSum(numbers));
    }
}
```

## Output:

### Test case 1:

```
C:\Users\LENOVO\.jdk\openjdk-23.0.1\bin\  
Enter numbers (type 'done' to finish):  
200  
300  
400  
done  
The sum of the list is: 900  
  
Process finished with exit code 0
```

### Test case 2:

```
C:\Users\LENOVO\.jdk\openjdk-23.0.1\bi  
Enter numbers (type 'done' to finish):  
40  
50  
60  
done  
The sum of the list is: 150  
  
Process finished with exit code 0
```

### Test case 3:

```
C:\Users\LENOVO\.jdk\openjdk-23.0.1\bin\java.  
Enter numbers (type 'done' to finish):  
200  
300  
400  
600  
abc  
Invalid number format: abc  
|
```

### • Learning Outcomes:

- Understand the concept of **autoboxing and unboxing** in Java and how primitive types are automatically converted to their wrapper classes and vice versa.
- Learn how to **convert string values into Integer objects** using `Integer.parseInt()` and store them in a list.
- Gain experience in **working with ArrayLists** to store and manipulate a collection of numbers dynamically.
- Develop proficiency in **iterating through collections** and performing arithmetic operations like summation.

## Experiment 5.2

**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. Objective:** The objective is to serialize and deserialize a Student object, store and retrieve its id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

### **3. Algorithm:**

Step 1: Initialize the Program

1. Start the program.
2. Import the necessary classes (java.io.\*).
3. Define a Student class implementing Serializable.
4. Declare attributes:
  - id (int)
  - name (String)
  - gpa (double)
5. Define a constructor to initialize Student objects.
6. Override toString() to display student details.

Step 2: Define the Serialization Method

1. Create serializeStudent(Student student).
2. Use a try-with-resources block to create an ObjectOutputStream:
  - Open a FileOutputStream to write to student.ser.
  - Write the Student object to the file using writeObject().
3. Handle exceptions:
  - FileNotFoundException → Print error message.
  - IOException → Print error message.
4. Print a success message if serialization is successful.

Step 3: Define the Deserialization Method

1. Create deserializeStudent().
2. Use a try-with-resources block to create an ObjectInputStream:
  - Open a FileInputStream to read student.ser.
  - Read the Student object using readObject().
3. Handle exceptions:
  - FileNotFoundException → Print error message.
  - IOException → Print error message.
  - ClassNotFoundException → Print error message.
4. Print the deserialized student details.

Step 4: Execute Main Function

1. Define main(String[] args).
2. Create a Student object with sample data.
3. Call serializeStudent() to save the object.
4. Call deserializeStudent() to read and display the object.
- 5.



6. Step 5: Terminate the Program
7. End execution.

#### 4. Implementation Code:

```
import java.io.*; // Importing required classes for serialization and file handling //Exp 5.2
import java.util.Scanner; // Importing Scanner class for user input

// Student class implementing Serializable to enable object serialization
class Student implements Serializable {
    private static final long serialVersionUID = 1L; // Recommended to ensure compatibility during
    deserialization
    // Private fields to store student details
    private int id;
    private String name;
    private double gpa;
    // Constructor to initialize Student object
    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    // Method to display student details
    public void display() {
        System.out.println("Student ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}

// Main class for serialization and deserialization
public class StudentSerialization {
    private static final String FILE_NAME = "student.ser"; // File where the student object will be
    saved

    // Method to serialize Student object to file
    public static void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(FILE_NAME))) {
            oos.writeObject(student); // Writing student object to file
            System.out.println("Student object has been serialized and saved to file.");
        } catch (IOException e) { // Handling possible I/O exceptions
            System.out.println("Error during serialization: " + e.getMessage());
        }
    }
}
```

```
// Method to deserialize Student object from file
public static Student deserializeStudent() {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        return (Student) ois.readObject(); // Reading object from file and casting it back to Student
    } catch (FileNotFoundException e) { // Handling file not found exception
        System.out.println("Error: File not found.");
    } catch (IOException e) { // Handling input/output errors
        System.out.println("Error during deserialization: " + e.getMessage());
    } catch (ClassNotFoundException e) { // Handling case where class is not found
        System.out.println("Error: Class not found.");
    }
    return null; // Return null if deserialization fails
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in); // Scanner object to take user input

    // Taking user input for Student details
    System.out.print("Enter Student ID: ");
    int id = scanner.nextInt(); // Reading integer input for student ID
    scanner.nextLine(); // Consuming the newline left after nextInt()

    System.out.print("Enter Student Name: ");
    String name = scanner.nextLine(); // Reading student name as a string

    System.out.print("Enter Student GPA: ");
    double gpa = scanner.nextDouble(); // Reading double input for GPA

    // Creating a Student object with the provided details
    Student student = new Student(id, name, gpa);

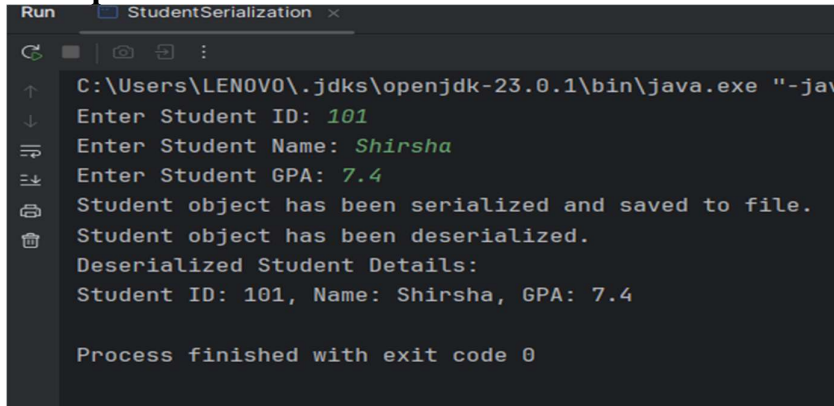
    // Calling method to serialize (save) the Student object to a file
    serializeStudent(student);

    // Calling method to deserialize (read) the Student object from the file
    Student deserializedStudent = deserializeStudent();

    // If deserialization was successful, display the student details
    if (deserializedStudent != null) {
        System.out.println("Student object has been deserialized.");
        System.out.println("Deserialized Student Details:");
        deserializedStudent.display(); // Calling display method of the deserialized object
    }

    scanner.close(); // Closing scanner to prevent memory leaks
}
```

## 5. Output



```
Run StudentSerialization x
C:\Users\LENOVO\jdk\openjdk-23.0.1\bin\java.exe "-jav
Enter Student ID: 101
Enter Student Name: Shirsha
Enter Student GPA: 7.4
Student object has been serialized and saved to file.
Student object has been deserialized.
Deserialized Student Details:
Student ID: 101, Name: Shirsha, GPA: 7.4

Process finished with exit code 0
```

## 6. Learning Outcomes:

- Understand object serialization and deserialization in Java.
- Learn how to use ObjectOutputStream and ObjectInputStream for file operations.
- Implement exception handling for FileNotFoundException, IOException, and ClassNotFoundException.
- Gain hands-on experience in storing and retrieving objects from a file.
- Develop skills in data persistence and file management using Java.

## Experiment 5.3

1. **Aim:** Create a menu-based Java application with the following options.

1. Add an Employee

2. 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. **Objective:** The objective is to develop a menu-based Java application that allows users to **add employee details, store them in a file, and display all stored employee records**, with an option to exit the program.

3. **Algorithm:**

1. Start

2. Display Menu:

- Add an Employee
- Display All Employees
- Exit

3. **Repeat Until User Selects Exit:**

- **If user selects option 1 (Add an Employee):**

- Prompt user to enter Employee Name, Employee ID, Designation, and Salary.

4. Store the details in an Employee object.

5. Append the Employee object to a file.

6. **If user selects option 2 (Display All Employees):**

Read employee details from the file.

Display all stored employee records.

7. **If user selects option 3 (Exit):**

Terminate the program.

**Else:**

Display an "Invalid option" message.

8. **End**

## 4. Implementation Code:

```
import java.io.*; // Importing necessary classes for file handling and serialization //Exp 5.3
```

```
import java.util.*; // Importing utility classes, including List and Scanner
```

```
// Employee class implementing Serializable to allow object serialization
```

```
class Employee implements Serializable {
```

```
    private static final long serialVersionUID = 1L; // Ensures compatibility during deserialization
```

```
    // Private fields to store employee details
```

```
    private int id;
```

```
    private String name;
```

```
    private String designation;
```

```
    private double salary;
```





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Constructor to initialize Employee object
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("Employee ID: " + id + ", Name: " + name + ", Designation: " +
designation + ", Salary: " + salary);
}

// Main class for employee management system
public class EmployeeManagement {
    private static final String FILE_NAME = "employees.ser"; // File where employee objects will be
stored
    private static final Scanner scanner = new Scanner(System.in); // Scanner object for user input

    // Method to add a new employee and save it to file
    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt(); // Read employee ID
        scanner.nextLine(); // Consume the newline character left by nextInt()

        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine(); // Read employee name

        System.out.print("Enter Designation: ");
        String designation = scanner.nextLine(); // Read employee designation

        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble(); // Read employee salary

        // Creating an Employee object with user input
        Employee employee = new Employee(id, name, designation, salary);

        // Save the employee object to the file
        saveEmployeeToFile(employee);

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

    // Method to save an employee object to file using serialization
    public static void saveEmployeeToFile(Employee employee) {
        List<Employee> employees = readEmployeesFromFile(); // Read existing employees from file
        employees.add(employee); // Add the new employee to the list
    }
}
```

```
// Serialize the updated list of employees and save it to the file
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
    oos.writeObject(employees); // Writing the list of employees to the file
} catch (IOException e) {
    e.printStackTrace(); // Print error details for debugging
}

// Method to display all employees stored in the file
public static void displayAllEmployees() {
    List<Employee> employees = readEmployeesFromFile(); // Retrieve list of employees from
file

    if (employees.isEmpty()) { // Check if there are no employees
        System.out.println("No employees found.");
    } else {
        for (Employee emp : employees) { // Loop through the list and display each employee
            emp.display();
        }
    }
}

// Method to read the list of employees from the file
@SuppressWarnings("unchecked") // Suppresses unchecked cast warning
public static List<Employee> readEmployeesFromFile() {
    List<Employee> employees = new ArrayList<>(); // Initialize an empty list to store employees

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        employees = (List<Employee>) ois.readObject(); // Read and cast object from file
    } catch (FileNotFoundException e) {
        // If file not found, no action needed (first run scenario)
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace(); // Print error details for debugging
    }

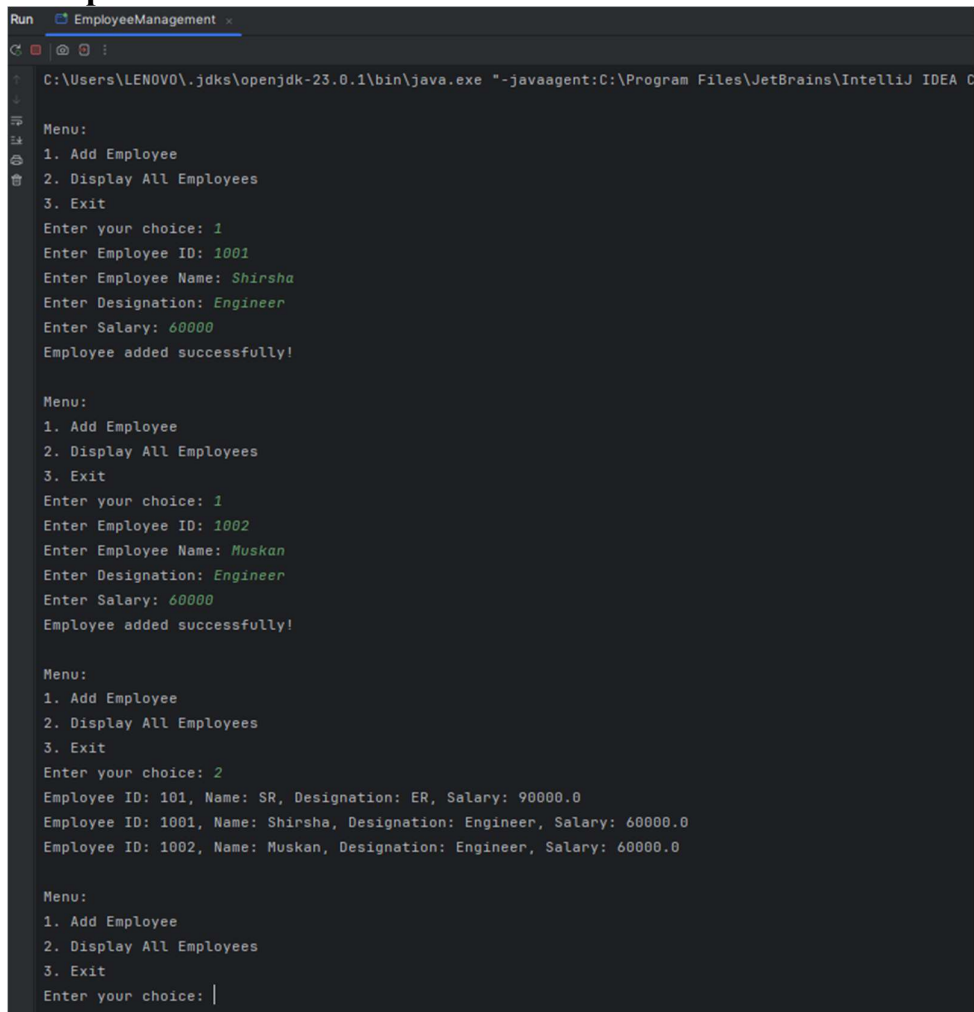
    return employees; // Return the list of employees
}

// Main method to display menu and handle user choices
public static void main(String[] args) {
    while (true) { // Infinite loop for menu until user chooses to exit
        System.out.println("\nMenu:");
        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(); // Read user choice
```

```
switch (choice) {  
    case 1:  
        addEmployee(); // Call method to add employee  
        break;  
    case 2:  
        displayAllEmployees(); // Call method to display employees  
        break;  
    case 3:  
        System.out.println("Exiting program...");  
        scanner.close(); // Close scanner before exiting  
        return; // Exit program  
    default:  
        System.out.println("Invalid choice. Please try again."); // Handle invalid input  
}  
}  
}
```

## 5.Output:



```
Run EmployeeManagement x  
C:\Users\LENOVO\jdk\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA C  
Menu:  
1. Add Employee  
2. Display All Employees  
3. Exit  
Enter your choice: 1  
Enter Employee ID: 1001  
Enter Employee Name: Shirsha  
Enter Designation: Engineer  
Enter Salary: 60000  
Employee added successfully!  
  
Menu:  
1. Add Employee  
2. Display All Employees  
3. Exit  
Enter your choice: 1  
Enter Employee ID: 1002  
Enter Employee Name: Muskan  
Enter Designation: Engineer  
Enter Salary: 60000  
Employee added successfully!  
  
Menu:  
1. Add Employee  
2. Display All Employees  
3. Exit  
Enter your choice: 2  
Employee ID: 101, Name: SR, Designation: ER, Salary: 90000.0  
Employee ID: 1001, Name: Shirsha, Designation: Engineer, Salary: 60000.0  
Employee ID: 1002, Name: Muskan, Designation: Engineer, Salary: 60000.0  
  
Menu:  
1. Add Employee  
2. Display All Employees  
3. Exit  
Enter your choice: |
```



## 6.Learning Outcomes:

- Understand file handling and serialization in Java to store and retrieve objects persistently.
- Learn how to implement a menu-driven console application using loops and conditional statements.
- Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.
- Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
- Develop skills in list manipulation and user input handling using ArrayList and Scanner.