



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 5

Student Name: Ronit Jain

Branch: IT

Semester: 6

Subject Name: Java Lab

UID: 22BET10242

Section/Group: 22BET_701_A

Date of Performance: 18-2-2025

Subject Code: 22ITH-352

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

2. Objective:

- To Demonstrate Autoboxing and Unboxing
- To calculate sum of integers.

3. Implementation/Code:

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

public class SumUsingAutoboxing
{
    public static void main(String[] args)
    {
        String[] numbers = {"10", "20", "30", "40", "50"};
        List<Integer> integerList = new ArrayList<>();
        for (String num : numbers) {
            integerList.add(Integer.parseInt(num)); // Autoboxing
        }

        int sum = calculateSum(integerList);
        System.out.println("Sum of numbers: " + sum);
    }

    public static int calculateSum(List<Integer> list)
    {
        int sum = 0;
        for (Integer num : list)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        { sum += num;
        }
    return sum;
}
}
```

5. Output:

```
<terminated> EXP5_1 [Java Application] C:\Program Files\Java\jdk-23\bin\jav
Enter numbers separated by space:
7 8 9 6 54 2
Sum of numbers: 86
```

6. Learning Outcome:

- Learn how Java automatically converts between primitive types and their wrapper classes when adding/removing elements from collections
- Gain experience in reading user input, splitting strings, and converting them into numerical values using `Integer.parseInt()`.
- Learn how to store user-provided integers in an `ArrayList<Integer>`, iterate through the list, and perform calculations using loops



Problem 2

4. **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.

5. **Objective:**

- To Convert a Student object into a binary format and store it in a file
- To Retrieve the object from the file and reconstruct it using deserialization
- To implement the Serializable interface to allow objects to be written to and read from a file

6. **Implementation/Code:**

```
import java.io.*;

class Student implements Serializable {

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

    int id;

    String name;

    double gpa;

    public Student(int id, String name, double gpa)

        { this.id = id;

          this.name = name;

          this.gpa = gpa;

        }

    // Display Student details

    public void display() {

        System.out.println("Student ID: " + id);
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        System.out.println("Name: " + name);

        System.out.println("GPA: " + gpa);

    }

}

public class StudentSerialization

{ public static void main(String[] args)

{

    Student student = new Student(101, "Shuvam", 7.8);

    String filename = "student.ser"; // File to store serialized object

    serializeStudent(student, filename);

    Student deserializedStudent = deserializeStudent(filename);

    if (deserializedStudent != null) {

        System.out.println("\nDeserialized Student Details:");

        deserializedStudent.display();

    }

}

public static void serializeStudent(Student student, String filename)

{ try (ObjectOutputStream oos = new ObjectOutputStream(new

FileOutputStream(filename)))

{ oos.writeObject(student); // Serialize

object

    System.out.println("Student object serialized successfully.");

} catch (IOException e) {

    System.out.println("Error during serialization: " + e.getMessage());

}

}

public static Student deserializeStudent(String filename) {

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filename)))
```

```
        { return (Student) ois.readObject();  
      } catch (FileNotFoundException e)  
      { System.out.println("File not found: " + filename);  
    } catch (IOException e) {  
      System.out.println("Error during deserialization: " + e.getMessage());  
    } catch (ClassNotFoundException e) {  
      System.out.println("Class not found error: " + e.getMessage());  
    }  
    return null;  
  }  
}
```

7. Output:

```
Student object serialized successfully.  
  
Deserialized Student Details:  
Student ID: 80001  
Name: Tejasv  
GPA: 7.8
```

8. Learning Outcome:

- Learn how to convert Java objects into a binary format for storage and retrieve them later while maintaining their state
- Gain hands-on experience in handling exceptions like FileNotFoundException, IOException, and ClassNotFoundException to ensure error-free file operations
- Learn how to implement the Serializable interface and use ObjectOutputStream and ObjectInputStream for efficient object persistence

Problem 3

7. 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

8. Objective:

- to read from and write to a file using FileWriter, BufferedWriter, and PrintWriter for storing and retrieving employee data.
- designing user-friendly menu-based applications using loops and switch cases for handling user choices.
- to take structured user input (such as integers, strings, and doubles) and process it correctly to avoid common input-related errors.

9. Implementation/Code:

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

public class EmployeeManagement {
    private static final String FILE_NAME = "employees.txt"; // File to store employee data

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

        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (choice)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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



```
// Method to add an employee and store details in a file
public static void addEmployee(Scanner scanner) {
    try (FileWriter fw = new FileWriter(FILE_NAME, true);
        BufferedWriter bw = new BufferedWriter(fw);
        PrintWriter out = new PrintWriter(bw)) {

        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        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 = scanner.nextDouble();

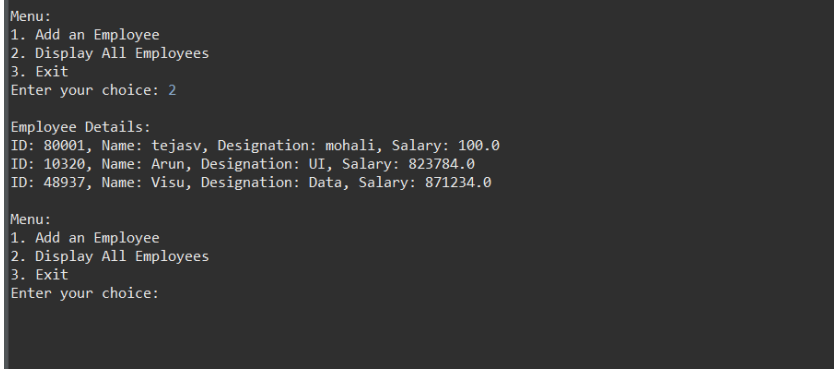
        // Store details in file
        out.println(id + "," + name + "," + designation + "," + salary);
        System.out.println("Employee added successfully!");

    } catch (IOException e) {
        System.out.println("Error writing to file: " + e.getMessage());
    }
}
```

```
}

// Method to display all employees by reading the file
public static void displayEmployees() {
    try (BufferedReader br = new BufferedReader(new FileReader(FILE_NAME))) {
        String line;
        System.out.println("\nEmployee Details:");
        while ((line = br.readLine()) != null) {
            String[] data = line.split(",");
            System.out.println("ID: " + data[0] + ", Name: " + data[1] +
                               ", Designation: " + data[2] + ", Salary: " + data[3]);
        }
    } catch (FileNotFoundException e)
    { System.out.println("No employee records found.");
    } catch (IOException e) {
        System.out.println("Error reading file: " + e.getMessage());
    }
}
}
```

9. Output:



```
Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2

Employee Details:
ID: 80001, Name: tejasv, Designation: mohali, Salary: 100.0
ID: 10320, Name: Arun, Designation: UI, Salary: 823784.0
ID: 48937, Name: Visu, Designation: Data, Salary: 871234.0

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice:
```

10. Learning Outcome:

- Successfully read and write employee data to a file using FileWriter, BufferedReader, and PrintWriter.
- Learn to handle file-related exceptions such as IOException and FileNotFoundException, ensuring program stability
- Gain experience in storing structured employee data in a text file and retrieving it using string manipulation techniques like split()