



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment -5

StudentName:Dilbag

Branch: BE-CSE

Semester:6th

Subject Name: Project Based Learning
in Java with Lab

UID:22BCS16350

Section/Group:IOT_642-B

Date of Performance:24/02/2025

Subject Code: 22CSH-359

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

5.1.2Objective: The program demonstrates autoboxing and unboxing in Java by parsing a list of integer strings into their respective wrapper classes (Integer), computing their sum, and printing the result.

5.1.3 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("Test Case 1 Output: 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"));
```

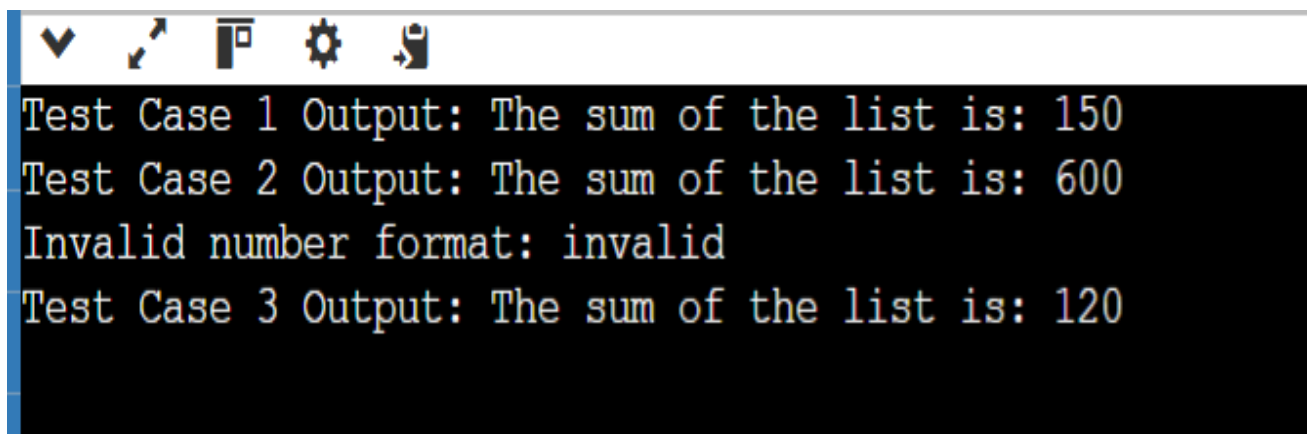
```
System.out.println("Test Case 2 Output: The sum of the list is: " +  
calculateSum(list2));
```

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

```
System.out.println("Test Case 3 Output: The sum of the list is: " +  
calculateSum(list3));
```

```
}  
}
```

5.1.4 Output:



```
Test Case 1 Output: The sum of the list is: 150  
Test Case 2 Output: The sum of the list is: 600  
Invalid number format: invalid  
Test Case 3 Output: The sum of the list is: 120
```

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 Exception using exception handling.

5.2.2 Objective: The program demonstrates serialization and deserialization in Java by storing a Student object (with id, name, and GPA) to a file and then retrieving it while handling possible exceptions.

5.2.3 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 void display() {
        System.out.println("Student ID: " + id + ", Name: " + name + ", GPA: " + gpa);
    }
}

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

    public static void main(String[] args) {
        // **Test Case 1: Serialize and Deserialize a valid student object**
        System.out.println("=== Test Case 1: Valid Serialization & Deserialization ===");
        Student student = new Student(1, "John Doe", 3.75);
        serializeStudent(student);
        Student deserializedStudent = deserializeStudent(FILE_NAME);
        if (deserializedStudent != null) {
            System.out.println("Deserialized Student Details:");
            deserializedStudent.display();
        }
    }
}
```

```
// **Test Case 2: Try to deserialize from a non-existent file**
System.out.println("\n=== Test Case 2: Deserialize from a Non-Existent File ===");
deserializeStudent("non_existent_file.ser");

// **Test Case 3: Handle invalid class during deserialization**

System.out.println("\n=== Test Case 3: Simulating ClassNotFoundException ===");
deserializeInvalidClass();
}

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 (FileNotFoundException e) {
        System.err.println("Error: File not found.");
    } catch (IOException e) {
        System.err.println("Error: IO Exception occurred during serialization.");
    }
}

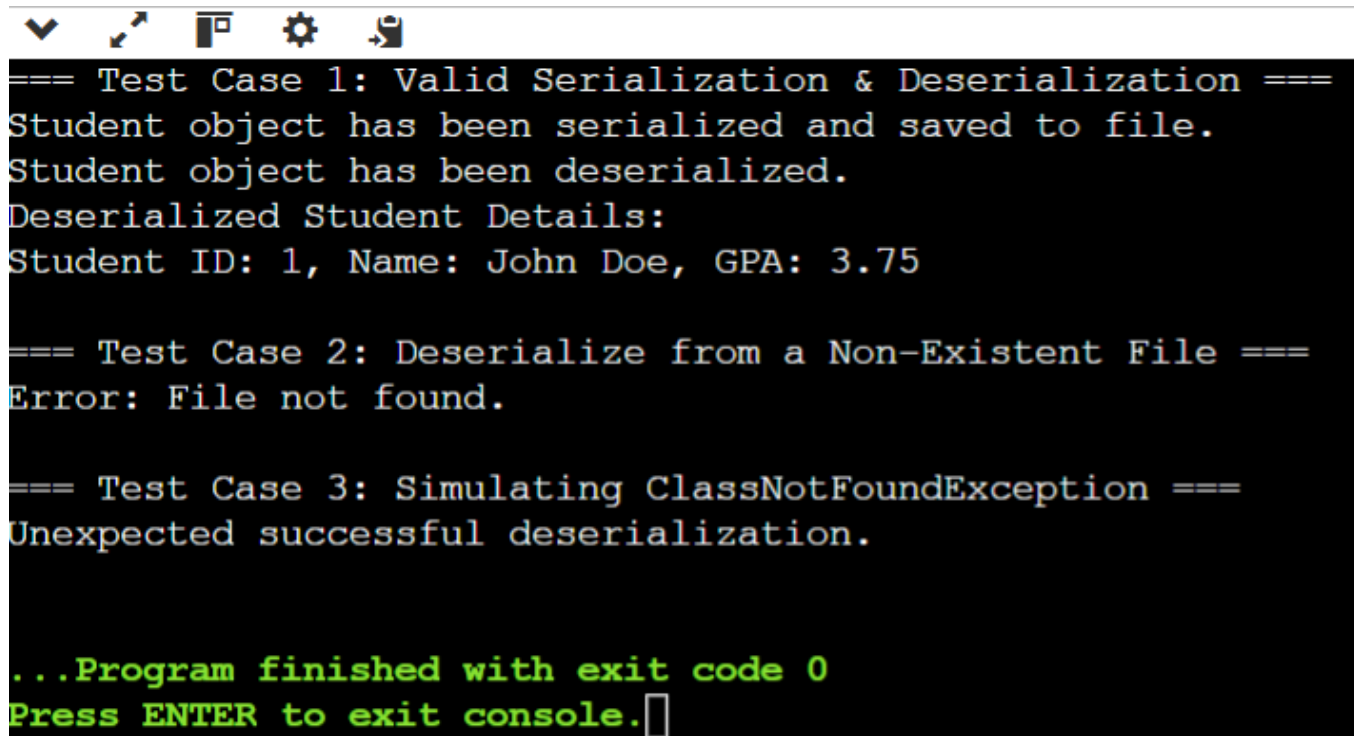
public static Student deserializeStudent(String fileName) {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
        System.out.println("Student object has been deserialized.");
        return (Student) ois.readObject();
    } catch (FileNotFoundException e) {
        System.err.println("Error: File not found.");
    } catch (IOException e) {
        System.err.println("Error: IO Exception occurred during deserialization.");
    } catch (ClassNotFoundException e) {
        System.err.println("Error: Class not found.");
    }
    return null;
}

public static void deserializeInvalidClass() {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream
        (FILE_NAME))) {
        Object obj = ois.readObject();
        if (!(obj instanceof Student)) {

```

```
        throw new ClassNotFoundException();
    }
    System.out.println("Unexpected successful deserialization.");
} catch (FileNotFoundException e) {
    System.err.println("Error: File not found.");
} catch (IOException e) {
    System.err.println("Error: IO Exception occurred during deserialization.");
} catch (ClassNotFoundException e) {
    System.err.println("Error: Class not found.");
}
}
}
```

5.2.4Output:



```
✓ ↗ 📄 ⚙️ 📁
=== Test Case 1: Valid Serialization & Deserialization ===
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: Deserialize from a Non-Existent File ===
Error: File not found.

=== Test Case 3: Simulating ClassNotFoundException ===
Unexpected successful deserialization.

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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.

5.3.2 Objective: The objective of this program is to develop a menu-driven Java application that allows users to add employee details, display all stored employees, and exit the program. Employee details, including ID, name, designation, and salary, are stored persistently in a file using serialization.

5.3.3 Code:

```
import java.io.*;
import java.util.*;
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name, designation;
    private double salary;

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

    public void display() {
        System.out.println("Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " + salary);
    }
}

public class EmployeeManagement {
    private static final String FILE_NAME = "employees.dat";
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Employee\n2. Display All Employees\n3. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayAllEmployees();
                break;
            case 3:
                System.out.println("Exiting...");
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice! Try again.");
        }
    }
}

private static void addEmployee() {
    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();

    Employee emp = new Employee(id, name, designation, salary);
    saveEmployeeToFile(emp);
    System.out.println("Employee added successfully!");
}

private static void saveEmployeeToFile(Employee employee) {
    List<Employee> employees = readEmployeesFromFile();
    employees.add(employee);

    try(ObjectOutputStreamoos=newObjectOutputStream(new
    FileOutputStream(FILE_NAME))) {
        for (Employee emp : employees) {
            oos.writeObject(emp);
```

```
    }  
    } catch (IOException e) {  
        System.err.println("Error: Unable to save employee details.");  
    }  
}  
  
private static void displayAllEmployees() {  
    List<Employee> employees = readEmployeesFromFile();  
    if (employees.isEmpty()) {  
        System.out.println("No employees found.");  
    } else {  
        System.out.println("\nEmployee List:");  
        for (Employee emp : employees) {  
            emp.display();  
        }  
    }  
}  
  
private static List<Employee> readEmployeesFromFile() {  
    List<Employee> employees = new ArrayList<>();  
    try (ObjectInputStream ois = new ObjectInputStream(new  
        FileInputStream(FILE_NAME))) {  
        while (true) {  
            Employee emp = (Employee) ois.readObject();  
            employees.add(emp);  
        }  
    } catch (EOFException e) {  
    } catch (FileNotFoundException e) {  
        System.err.println("Error: No employee records found.");  
    } catch (IOException | ClassNotFoundException e) {  
        System.err.println("Error: Unable to read employee details.");  
    }  
    return employees;  
}
```


5.3.4 Output:

```
input
Enter your choice: 1
Enter Employee ID: 101
Enter Name: John Doe
Enter Designation: Software Engineer
Enter Salary: 50000
Error: No employee records found.
Employee added successfully!

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

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

1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 102
Enter Name: Jane Smith
Enter Designation: Manager
Enter Salary: 75000
Employee added successfully!

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

Employee List:
Employee ID: 101, Name: John Doe, Designation: Software Engineer, Salary: 50000.0
Employee ID: 102, Name: Jane Smith, Designation: Manager, Salary: 75000.0
```

Learning Outcomes:

1. Autoboxing & Unboxing: Convert between primitive types and wrapper classes while summing integers.
2. Serialization & Deserialization: Store and retrieve objects with exception handling.
3. File Handling & Menu-Driven Applications: Manage structured data and user interactions efficiently.