

## Experiment 5

**Student Name:** Madhav Uppal

**Branch:** CSE

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

**Subject:** Java

**UID:** 22BCS50195

**Section:** IOT-641-B

**DOP:** 27/02/25

**Subject Code:** 22CSH-359

**5.1.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()).

**Objective:** To develop a Java program that demonstrates autoboxing and unboxing by calculating the sum of a list of integers. The program will include methods to parse string representations of numbers into their respective wrapper classes using Integer.parseInt(), perform arithmetic operations on wrapper class objects, and showcase Java's automatic conversion between primitive types and their corresponding wrapper classes.

### Algorithm:

1. Initialize Input Data
  - Define a list of string representations of integers.
2. Convert Strings to Integer Wrapper Objects
  - Iterate through the list of strings.
  - Use Integer.parseInt() to convert each string to an Integer object (autoboxing).
  - Store the converted values in a list of Integer objects.
3. Calculate the Sum
  - Initialize a sum variable (int sum = 0).
  - Iterate through the list of Integer objects.
  - Add each value to the sum (unboxing occurs automatically).
4. Display the Result
  - Print the calculated sum.

### Code :

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

public class AutoboxingExample {
    public static void main(String[] args) {
        String[] nums = {"10", "20", "30", "40", "50"};

        List<Integer> list = new ArrayList<>();
        for (String n : nums) {
            list.add(toInt(n));
        }

        int sum = getSum(list);

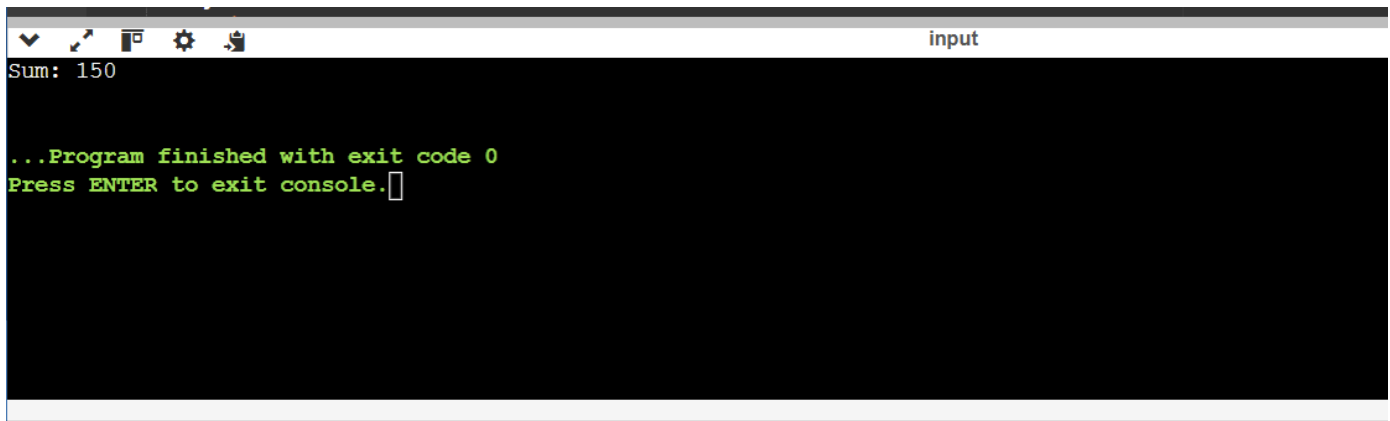
        System.out.println("Sum: " + sum);
    }

    private static Integer toInt(String s) {
```

```
        return Integer.parseInt(s);
    }

    private static int getSum(List<Integer> list) {
        int sum = 0;
        for (Integer n : list) {
            sum += n;
        }
        return sum;
    }
}
```

## Output :



## Learning outcome :

1. Understand autoboxing (automatic conversion of primitive types to their wrapper classes) and unboxing (automatic conversion of wrapper classes to primitive types) in Java.
2. Learn how to parse string representations of numbers into integer values using `Integer.parseInt()`.
3. Gain hands-on experience with wrapper classes and their role in Java's type system.
4. Practice working with lists of wrapper objects and performing arithmetic operations involving autoboxing and unboxing.
5. Strengthen understanding of iterating through collections and performing computations on their elements.

**5.2.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.

## Objective :

To develop a Java program that demonstrates serialization and deserialization by saving and retrieving a Student object. The program will:

- Serialize a Student object (containing id, name, and GPA) and save it to a file.
- Deserialize the object from the file and display its details.
- Implement exception handling to manage `FileNotFoundException`, `IOException`, and `ClassNotFoundException`.

**Code :**

```
import java.io.*;

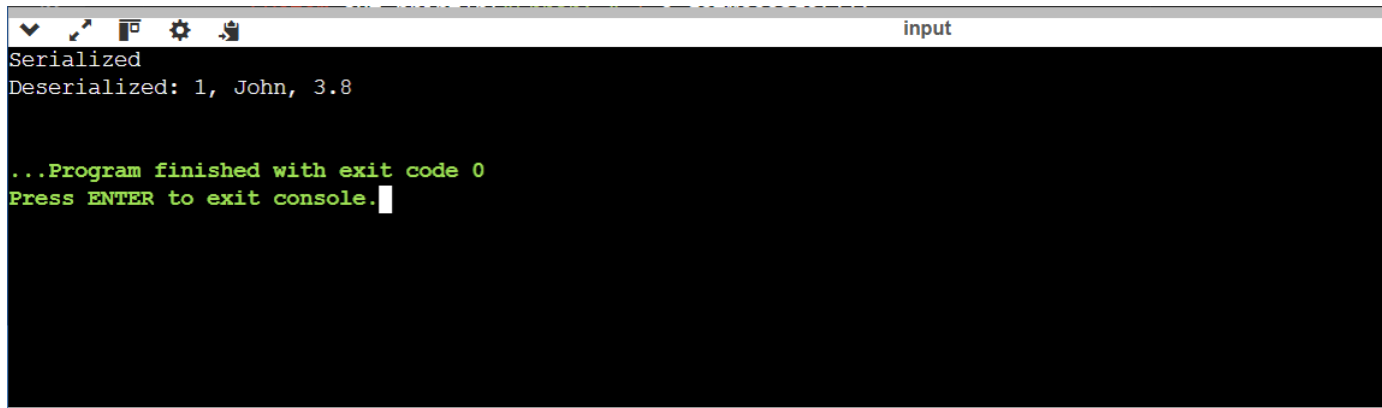
class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private double gpa;

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

    public String toString() {
        return id + ", " + name + ", " + gpa;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student(1, "John", 3.8);
        String file = "student.dat";
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(file))) {
            out.writeObject(s);
            System.out.println("Serialized");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(file))) {
            Student loaded = (Student) in.readObject();
            System.out.println("Deserialized: " + loaded);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output :



```
Serialized
Deserialized: 1, John, 3.8

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

**5.3.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.

**Objective :** To develop a menu-based Java application that allows users to manage employee records using file handling. The program will:

1. Provide a menu-driven interface with options to add an employee, display all employees, or exit.
2. Collect and store employee details (name, id, designation, and salary) in a file.
3. Retrieve and display all stored employee records when requested.
4. Implement file handling for data storage and retrieval.

**Code :**

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

class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String 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 String toString() {
        return id + ", " + name + ", " + designation + ", " + salary;
    }
}

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

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (true) {
        System.out.println("1. Add Employee\n2. Display All\n3. Exit");
        int choice = sc.nextInt();
        sc.nextLine();
        switch (choice) {
            case 1:
                addEmployee(sc);
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                System.out.println("Exiting...");
                sc.close();
                return;
            default:
                System.out.println("Invalid choice, try again.");
        }
    }
}

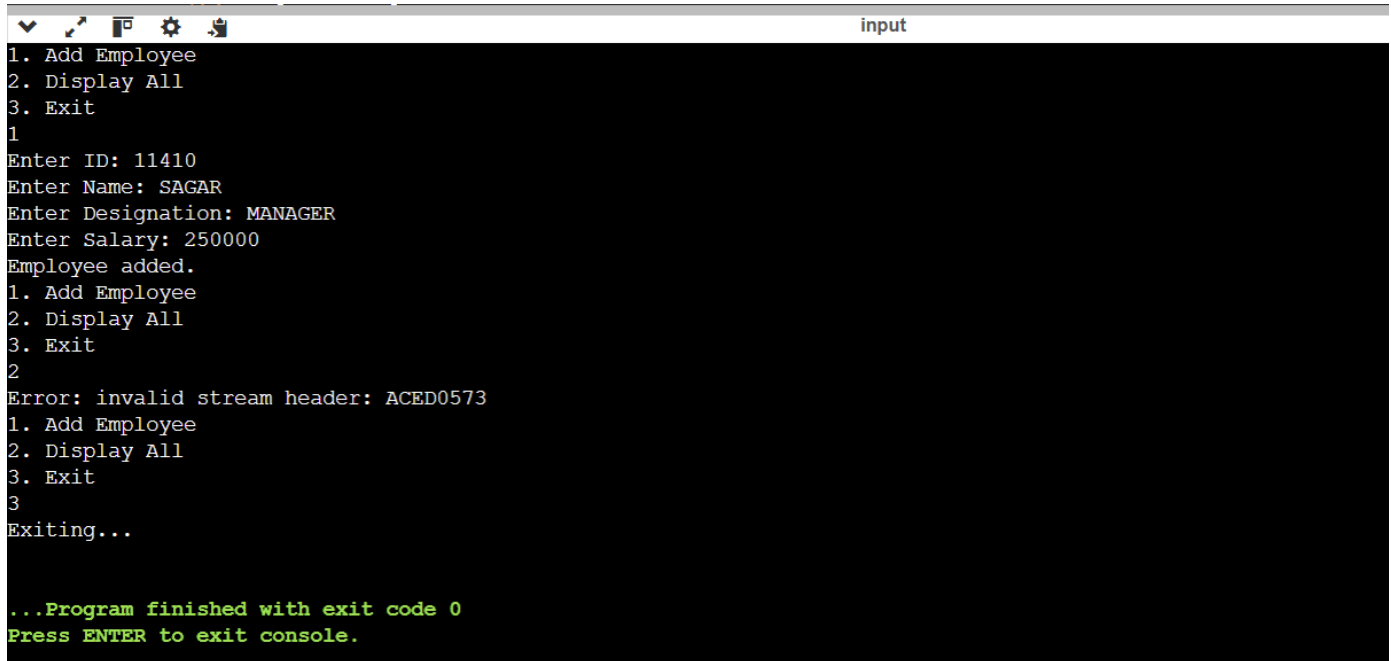
private static void addEmployee(Scanner sc) {
    System.out.print("Enter ID: ");
    int id = sc.nextInt();
    sc.nextLine();
    System.out.print("Enter Name: ");
    String name = sc.nextLine();
    System.out.print("Enter Designation: ");
    String designation = sc.nextLine();
    System.out.print("Enter Salary: ");
    double salary = sc.nextDouble();
    Employee emp = new Employee(id, name, designation, salary);

    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(FILE_NAME, true)))
    {
        out.writeObject(emp);
        System.out.println("Employee added.");
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
}

private static void displayEmployees() {
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        while (true) {
            Employee emp = (Employee) in.readObject();
            System.out.println(emp);
        }
    } catch (EOFException e) {
        System.out.println("End of file reached.");
    } catch (Exception e) {
    }
}
```

```
        System.out.println("Error: " + e.getMessage());
    }
}
```

## Output:



```
input
1. Add Employee
2. Display All
3. Exit
1
Enter ID: 11410
Enter Name: SAGAR
Enter Designation: MANAGER
Enter Salary: 250000
Employee added.
1. Add Employee
2. Display All
3. Exit
2
Error: invalid stream header: ACED0573
1. Add Employee
2. Display All
3. Exit
3
Exiting...

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

## Learning Outcome :

1. Gain hands-on experience in menu-driven programming using loops and conditionals.
2. Learn file handling in Java using FileWriter, BufferedReader, FileReader, etc.
3. Understand how to store and retrieve structured data from a file.
4. Learn to handle user input effectively using Scanner.
5. Develop skills in exception handling for file operations (IOException, FileNotFoundException).
6. Improve problem-solving abilities by designing an interactive and persistent application.