



## Experiment-5

**Name:** Gurpreet Yadav

**Branch:** BE-IT

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

**Subject:** Project Based Learning in Java

**UID :**22BET10336

**Section/Group:**22BET\_IOT-702/B

**Date of Performance:** 19/02/2025

**Subject Code:** 22ITH-359

### Problem-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()).

#### **2.Objective:**

To develop a Java program that efficiently calculates the sum of a list of integers using autoboxing and unboxing, while parsing string inputs into their respective wrapper classes (Integer) and handling invalid data through exception management.

#### **3.Code:**

```
package exp5;

import java.util.ArrayList;

import java.util.List;

public class SumOfIntegers {

    public static void main(String[] args) {

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

        // Adding integers to the list using autoboxing

        numbers.add(Integer.parseInt("10"));

        numbers.add(Integer.parseInt("20"));

        numbers.add(Integer.parseInt("30"));

        numbers.add(Integer.parseInt("40"));

        // Calculating the sum using unboxing

        int sum = 0;

        for (Integer num : numbers) {
```

```
sum += num; // Unboxing
```

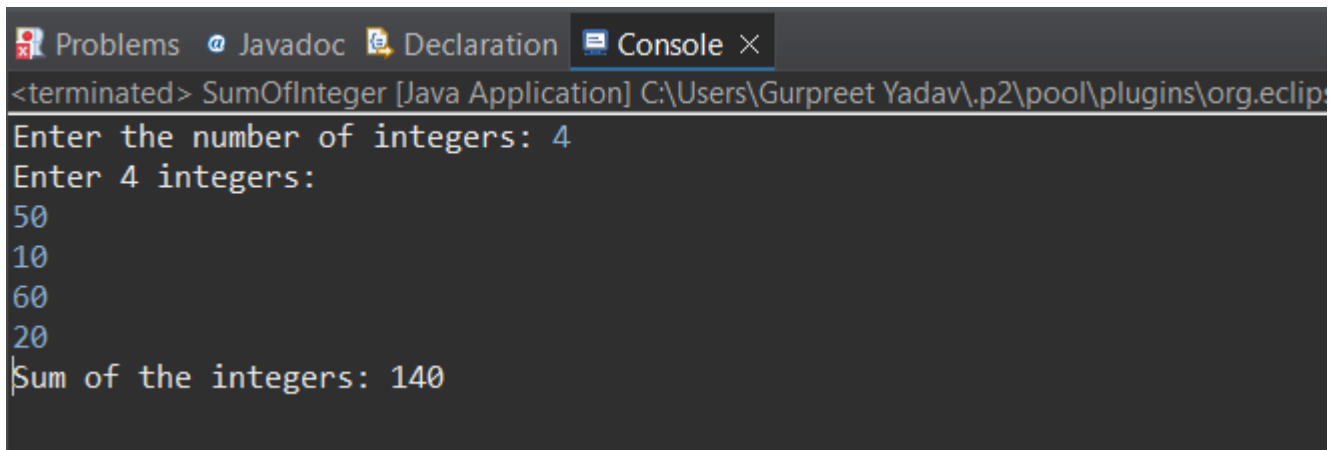
```
}
```

```
System.out.println("Sum of the integers: " + sum);
```

```
}
```

```
}
```

## 4.Output:



```
<terminated> SumOfInteger [Java Application] C:\Users\Gurpreet Yadav\.p2\pool\plugins\org.eclipse
Enter the number of integers: 4
Enter 4 integers:
50
10
60
20
Sum of the integers: 140
```

**Fig1: Sum of Integers**

## 5.Learning Outcomes:

- Understand and apply autoboxing and unboxing concepts in Java.
- Utilize wrapper classes and methods like `Integer.parseInt()` for data parsing.
- Implement exception handling to manage invalid inputs efficiently.
- Process and manage data using collections (`ArrayList`) for optimized performance.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Problem-2

### 1.Aim:

Create a Java program to serialize and deserialize a Student object .

### 2.Objective:

To develop a Java program that demonstrates the concepts of **serialization** and **deserialization** by saving the state of a `Student` object to a file and restoring it, showcasing efficient file handling and object persistence in Java.

### 3.Code: package

```
exp5;
```

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

```
class      Student
```

```
implements
```

```
Serializable {
```

```
    private      static
```

```
    final          long
```

```
serialVersionUID
```

```
= 1L;
```

```
    private      String
```

```
name;
```

```
    private int age;
```

```
    public
```

```
Student(String
```

```
name, int age) {
```

```
    this.name =
```

```
name;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

this.age =

age;

}

*@Override*

public String

toString() {

return

"Student{name="

+ name + ", age="

+ age + "}";

}

}

public class

SerializationExa

mple {

public static

void

main(String[]

args) {

Student

student = new

Student("gurpreet

", 20);



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

//Serialization

```
try
(OutputStream
eam oos = new
OutputStream
am(new
FileOutputStream
("student.ser"))) {

oos.writeObject(s
tudent);

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

} catch
(IOException e) {

e.printStackTrace

();

}
```

//

Deserialization



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

try

```
(ObjectInputStrea
```

```
m ois = new
```

```
ObjectInputStrea
```

```
m(new
```

```
FileInputStream("
```

```
student.ser")) {
```

```
    Student
```

```
deserializedStude
```

```
nt = (Student)
```

```
ois.readObject();
```

```
System.out.printl
```

```
n("Student object
```

```
deserialized: " +
```

```
deserializedStude
```

```
nt);
```

```
    } catch
```

```
(IOException |
```

```
ClassNotFoundE
```

```
xception e) {
```

```
    e.printStackTrace
```

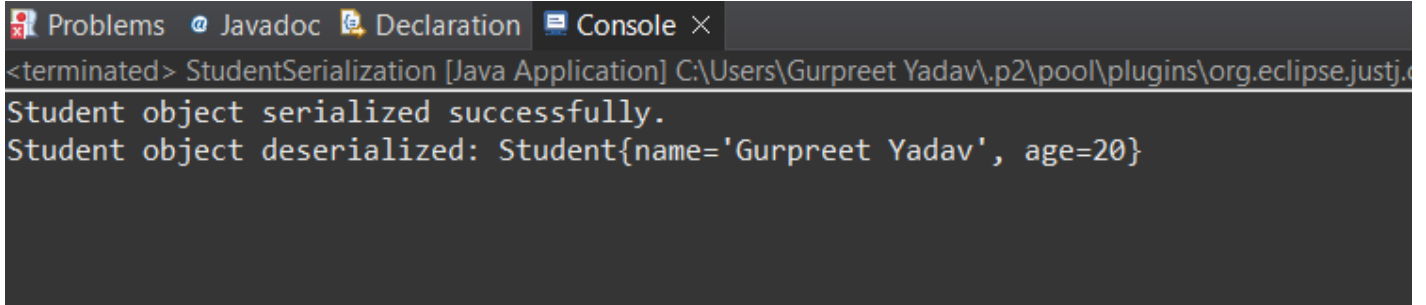
```
    ();
```

```
}
```

}

}

#### 4.Output:



```
Problems Javadoc Declaration Console ×
<terminated> StudentSerialization [Java Application] C:\Users\Gurpreet Yadav\p2\pool\plugins\org.eclipse.justj
Student object serialized successfully.
Student object deserialized: Student{name='Gurpreet Yadav', age=20}
```

Fig2: Serialisation

#### 5.Learning Outcomes:

- Understand how to save and restore objects using serialization and deserialization.
- Learn to use the Serializable interface in Java.
- Perform file handling with FileOutputStream and FileInputStream.

### Problem-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:

To develop a menu-driven Java application that allows adding employee details to a file, displaying all stored employee records, and exiting the program using efficient file handling techniques.

#### 3.Code:

```
package exp5;

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

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

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

    @Override
    public String toString() {
        return "Employee{name='" + name + "', id=" + id + ", designation='" + designation + "', salary=" + salary
        + "'}";
    }
}

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

    public static void main(String[] args) {
```





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
Scanner scanner = new Scanner(System.in);
List<Employee> employees = new ArrayList<>();

while (true) {
    System.out.println("1. Add an Employee");
    System.out.println("2. Display All Employees");
    System.out.println("3. Exit");
    System.out.print("Choose an option: ");
    int choice = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    switch (choice) {
        case 1:
            System.out.print("Enter employee name: ");
            String name = scanner.nextLine();
            System.out.print("Enter employee ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            System.out.print("Enter employee designation: ");
            String designation = scanner.nextLine();
            System.out.print("Enter employee salary: ");
            double salary = scanner.nextDouble();
            scanner.nextLine(); // Consume newline

            Employee employee = new Employee(name, id, designation, salary);
            employees.add(employee);
            saveEmployees(employees);
            System.out.println("Employee added successfully.");
            break;

        case 2:
            employees = loadEmployees();
            if (employees.isEmpty()) {
                System.out.println("No employees found.");
            } else {
                for (Employee emp : employees) {
                    System.out.println(emp);
                }
            }
            break;

        case 3:
            System.out.println("Exiting...");
            scanner.close();
            return;
    }
}
```

default:

```
System.out.println("Invalid option. Please try again.");
```

```
    }  
    }  
}
```

```
private static void saveEmployees(List<Employee> employees) {  
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {  
        oos.writeObject(employees);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

```
@SuppressWarnings("unchecked")
```

```
private static List<Employee> loadEmployees() {  
    List<Employee> employees = new ArrayList<>();  
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {  
        employees = (List<Employee>) ois.readObject();  
    } catch (IOException | ClassNotFoundException e) {  
        e.printStackTrace();  
    }  
    return employees;  
}
```

## 4.Output:

```
Employee Management System  
1. Add an Employee  
2. Display All Employees  
3. Exit  
Choose an option: 1  
Enter employee name: gurpreet  
Enter employee ID: 1001  
Enter employee designation: samrala  
Enter employee salary: 25000  
Employee added successfully.  
  
Employee Management System  
1. Add an Employee  
2. Display All Employees  
3. Exit  
Choose an option: 2  
  
List of Employees:  
Employee{name='gurpreet', id=1001, designation='samrala', salary=25000.0}  
  
Employee Management System  
1. Add an Employee  
2. Display All Employees  
3. Exit  
Choose an option:
```

Fig3: EmployeeManagementApp



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 5.Learning Outcomes:

- Understand how to create menu-driven applications in Java using control statements.
- Perform file handling operations to read from and write employee details to a file.
- Implement object storage and retrieval for managing multiple employee records.
- Handle user input efficiently using classes like `Scanner`.
- Apply loops and conditional statements to manage application flow and user choices.