

## Experiment 5

**Student Name:** Dip Biswas

**Branch:** BE-IT

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

**Subject Name:** Project based learning in Java

**UID:** 22BET10001

**Section/Group:** 22BET\_IOT-601

**Date of Performance:** 18-02-25

**Subject Code:** 22ITH-359

**Problem 1:** 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()).

### 1. Objectives:

- Demonstrate the use of autoboxing (automatic conversion of primitive types to their wrapper class objects) and unboxing (automatic conversion of wrapper class objects to primitive types) while performing arithmetic operations.
- Implement methods to convert string representations of numbers into their respective wrapper classes using methods like Integer.parseInt().
- Sum the list of integers while ensuring the use of autoboxing and unboxing to demonstrate their effect.

### 2. Code:

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

public class SumOfIntegers {

    // Method to parse a string into an Integer
    public static Integer parseStringToInteger(String str) { try
        { return Integer.parseInt(str);
        } catch (NumberFormatException e) {
            System.out.println("Invalid input: " + str + " is not a valid integer."); return
            null; // Return null if parsing fails
        }
    }

    // Method to calculate the sum of a list of integers public static
    int calculateSum(List<Integer> integers) {
        int sum = 0;
        for (Integer number : integers) {
            // Unboxing: Integer to int sum +=
            number;
        }
        return sum;
    }
}
```

```
public static void main(String[] args)
{
    Scanner scanner = new
    Scanner(System.in);
    List<Integer> integerList = new ArrayList<>();

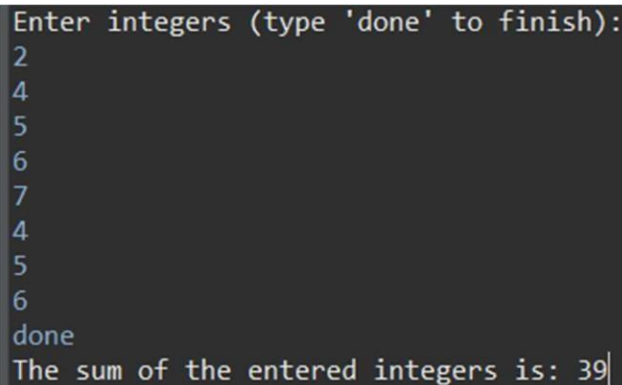
    System.out.println("Enter integers (type 'done' to finish):");

    while (true) {
        String input = scanner.nextLine();
        if (input.equalsIgnoreCase("done")) {
            break; // Exit the loop if the user types 'done'
        }
        Integer number = parseStringToInteger(input);
        if (number != null) {
            // Autoboxing: int to Integer
            integerList.add(number);
        }
    }

    // Calculate the sum of the integers in the list
    int sum = calculateSum(integerList);
    System.out.println("The sum of the entered integers is: " + sum);

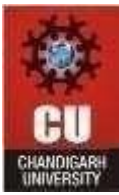
    scanner.close();
}
}
```

### 3. Output:



```
Enter integers (type 'done' to finish):
2
4
5
6
7
4
5
6
done
The sum of the entered integers is: 39|
```

**Fig:1** Sum of elements of array



## 4. Learning Outcomes:

- Learn how Java automatically converts primitive types to their corresponding wrapper classes (autoboxing) and vice versa (unboxing).
- Recognize the importance of autoboxing/unboxing in arithmetic operations and collections like `ArrayList<Integer>`.
- Gain experience in working with Java wrapper classes (Integer, Double, etc.).
- Learn how to convert string representations of numbers into their respective wrapper types using methods like `Integer.parseInt()`.

**Problem 2:** 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.

## 1. Objectives:

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

## 2. Code:

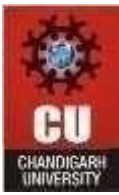
```
import java.io.*;
import java.util.Scanner;

class Student implements Serializable {
    private static final long serialVersionUID = 1L; // For serialization
    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 int getId() { return
        id;
    }

    public String getName() { return
        name;
    }
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public double getGpa()
{ return gpa;
}

@Override
public String toString() {
    return "Student ID: " + id + ", Name: " + name + ", GPA: " + gpa;
}
}

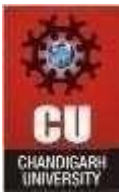
public class StudentSerialization {

    private static final String FILE_NAME = "student.ser";

    public static void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(FILE_NAME))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (FileNotFoundException e)
        { System.out.println("File not found: " +
        e.getMessage());
        } catch (IOException e) {
            System.out.println("IOException occurred: " + e.getMessage());
        }
    }

    public static Student deserializeStudent()
    { Student student = null;
        try (ObjectInputStream ois = new
            ObjectInputStream(new FileInputStream(FILE_NAME)))
        { student = (Student) ois.readObject();
            System.out.println("Student object deserialized successfully.");
        } catch (FileNotFoundException e)
        { System.out.println("File not found: " +
        e.getMessage());
        } catch (IOException e) {
            System.out.println("IOException occurred: " + e.getMessage());
        } catch (ClassNotFoundException e)
        { System.out.println("Class not found: " + e.getMessage());
        }
        return student;
    }

    public static void main(String[] args)
    { Scanner scanner = new
        Scanner(System.in); int choice;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
do {
    System.out.println("\nMenu:");
    System.out.println("1. Serialize Student");
    System.out.println("2. Deserialize Student");
    System.out.println("3. Exit");
    System.out.print("Enter your choice: ");
    choice = scanner.nextInt();
    scanner.nextLine(); // Consume newline

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

        Student student = new Student(id, name, gpa);
        serializeStudent(student);
        break;

        case 2:
            Student deserializedStudent = deserializeStudent();
            if (deserializedStudent != null) {
                System.out.println("Deserialized Student Details: " + deserializedStudent);
            }
            break;

        case 3:
            System.out.println("Exiting the program.");
            break;

        default:
            System.out.println("Invalid choice. Please try again.");
    }
} while (choice != 3);

scanner.close();
}
```

### 3. Output:

```
Menu:
1. Serialize Student
2. Deserialize Student
3. Exit
Enter your choice: 1
Enter Student ID: 10026
Enter Student Name: Arsh
Enter Student GPA: 8.3
Student object serialized successfully.

Menu:
1. Serialize Student
2. Deserialize Student
3. Exit
Enter your choice: 3
Exiting the program.
```

**Fig:2** Student management system using serialization & deserialization

### 4. Learning Outcomes:

- Understand the concept of object serialization and deserialization in Java.
- Learn how to convert a Java object into a byte stream and save it to a file.
- Gain knowledge of reading and converting a byte stream back into a Java object.

**Problem 3:** 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.

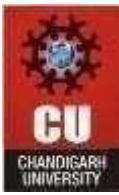
#### 1. Objectives:

1. Create a menu with options to add an employee, display all employees, or exit.
2. Add employee details (name, id, designation, salary) and store them in a file.
3. Display all employee details from the file.

#### 2. Code:

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

class EmployeeRecord implements Serializable {
    private static final long serialVersionUID = 1L; // For serialization
    private String name;
    private int id;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
private String designation;
private double salary;

public EmployeeRecord(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 ID: " + id + ", Name: " + name + ", Designation: " + designation +
        ", Salary: " + salary;
}

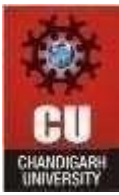
}

public class EmployeeManagement {

    private static final String FILE_NAME = "employees.ser";

    public static void addEmployee(EmployeeRecord employee) {
        List<EmployeeRecord> employees = readEmployees();
        employees.add(employee);
        try (ObjectOutputStream oos = new ObjectOutputStream(new
            FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
            System.out.println("Employee added successfully.");
        } catch (IOException e) {
            System.out.println("Error saving employee: " + e.getMessage());
        }
    }

    public static List<EmployeeRecord> readEmployees()
    {
        List<EmployeeRecord> employees = new
            ArrayList<>();
        try (ObjectInputStream ois = new ObjectInputStream(new
            FileInputStream(FILE_NAME))) {
            employees = (List<EmployeeRecord>) ois.readObject();
        } catch (FileNotFoundException e) {
            // File not found, return empty list
        } catch (IOException | ClassNotFoundException e)
        {
            System.out.println("Error reading employees: " + e.getMessage());
        }
        return employees;
    }
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public static void displayEmployees()
{ List<EmployeeRecord> employees =
readEmployees(); if (employees.isEmpty()) {
    System.out.println("No employees found.");
} else {
    System.out.println("Employee Details:");
    for (EmployeeRecord employee : employees)
        { System.out.println(employee);
        }
    }
}

public static void main(String[] args)
{ Scanner scanner = new
Scanner(System.in); int choice;

do {
    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: ");
    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 Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();

        EmployeeRecord employee = new EmployeeRecord(name, id, designation,
salary);
        addEmployee(employee);
        break;

        case 2:
            displayEmployees();
            break;
```

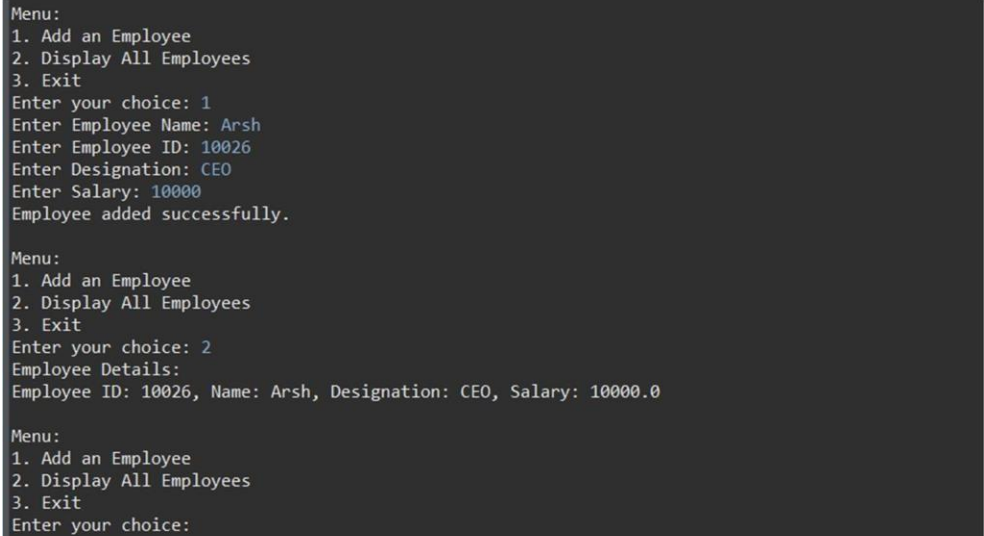


```
        case 3:
            System.out.println("Exiting the program.");
            break;

        default:
            System.out.println("Invalid choice. Please try again.");
    }
} while (choice != 3);

scanner.close();
}
```

### 3. Output:



```
Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee Name: Arsh
Enter Employee ID: 10026
Enter Designation: CEO
Enter Salary: 10000
Employee added successfully.

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2
Employee Details:
Employee ID: 10026, Name: Arsh, Designation: CEO, Salary: 10000.0

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

**Fig:3** Employee Details

### 4. Learning Outcomes:

- Learn how to create a menu-driven application in Java.
- Understand how to gather user input and store it in a file.
- Gain experience in reading from and displaying data stored in a file.
- Develop skills in managing application flow with user-driven options.