**DEPARTMENT OF****COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

**Experiment 5****Student Name:** Arun**Branch:** BE-IT**Semester:** 6<sup>th</sup>**Subject:** Project Based Learning in JAVA with Lab**UID:** 22BET10320**Section/Group:** 22BET\_IOT-701/A**Date of Performance:** 18/02/2025**Subject Code:** 22ITH-359**Problem-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 implement a Java program that calculates the sum of integers using autoboxing and unboxing, while converting string inputs into wrapper class objects.

**Code:**

```
package sum;
import java.util.*;

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

        System.out.println("Enter numbers separated by space (press Enter to finish):");
        String input = scanner.nextLine();
        String[] tokens = input.split(" ");

        for (String token : tokens) {
            try {
                Integer num = Integer.parseInt(token); // Autoboxing from int to Integer
                numbers.add(num);
            } catch (NumberFormatException e) {
                System.out.println("Invalid number: " + token);
            }
        }

        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Unboxing from Integer to int
        }
        System.out.println("Sum of numbers: " + sum);
        scanner.close();
    }
}
```

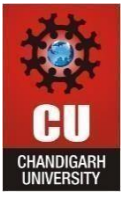
### Output

```
Enter numbers separated by space (press Enter to finish):  
2 2 1 0 3 2 0  
Sum of numbers: 10
```

Fig.1: Adding Numbers

### Learning Outcomes

- Understand autoboxing and unboxing in Java with wrapper classes.
- Learn to convert strings into integers using `Integer.parseInt()`.
- Implement list operations with `ArrayList<Integer>`.
- Handle user input and exceptions for number parsing.



## Problem-2

**Aim:** Create a Java program to serialize and deserialize a student object.

**Objective:** To develop a Java program that demonstrates object serialization and deserialization by saving and retrieving a student object using file handling.

**Code:**

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

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private int age;
    private String id;
    private String universityName; // Added field for University Name

    public Student(String name, int age, String id, String universityName) {
        this.name = name;
        this.age = age;
        this.id = id;
        this.universityName = universityName;
    }

    public void display() {
        System.out.println("Student Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("UID: " + id);
        System.out.println("University Name: " + universityName);
    }
}

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

        System.out.println("Enter student name:");
        String name = scanner.nextLine();

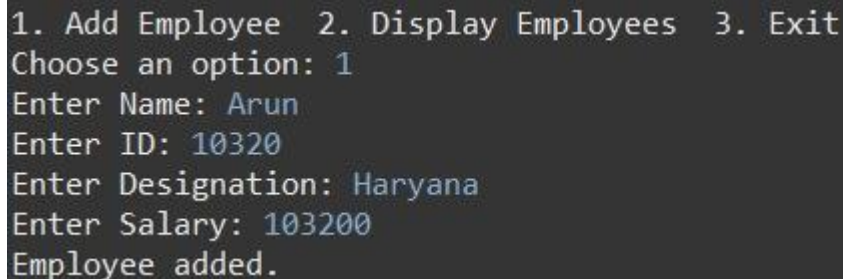
        System.out.println("Enter student age:");
        int age = scanner.nextInt();

        scanner.nextLine(); // Consume newline
        System.out.println("Enter student UID:");
        String id = scanner.nextLine();
        System.out.println("Enter University Name:");
        String universityName = scanner.nextLine();
        Student student = new Student(name, age, id, universityName);
    }
}
```

```
// Serialization try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("student.ser"))) {
oos.writeObject(student);
    System.out.println("Student object serialized successfully.");
} catch (IOException e) {
e.printStackTrace();
}
// Deserialization
try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"))) {
    Student deserializedStudent = (Student) ois.readObject();
System.out.println("\nDeserialized Student Object:");
deserializedStudent.display();
    } catch (IOException | ClassNotFoundException e)
{
    e.printStackTrace();
}

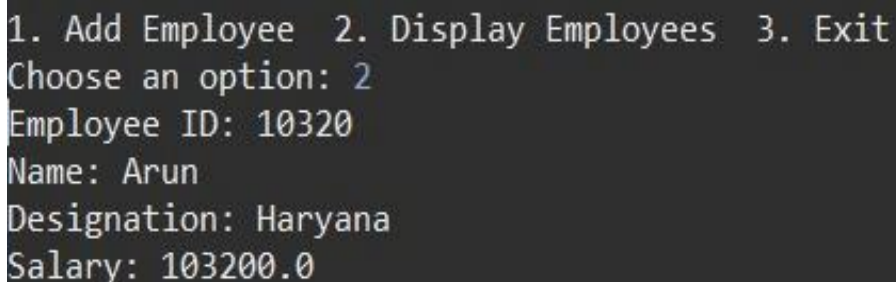
    scanner.close();
}
}
```

## Output



```
1. Add Employee 2. Display Employees 3. Exit
Choose an option: 1
Enter Name: Arun
Enter ID: 10320
Enter Designation: Haryana
Enter Salary: 103200
Employee added.
```

Fig.1: Add Student Details



```
1. Add Employee 2. Display Employees 3. Exit
Choose an option: 2
Employee ID: 10320
Name: Arun
Designation: Haryana
Salary: 103200.0
```

Fig.2: Display Student Details

## Learning Outcomes

- Understand the concept of serialization and how to store objects in a file.
- Learn how to deserialize an object to restore its state.
- Implement file handling in Java using ObjectOutputStream and ObjectInputStream.
- Gain practical experience in working with Serializable interface in Java.

## Problem-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 for managing employee records using file handling and object serialization.

**Code:**

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

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

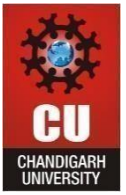
    public Employee(String name, int id, String designation, double salary) {
        this.name = name;
        this.id = id;
        this.designation = designation;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee ID: " + id + "\nName: " + name + "\nDesignation: " + designation + "\nSalary: " + salary + "\n";
    }
}

public class EmployeeManagement {
    private static final String FILE_NAME = "employees.ser";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Employee> allEmployees = loadEmployees(); // Load all employees from file
        List<Employee> sessionEmployees = new ArrayList<>(); // Store employees added in this session

        while (true) {
```



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

```
System.out.println("\nMenu:");
System.out.println("1. Add an Employee");
System.out.println("2. Display Recently Added Employees");
System.out.println("3. Display All Employees");
System.out.println("4. 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 Designation: ");
    String designation = scanner.nextLine();

    System.out.print("Enter Salary: ");
    double salary = scanner.nextDouble();
    scanner.nextLine(); // Consume newline

    Employee emp = new Employee(name, id, designation, salary);
    sessionEmployees.add(emp); // Add to session list
    allEmployees.add(emp); // Add to full list

    saveEmployees(allEmployees); // Save full list to file
    System.out.println("Employee added successfully.");
    break;
case 2:
    System.out.println("\nRecently Added Employees (This Session):");
    if (sessionEmployees.isEmpty()) {
        System.out.println("No employees added in this session.");
    } else {
        for (Employee e : sessionEmployees) {
            System.out.println(e);
        }
    }
    break;
case 3:
    System.out.println("\nAll Employees:");
    if (allEmployees.isEmpty()) {
        System.out.println("No employees found.");
    } else {
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        for (Employee e : allEmployees) {
            System.out.println(e);
        }
    }
    break;

    case 4:
        System.out.println("Exiting application.");
        scanner.close();
        System.exit(0);
        break;

default:
    System.out.println("Invalid choice. 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) {
        System.err.println("Error saving employee data: " + e.getMessage());
    }
}

@SuppressWarnings("unchecked")
private static List<Employee> loadEmployees() {
    File file = new File(FILE_NAME);
    if (!file.exists()) {
        return new ArrayList<>();
    }

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
        return (List<Employee>) ois.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Error loading employee data: " + e.getMessage());
        return new ArrayList<>();
    }
}
}
```

## Output

```
Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 10320
Enter Employee Name: Arun
Enter Designation: UI
Enter Salary: 823784
Employee added successfully!
```

Fig.1: Add Employees

```
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
```

Fig.2: Display All Employees

## Learning Outcomes

- Understand file handling and object serialization in Java.
- Implement menu-driven programming and user input handling.
- Apply OOP concepts like encapsulation and constructors.
- Handle exceptions for file I/O operations.



