

## Experiment 5

**Student Name: Kushagar**

**Branch: CSE**

**Semester:**

**Subject: PBLJ**

**UID: 22BCS10429**

**Section: 22BCS\_IOT-642/A**

**DOP:04/03/25**

**Subject Code:22CSH-359**

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:** Demonstrate **autoboxing** and **unboxing** in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

### 3. Algorithm:

#### Step 1: Initialize the Program

1. Start the program.
2. Import ArrayList and List classes.
3. Define the AutoboxingExample class.

#### Step 2: Convert String Array to Integer List

1. Define the method parseStringArrayToIntegers(String[] strings).
2. Create an empty ArrayList<Integer>.
3. Iterate through the string array:
  - o Convert each string to an Integer using Integer.parseInt(str).
  - o Add the integer to the list (**autoboxing** happens here).
4. Return the list of integers.

#### Step 3: Calculate the Sum of Integers

1. Define the method calculateSum(List<Integer> numbers).
2. Initialize a variable sum to 0.
3. Iterate through the list:
  - o Extract each integer (**unboxing** happens here).
  - o Add it to sum.
4. Return the total sum.

#### Step 4: Execute Main Function

1. Define main(String[] args).
2. Create a string array with numeric values.
3. Call parseStringArrayToIntegers() to convert it into a list of integers.
4. Call calculateSum() to compute the sum.
5. Print the result.

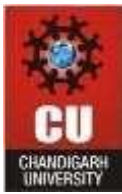
#### Step 5: Terminate the Program

1. End the execution.

### 4. Code:

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

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



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

```
String[] numberStrings = {"10", "20", "30", "40", "50", "70", "154"};

List<Integer> numbers = parseStringArrayToIntegers(numberStrings); int

sum = calculateSum(numbers);

System.out.println("The sum of the numbers is: " + sum); }

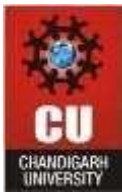
public static List<Integer> parseStringArrayToIntegers(String[] strings) {
    List<Integer> integerList = new ArrayList<>();
    for (String str : strings) { integerList.add(Integer.parseInt(str));
    }
    return integerList;
}

public static int calculateSum(List<Integer> numbers) { int
    sum = 0;
    for (Integer num : numbers) { sum
        += num;
    }
    return sum;
}
}
```

## 5. Output:

A screenshot of an IDE's console window. The window has tabs for 'Javadoc', 'Coverage', 'Gradle Tasks', 'Console', and 'Gradle Executions'. The 'Console' tab is active, showing the output of a Java application. The text in the console is: '<terminated> classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (03-Mar-2025, 11:57: The sum of the numbers is: 374'. The output is displayed in a dark-themed window with a light-colored font.

```
<terminated> classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (03-Mar-2025, 11:57:
The sum of the numbers is: 374
```



## Code 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 using exception handling.

**2. Objective:** The objective is to serialize and deserialize a Student object, store and retrieve its id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

### 3. Algorithm:

#### Step 1: Initialize the Program

1. Start the program.
2. Import the necessary classes (java.io.\*).
3. Define a Student class implementing Serializable.
4. Declare attributes:
  - id (int) ◦ name (String) ◦ gpa (double)
5. Define a constructor to initialize Student objects.
6. Override toString() to display student details.

#### Step 2: Define the Serialization Method

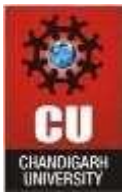
1. Create serializeStudent(Student student).
2. Use a try-with-resources block to create an ObjectOutputStream:
  - Open a FileOutputStream to write to student.ser.
  - Write the Student object to the file using writeObject().
3. Handle exceptions:
  - FileNotFoundException → Print error message.
  - IOException → Print error message.
4. Print a success message if serialization is successful.

#### Step 3: Define the Deserialization Method

1. Create deserializeStudent().
2. Use a try-with-resources block to create an ObjectInputStream:
  - Open a FileInputStream to read student.ser.
  - Read the Student object using readObject().
3. Handle exceptions:
  - FileNotFoundException → Print error message.
  - IOException → Print error message.
  - ClassNotFoundException → Print error message.
4. Print the deserialized student details.

#### Step 4: Execute Main Function

1. Define main(String[] args).
2. Create a Student object with sample data.



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

3. Call `serializeStudent()` to save the object.
4. Call `deserializeStudent()` to read and display the object.

#### **Step 5: Terminate the Program**

1. End execution.

#### **4. Implementation 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;
    }

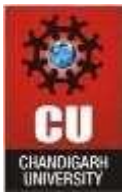
    @Override
    public String toString() { return "Student{id=" + id + ", name=" + name
        + ", cgpa=" + gpa + "}";
    }
}
```

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

```
    public static void main(String[] args) {
        Student student = new Student(1, "Kushagar", 7.3);
        serializeStudent(student);
        deserializeStudent();
    }
```

```
    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.err.println("File not found: " + e.getMessage());
    } catch (IOException e) {
        System.err.println("IOException occurred: " + e.getMessage());
    }
}
```

```
    public static void deserializeStudent() { try (ObjectInputStream ois = new
        ObjectInputStream(new FileInputStream(FILE_NAME)))
    {
        Student student = (Student) ois.readObject();
        System.out.println("Deserialized Student: " + student);
    } catch (FileNotFoundException e) {
```



**DEPARTMENT OF**

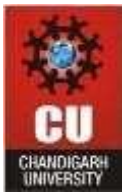
**COMPUTER SCIENCE & ENGINEERING**

*Discover. Learn. Empower.*

```
        System.err.println("File not found: " + e.getMessage());
    } catch (IOException e) {
        System.err.println("IOException occurred: " + e.getMessage()); }
    catch (ClassNotFoundException e) {
        System.err.println("Class not found: " + e.getMessage());
    }
}
}
```

## 5.Output

```
@ Javadoc Coverage Gradle Tasks Console × Gradle Executions
<terminated> classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (03-Mar-2025, 11:59:37 pm – 11:59:38 pm)
Student object serialized successfully.
Deserialized Student: Student{id=1, name='Kushagar', cgpa=7.3}
```



## Code 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:** The objective is to develop a menu-based Java application that allows users to **add employee details, store them in a file, and display all stored employee records**, with an option to exit the program.

### **3. Algorithm:**

#### **Step 1: Initialize the Program**

1. Start the program.
2. Import `java.util.*` and `java.util.concurrent.*` for thread handling.
3. Define a class `TicketBookingSystem` with:
  - A `List<Boolean>` representing seat availability (true for available, false for booked).
  - A synchronized method `bookSeat(int seatNumber, String passengerName)` to ensure thread safety.

#### **Step 2: Implement Seat Booking Logic**

1. Define `bookSeat(int seatNumber, String passengerName)`:
  - If the seat is available (true), mark it as booked (false).
  - Print confirmation: "Seat X booked successfully by Y".
  - If already booked, print: "Seat X is already booked."

#### **Step 3: Define Booking Threads**

1. Create a class `PassengerThread` extending `Thread`:
  - Store passenger name, seat number, and booking system reference.
  - Implement `run()` method to call `bookSeat()`.

#### **Step 4: Assign Thread Priorities**

1. Create VIP and Regular passenger threads.
2. Set higher priority for VIP passengers using `setPriority(Thread.MAX_PRIORITY)`.
3. Set default priority for regular passengers.

#### **Step 5: Handle User Input & Simulate Booking**

1. In `main()`, create an instance of `TicketBookingSystem`.
2. Accept number of seats and bookings from the user.
3. Create multiple `PassengerThread` instances for VIP and regular passengers.
4. Start all threads using `start()`.

#### **Step 6: Synchronization & Preventing Double Booking**

1. Use the synchronized keyword in `bookSeat()` to ensure only one thread accesses it at a time.
2. Ensure thread execution order by assigning higher priority to VIP threads.

#### **Step 7: Display Final Booking Status**

1. After all threads finish execution, display the list of booked seats.
2. End the program with a message: "All bookings completed successfully."

#### 4. Implementation 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;
    }

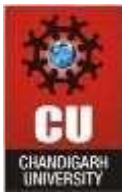
    @Override
    public String toString() {
        return "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " + salary;
    }
}

public class classwork3 {
    private static final String FILE_NAME = "employees.ser";
    private static List<Employee> employees = new ArrayList<>();

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

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

    public static void displayAllEmployees() {
        loadEmployees();
        if (employees.isEmpty()) {
```



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

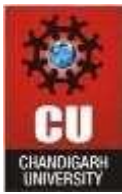
```
System.out.println("No employees found.");
} else {
for (Employee employee : employees) {
System.out.println(employee);
}}}
```

```
private static void saveEmployees() {
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
oos.writeObject(employees);
} catch (IOException e) {
System.err.println("Error saving employees: " + e.getMessage());
}}
@SuppressWarnings("unchecked")
private static void loadEmployees() {
try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME)))
{
employees = (List<Employee>) ois.readObject();
} catch (FileNotFoundException e) {
employees = new ArrayList<>();
} catch (IOException | ClassNotFoundException e) {
System.err.println("Error loading employees: " + e.getMessage());
}}}
```

```
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
while (true) {
System.out.println("\nEmployee Management System");
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: ");
int choice = scanner.nextInt();
scanner.nextLine();
```

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





**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

### Output:

classwork3 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (04-Mar-2025, 12:02:41 am) [pid: 34520]

Employee Management System

1. Add an Employee
2. Display All Employees
3. Exit

Enter your choice: 1

Enter Employee ID: 15

Enter Employee Name: Kush

Enter Designation: Intern

Enter Salary: 20000

Employee added successfully!

Employee Management System

1. Add an Employee
2. Display All Employees
3. Exit

Enter your choice: 1

Enter Employee ID: 16

Enter Employee Name: abc

Enter Designation: senior developer

Enter Salary: 68000

Employee added successfully!

Employee Management System

1. Add an Employee
2. Display All Employees
3. Exit

Enter your choice: 2

Employee ID: 15, Name: Kush, Designation: Intern, Salary: 20000.0

Employee ID: 16, Name: abc, Designation: senior developer, Salary: 68000.0

Employee Management System

1. Add an Employee
2. Display All Employees
3. Exit

Enter your choice:

### 5. Learning Outcomes:

- Understand file handling and serialization in Java to store and retrieve objects persistently.
- Learn how to implement a menu-driven console application using loops and conditional statements.
- Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.
- Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
- Develop skills in list manipulation and user input handling using ArrayList and Scanner.