Experiment 5

Student Name: Anshuman Sharma
UID: 22BCS10676
Branch: BE-CSE
Section/Group: 22BCS_IOT-639(B)
Semester: 6th
Date of Performance: 5/3/25
Subject Name: PBLJ
Subject Code: 22CSH-359

1. Aim: Use of wrapper classes in Java- Integer, Character, Long, Boolean. Autoboxing and Unboxing. Byte stream, Character stream, Object serialization, cloning. Introduce lambda syntax, functional interfaces, method references, stream operations, sorting, filtering, mapping, reducing.

2. Programming Problems:

a) 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()).

- Implementation/Code:

```
import java.util.*;

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

        System.out.println("Enter integers (type 'done' to finish):");
        while (scanner.hasNext()) {
            String input = scanner.next();
            if (input.equalsIgnoreCase("done")) {
                break;
            }
            try {
                numbers.add(Integer.parseInt(input)); //
            Autoboxing } catch (NumberFormatException e) {
                System.out.println("Invalid input! Please enter an integer.");
```

```
            }
        }

        int sum = 0;
        for (Integer num : numbers) {
            sum += num; // Unboxing
        }

        System.out.println("Sum of entered integers: " + sum);
        scanner.close();
    }
}
```

- Output:

```
Enter integers (type 'done' to finish):
10
30
20
50
done
Sum of entered integers: 660

Process finished with exit code 0
```

b)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.

- Implementation/Code:

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

```
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 void display()
      { System.out.println("Student ID: " +
      id); System.out.println("Name: " +
      name); System.out.println("GPA: " +
      gpa);
   }
}

public class partII {
   private static final String FILE_NAME = "student.dat";

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

      // Taking user input
      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);

      // Serialize Student object
      try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME)))
{
```

```
            System.out.println("Student data serialized
        successfully."); } catch (IOException e) {
            System.err.println("Error during serialization: " + e.getMessage());
        }

        // Deserialize Student object
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME)))
            { Student deserializedStudent = (Student) ois.readObject();
            System.out.println("\nDeserialized Student Details:");
            deserializedStudent.display();
        } catch (FileNotFoundException e) {
            System.err.println("File not found! Run serialization first.");
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error during deserialization: " + e.getMessage());
        }

        scanner.close();
    }
}
```

- Output:

```
Enter Student ID: 23456
Enter Student Name: Ichigo
Enter Student GPA: 8.94
Student data serialized successfully.

Deserialized Student Details:
Student ID: 23456
Name: Ichigo
GPA: 8.94

Process finished with exit code 0
```

b) 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.

- 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;
   }

   public void display()
      { System.out.println("\nEmployee ID: " + id);
      System.out.println("Name: " + name);
      System.out.println("Designation: " + designation);
      System.out.println("Salary: " + salary);
   }
}

public class partIII {
   private static final String FILE_NAME = "employees.dat";
   private static List<Employee> employeeList = new ArrayList<>();
   private static Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
      loadEmployees(); // Load existing employees from file
      while (true) {
         System.out.println("\n--- Employee 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(); // Consume newline

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                displayEmployees();
                break;
            case 3:
                saveEmployees(); // Save before exiting
                System.out.println("Exiting... Goodbye!");
                return;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    }
}

private static void addEmployee()
    { System.out.print("Enter Employee ID:
    "); int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    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 emp = new Employee(id, name, designation, salary);
    employeeList.add(emp);
    System.out.println("Employee added successfully.");
}
```

```
    private static void displayEmployees() {
        if (employeeList.isEmpty()) {
            System.out.println("No employees
        found."); } else {
            System.out.println("\n--- Employee List ---");
            for (Employee emp : employeeList) {
                emp.display();
            }
        }
    }

    private static void saveEmployees() {
        try (ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
            oos.writeObject(employeeList);
            System.out.println("Employee data saved successfully.");
        } catch (IOException e) {
            System.err.println("Error saving employee data: " + e.getMessage());
        }
    }

    private static void loadEmployees() {
        File file = new File(FILE_NAME);
        if (!file.exists()) return; // No previous data

        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            employeeList = (List<Employee>) ois.readObject();
        } catch (FileNotFoundException e) {
            System.err.println("Employee data file not
        found."); } catch (IOException |
        ClassNotFoundException e) {
            System.err.println("Error loading employee data: " + e.getMessage());
        }
    }
}
```

```
--- Employee Management System ---
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 45567
Enter Employee Name: yori
Enter Designation: Frontend developer
Enter Salary: 500000
Employee added successfully.

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

3. Learning Outcome:

- Understand the use of wrapper classes (Integer, Character, Long, Boolean) and concepts of autoboxing & unboxing in Java.
- Learn byte & character streams, object serialization, and cloning for efficient data handling.
- Explore lambda expressions, functional interfaces, and method references for functional programming.
- Master stream operations like sorting, filtering, mapping, and reducing for effective data processing.