Experiment 4

Student Name: Aditya Nandal **UID:22BCS14583**
Branch: CSE **Section:22BCS_IOT-642A**
Semester: 6th **DOP:25/02/25**
Subject: Project Based Learning in Java **Subject Code:22CSH-359**

Code 1:

1. **Aim:** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

2. **Objective:** Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

3. **Code:**

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

class Employee {
    int id;
    String name;
    double salary;

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

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

```
public class EmployeeManager {
    private ArrayList<Employee> employees = new ArrayList<>();
    private Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        new EmployeeManager().run();
    }

    public void run() {
        while (true) {
            System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove Employee\n4. Search Employee\n5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();  // consume newline

            switch (choice) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice, please try again.");
            }
        }
    }

    private void addEmployee() {
        System.out.print("Enter ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();  // consume newline
        System.out.print("Enter Name: ");
```

```
        String name = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully.");
    }

    private void updateEmployee() {
        System.out.print("Enter ID of the employee to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();  // consume newline
        Employee emp = findEmployee(id);
        if (emp == null) {
            System.out.println("Employee not found.");
            return;
        }
        System.out.print("Enter new Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter new Salary: ");
        double salary = scanner.nextDouble();
        emp.name = name;
        emp.salary = salary;
        System.out.println("Employee updated successfully.");
    }

    private void removeEmployee() {
        System.out.print("Enter ID of the employee to remove: ");
        int id = scanner.nextInt();
        Employee emp = findEmployee(id);
        if (emp == null) {
            System.out.println("Employee not found.");
            return;
        }
        employees.remove(emp);
```

```
        System.out.println("Employee removed successfully.");
    }

    private void searchEmployee() {
        System.out.print("Enter ID of the employee to search: ");
        int id = scanner.nextInt();
        Employee emp = findEmployee(id);
        System.out.println(emp == null ? "Employee not found." : emp);
    }

    private Employee findEmployee(int id) {
        return employees.stream().filter(e -> e.id == id).findFirst().orElse(null);
    }
}
```

4. **Output:**

```
input

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter your choice: 1
Enter ID: 145
Enter Name: Aditya
Enter Salary: 78,000
Employee added successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter your choice: 1
Enter ID: 835
Enter Name: Nandal
Enter Salary: 90,000
Employee added successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter your choice: 2
Enter ID of the employee to update: 145
Enter new Name: Aaditya
Enter new Salary: 98,000
Employee updated successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
```

Code 2:

1. **Aim:** Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

2. **Objective:** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

3. **Code:**

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

class Card {
    String symbol;
    String value;

    Card(String symbol, String value) {
        this.symbol = symbol;
        this.value = value;
    }

    @Override
    public String toString() {
        return value + " of " + symbol;
    }
}

public class CardManager {
    private Collection<Card> cards = new ArrayList<>();
    private Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        new CardManager().run();
    }

    public void run() {
        while (true) {
            System.out.println("\n1. Add Card\n2. Find Cards by Symbol\n3. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();  // consume newline
```

```
            switch (choice) {
                case 1 -> addCard();
                case 2 -> findCardsBySymbol();
                case 3 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice, please try again.");
            }
        }
    }

    private void addCard() {
        System.out.print("Enter Symbol (e.g., Hearts, Spades): ");
        String symbol = scanner.nextLine();
        System.out.print("Enter Value (e.g., Ace, 2, 3, ...): ");
        String value = scanner.nextLine();
        cards.add(new Card(symbol, value));
        System.out.println("Card added successfully.");
    }

    private void findCardsBySymbol() {
        System.out.print("Enter Symbol to search (e.g., Hearts, Spades): ");
        String symbol = scanner.nextLine();
        System.out.println("Cards with symbol " + symbol + ":");
        for (Card card : cards) {
            if (card.symbol.equalsIgnoreCase(symbol)) {
                System.out.println(card);
            }
        }
    }
}
```

4. Output:

```
input

1. Add Card
2. Find Cards by Symbol
3. Exit
Enter your choice: 1
Enter Symbol (e.g., Hearts, Spades): Hearts
Enter Value (e.g., Ace, 2, 3, ...): Ace
Card added successfully.

1. Add Card
2. Find Cards by Symbol
3. Exit
Enter your choice: 1
Enter Symbol (e.g., Hearts, Spades): Spades
Enter Value (e.g., Ace, 2, 3, ...): 2
Card added successfully.

1. Add Card
2. Find Cards by Symbol
3. Exit
Enter your choice: 2
Enter Symbol to search (e.g., Hearts, Spades): Hearts
Cards with symbol Hearts:
Ace of Hearts

1. Add Card
2. Find Cards by Symbol
3. Exit
Enter your choice: 3
Exiting...


...Program finished with exit code 0
Press ENTER to exit console.
```

Code 3:

1. **Aim:** Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

2. **Objective:** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

3. **Code:**

```
class Seat {
boolean booked = false; synchronized boolean book() {
if (booked) return false;

booked = true; return true;
}
}

class BookingSystem { Seat[] seats;
BookingSystem(int n) { seats = new Seat[n]; for (int i = 0; i < n; i++) seats[i] = new
Seat(); } synchronized boolean bookSeat(int n) { return seats[n - 1].book(); }
}

class BookingThread extends Thread { BookingSystem system; int seat;
BookingThread(BookingSystem s, int seat, String name, int priority) { super(name);
this.system = s; this.seat = seat; setPriority(priority);
}
public void run() {
System.out.println(getName() + (system.bookSeat(seat) ? " booked seat " : " failed
to book seat ") + seat);
}
}

public class TicketBookingSystem { public static void main(String[] args) {
BookingSystem system = new BookingSystem(10);
new BookingThread(system, 5, "VIP-1", Thread.MAX_PRIORITY).start(); new
BookingThread(system, 5, "VIP-2", Thread.MAX_PRIORITY).start();
new BookingThread(system, 5, "Regular-1", Thread.NORM_PRIORITY).start();
new BookingThread(system, 6, "Regular-2", Thread.NORM_PRIORITY).start();
new BookingThread(system, 7, "Regular-3", Thread.NORM_PRIORITY).start();
}
}
```

4. **Output:**

```
input
Regular-2 booked seat 6
Regular-3 booked seat 7
VIP-1 booked seat 5
Regular-1 failed to book seat 5
VIP-2 failed to book seat 5


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

1. Demonstrate: Apply key concepts to real-world scenarios to showcase understanding.
2. Analyze: Critically evaluate information, identify patterns, and draw meaningful conclusions.
3. Create: Develop original work, including presentations, reports, or projects, to exhibit comprehension and skills.