Experiment 4

Student Name: Madhav Uppal **UID** 22BCS50195

Branch: CSE **Section/Group:** 641/B

Semester: 6th **Date of Performance:** 5/03/25

Subject Code: 22CSH-359

Subject Name: PBLJ

1. **Aim**: 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.

2. **Objective :** Java program that uses an ArrayList to store employee details (ID, Name, and Salary). The program allows users to add, update, remove, and search for employees.

3. **Algorithms:**
   1. Start
   2. Create an Employee class with attributes:
   3. ID (int), Name (String), Salary (double).
   4. Use an ArrayList to store multiple employees.
   5. Display a menu with options:
   6. Add an Employee
   7. Update Employee Details
   8. Remove an Employee
   9. Search for an Employee
   10. Display All Employees
   11. Exit

4. **Code :**

Main.java

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

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

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
} public class Main {
    private static ArrayList<Employee> employees = new ArrayList<>();

    public static void addEmployee(int id, String name, double salary) {
        Employee employee = new Employee(id, name, salary);
        employees.add(employee);
        System.out.println("Employee added successfully.");
    }

    public static void updateEmployee(int id, String name, double salary) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                emp.name = name;
                emp.salary = salary;
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    public static void removeEmployee(int id) {
        for (Employee emp : employees) {
```

```
        }
        System.out.println("Employee not found.");
    }

    public static void removeEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                employees.remove(emp);
                System.out.println("Employee removed successfully.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    public static void searchEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found.");
    }
    public static void displayAllEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees to display.");
            return;
        }
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }

    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 Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
            System.out.println("4. Search Employee");
            System.out.println("5. Display All Employees");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline
```

```
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline

            switch (choice) {
            case 1:
                System.out.print("Enter Employee ID: ");
                int id = scanner.nextInt();
                scanner.nextLine(); // Consume the newline
                String name = scanner.nextLine();
                System.out.print("Enter Employee Salary: ");
                double salary = scanner.nextDouble();
                addEmployee(id, name, salary);
                break;
            case 2:
                System.out.print("Enter Employee ID to update: ");
                id = scanner.nextInt();
                scanner.nextLine();
                System.out.print("Enter new Name: ");
                name = scanner.nextLine();
                System.out.print("Enter new Salary: ");
                salary = scanner.nextDouble();
                updateEmployee(id, name, salary);
                break;
            case 3:
                System.out.print("Enter Employee ID to remove: ");
                id = scanner.nextInt();
                removeEmployee(id);
                break;
            case 4:
                System.out.print("Enter Employee ID to search: ");
                id = scanner.nextInt();
                searchEmployee(id);
                break;
            case 5:
                displayAllEmployees();
                break;
            case 6:
                System.out.println("Exiting system...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

5. Output

```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 1001
Enter Employee Name: Vinod
Enter Employee Salary: 50000
Employee added successfully.
```

B}

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: To develop a Java program using the Collection interface to store and manage playing cards. The program will help users:

Store cards in a collection.
Search for cards by a given symbol (e.g., Hearts, Spades).
Display all available cards in the collection.

3. Algorithm:

- Start
- Create a Card class with attributes:
- Symbol (String), Number (String).
- Use a Collection (ArrayList) to store multiple card objects.
- Display a menu with options:
- Add a card.
- Find all cards by symbol.
- Display all stored cards.
- Exit the program

4. Code

Main.java

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

class Card {
    private String rank;
    private String symbol;

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

    public String getRank() {
        return rank;
    }

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

public class Main {
    private List<Card> cards;
    public Main() {
        cards = new ArrayList<>();
        createDeck();  // Populate the collection with a deck of cards
    }
    private void createDeck() {
        String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
        String[] symbols = {"Hearts", "Diamonds", "Clubs", "Spades"};

        for (String symbol : symbols) {
            for (String rank : ranks) {
                cards.add(new Card(rank, symbol));
            }
        }
    }
    public void findCardsBySymbol(String symbol) {
        boolean found = false;
        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                System.out.println(card);  // Print the card
                found = true;
```

```java
        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                System.out.println(card);  // Print the card
                found = true;
            }
        }
        if (!found) {
            System.out.println("No cards found with the symbol: " + symbol);
        }
    }
    public void displayAllCards() {
        for (Card card : cards) {
            System.out.println(card);
        }
    }

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

        while (true) {
            System.out.println("\nCard Collection Menu");
            System.out.println("1. Display all cards");
            System.out.println("2. Find cards by symbol");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
            case 1:
                cardCollection.displayAllCards();
                break;
            case 2:
                System.out.print("Enter the symbol to search for (Hearts, Diamonds, Clubs, Spades): ");
                String symbol = scanner.nextLine();
                cardCollection.findCardsBySymbol(symbol);
                break;
            case 3:
                System.out.println("Exiting the program...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
            }
        }
    }
```

4. Output

```
Card Collection Menu
1. Display all cards
2. Find cards by symbol
3. Exit
Enter your choice: 2
Enter the symbol to search for (Hearts, Diamonds, Clubs, Spades): Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
```

C}

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. **Code :**

Main.java

```
class TicketBookingSystem {
    private boolean[] seats;  // Array to store seat availability

    // Constants for seat availability
    private static final boolean BOOKED = true;
    private static final boolean AVAILABLE = false;

    public TicketBookingSystem(int totalSeats) {
        seats = new boolean[totalSeats];  // Initialize all seats to available (false)
    }

    // Synchronized method to book a seat
    public synchronized boolean bookSeat(int seatNumber, String customerType) {
        if (seatNumber < 0 || seatNumber >= seats.length) {
            System.out.println("Invalid seat number.");
            return false;
        }

        if (seats[seatNumber] == BOOKED) {
            System.out.println(customerType + " failed to book Seat " + seatNumber + ". Seat already booked.");
            return false;
        } else {
            // Book the seat
            seats[seatNumber] = BOOKED;
            System.out.println(customerType + " successfully booked Seat " + seatNumber);
            return true;
        }
    }

    public void displayAvailableSeats() {
        System.out.println("Available Seats:");
        for (int i = 0; i < seats.length; i++) {
            if (seats[i] == AVAILABLE) {
                System.out.println("Seat " + i + " is available.");
            }
        }
    }
}

class BookingThread extends Thread {
    private TicketBookingSystem system;
    private int seatNumber;
    private String customerType;

    public BookingThread(TicketBookingSystem system, int seatNumber, String customerType) {
        this.system = system;
        this.seatNumber = seatNumber;
```

Main.java

```
        this.system = system;
        this.seatNumber = seatNumber;
        this.customerType = customerType;
    }

    @Override
    public void run() {
        // Try to book the seat, if unsuccessful, retry
        while (!system.bookSeat(seatNumber, customerType)) {
            try {
                Thread.sleep(500);  // Simulate some waiting time before retrying
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a ticket booking system with 10 seats
        TicketBookingSystem system = new TicketBookingSystem(10);

        // Display available seats
        system.displayAvailableSeats();

        // Create normal and VIP customers (Threads)
        BookingThread normalCustomer1 = new BookingThread(system, 5, "Normal Customer 1");
        BookingThread normalCustomer2 = new BookingThread(system, 5, "Normal Customer 2");
        BookingThread vipCustomer1 = new BookingThread(system, 2, "VIP Customer 1");
        BookingThread normalCustomer3 = new BookingThread(system, 3, "Normal Customer 3");
        BookingThread vipCustomer2 = new BookingThread(system, 4, "VIP Customer 2");

        // Set priority: VIP customers should have a higher chance of booking
        vipCustomer1.setPriority(Thread.MAX_PRIORITY);
        vipCustomer2.setPriority(Thread.MAX_PRIORITY);
        normalCustomer1.setPriority(Thread.NORM_PRIORITY);
        normalCustomer2.setPriority(Thread.NORM_PRIORITY);
        normalCustomer3.setPriority(Thread.NORM_PRIORITY);

        // Start the threads (customers trying to book seats)
        vipCustomer1.start();
        vipCustomer2.start();
        normalCustomer1.start();
        normalCustomer2.start();
        normalCustomer3.start();
```

```
        // Start the threads (customers trying to book seats)
        vipCustomer1.start();
        vipCustomer2.start();
        normalCustomer1.start();
        normalCustomer2.start();
        normalCustomer3.start();

        try {
            // Wait for all threads to finish
            vipCustomer1.join();
            vipCustomer2.join();
            normalCustomer1.join();
            normalCustomer2.join();
            normalCustomer3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Display available seats after booking
        system.displayAvailableSeats();
    }
}
```

3. Screenshot of Outputs:

```
Available Seats:
Seat 0 is available.
Seat 1 is available.
Seat 2 is available.
Seat 3 is available.
Seat 4 is available.
Seat 5 is available.
Seat 6 is available.
Seat 7 is available.
Seat 8 is available.
Seat 9 is available.
VIP Customer 1 successfully booked Seat 2
Normal Customer 3 successfully booked Seat 3
Normal Customer 2 successfully booked Seat 5
Normal Customer 1 failed to book Seat 5. Seat already booked.
VIP Customer 2 successfully booked Seat 4
Normal Customer 1 failed to book Seat 5. Seat already booked.
Normal Customer 1 failed to book Seat 5. Seat already booked.
```

4. Learning Outcomes:

a. Understanding of Object-Oriented Programming (OOP)
b. Real-world Application of Software Design Principles.
c. Threading in java.