Experiment 4

Name: Deepanjali **UID: 22BCS14571**
Branch: BE-CSE **Section/Group: 22BC_IOT-639/A**
Semester: 6th **Date of Performance:14/02/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

EASY:

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.

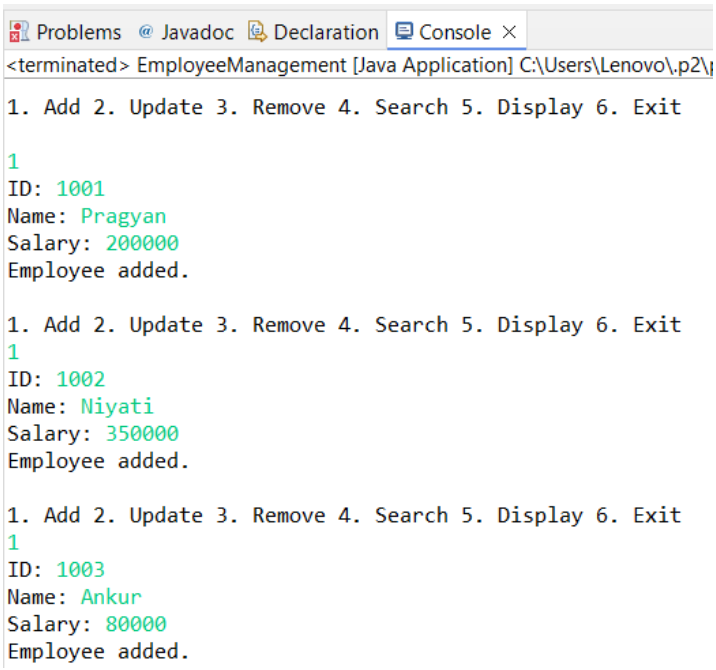
2. **Implementation/Code:**

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

class Employee {
    private int id;
    private String name;
    private double salary;
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    public int getId() { return id; }
    public String getName() { return name; }
    public double getSalary() { return salary; }
    public void setName(String name) { this.name = name; }
    public void setSalary(double salary) { this.salary = salary; }

    @Override
    public String toString() {
        return "Employee [ID=" + id + ", Name=" + name + ", Salary=" + salary + "]";
    }
}

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

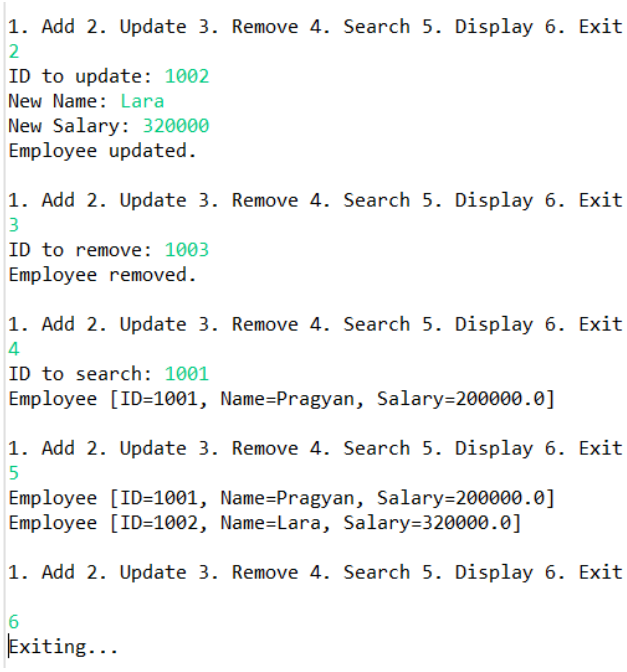
```
public static void main(String[] args) {
    while (true) {
        System.out.println("\n1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit");
        switch (scanner.nextInt()) {
            case 1 -> addEmployee();
            case 2 -> updateEmployee();
            case 3 -> removeEmployee();
            case 4 -> searchEmployee();
            case 5 -> displayAllEmployees();
            case 6 -> { System.out.println("Exiting..."); return; }
            default -> System.out.println("Invalid choice.");
        }
    }
}
private static void addEmployee() {
    System.out.print("ID: "); int id = scanner.nextInt();
    System.out.print("Name: "); String name = scanner.next();
    System.out.print("Salary: "); double salary = scanner.nextDouble();
    employees.add(new Employee(id, name, salary));
    System.out.println("Employee added.");
}
private static void updateEmployee() {
    System.out.print("ID to update: "); int id = scanner.nextInt();
    for (Employee e : employees) {
        if (e.getId() == id) {
            System.out.print("New Name: "); e.setName(scanner.next());
            System.out.print("New Salary: "); e.setSalary(scanner.nextDouble());
            System.out.println("Employee updated."); return;
        }
    }
    System.out.println("Employee not found.");
}
private static void removeEmployee() {
    System.out.print("ID to remove: "); int id = scanner.nextInt();
    employees.removeIf(e -> e.getId() == id);
    System.out.println("Employee removed.");
}
private static void searchEmployee() {
    System.out.print("ID to search: "); int id = scanner.nextInt();
    employees.stream().filter(e -> e.getId() == id).forEach(System.out::println);
}
private static void displayAllEmployees() {
    if (employees.isEmpty()) System.out.println("No employees found.");
```

```
        else employees.forEach(System.out::println);
    }
}
```

3. **Output**



```
Problems  Javadoc  Declaration  Console ×
<terminated> EmployeeManagement [Java Application] C:\Users\Lenovo\.p2\
1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit

1
ID: 1001
Name: Pragyan
Salary: 200000
Employee added.

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
1
ID: 1002
Name: Niyati
Salary: 350000
Employee added.

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
1
ID: 1003
Name: Ankur
Salary: 80000
Employee added.
```



```
1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
2
ID to update: 1002
New Name: Lara
New Salary: 320000
Employee updated.

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
3
ID to remove: 1003
Employee removed.

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
4
ID to search: 1001
Employee [ID=1001, Name=Pragyan, Salary=200000.0]

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
5
Employee [ID=1001, Name=Pragyan, Salary=200000.0]
Employee [ID=1002, Name=Lara, Salary=320000.0]

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit

6
Exiting...
```

MEDIUM:

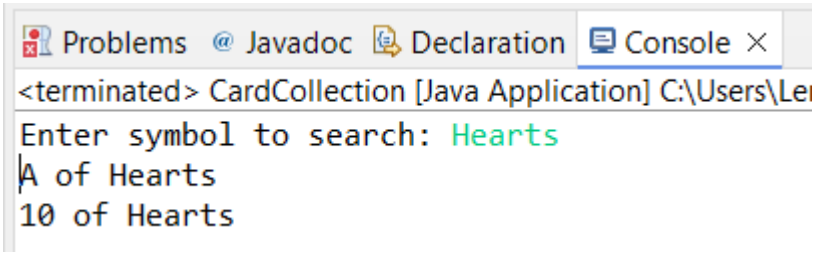
1. **Aim:** Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

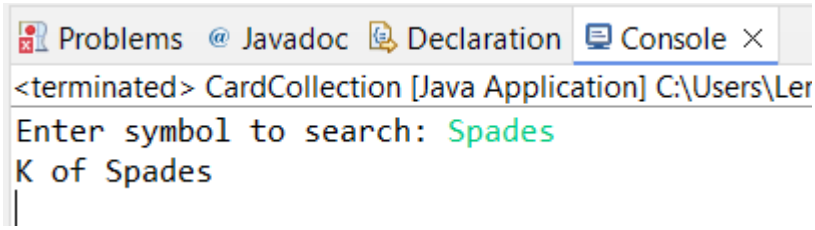
2. **Implementation/Code:**

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

class Card {
    String symbol, value;
    Card(String symbol, String value) { this.symbol = symbol; this.value = value; }
    public String toString() { return value + " of " + symbol; }
}
public class CardCollection {
    static Collection<Card> cards = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        addCard("Hearts", "A"); addCard("Spades", "K"); addCard("Hearts", "10");
```

```
        System.out.print("Enter symbol to search: ");
        String symbol = scanner.next();
        cards.stream().filter(c ->c.symbol.equalsIgnoreCase(symbol)).forEach(System.out::println);
    }
    static void addCard(String symbol, String value) { cards.add(new Card(symbol, value)); }
}
```

3. **Output:**





HARD:

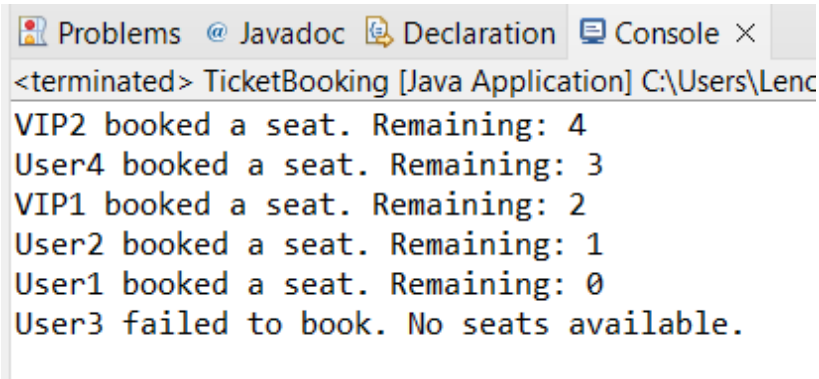
1. **Aim:** Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

2. **Implementation/Code:**

```
package Java;
class TicketBookingSystem {
    private int availableSeats = 5;
    public synchronized void bookTicket(String name) {
        if (availableSeats > 0) {
            System.out.println(name + " booked a seat. Remaining: " + (--availableSeats));
        } else {
            System.out.println(name + " failed to book. No seats available.");
        }
    }
}
class Passenger extends Thread {
    private TicketBookingSystem system;
    private String name;
```

```
    public Passenger(TicketBookingSystem system, String name, int priority) {
        this.system = system;
        this.name = name;
        setPriority(priority);
    }
    public void run() {
        system.bookTicket(name);
    }
}
public class TicketBooking {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        Passenger p1 = new Passenger(system, "VIP1", Thread.MAX_PRIORITY);
        Passenger p2 = new Passenger(system, "VIP2", Thread.MAX_PRIORITY);
        Passenger p3 = new Passenger(system, "User1", Thread.NORM_PRIORITY);
        Passenger p4 = new Passenger(system, "User2", Thread.NORM_PRIORITY);
        Passenger p5 = new Passenger(system, "User3", Thread.NORM_PRIORITY);
        Passenger p6 = new Passenger(system, "User4", Thread.MIN_PRIORITY);

        p1.start();
        p2.start();
        p3.start();
        p4.start();
        p5.start();
        p6.start();
    }
}
```

3. **Output:**



4. Learning Outcome

a) Learned how to use ArrayList and Collection interfaces to manage employee records and card collections.
b) Implemented synchronized methods to prevent race conditions in a multi-threaded environment.
c) Explored how thread priority affects execution order, ensuring VIP bookings are processed first.
d) Improved skills in handling user input and managing errors in real-world applications.
e) Utilized Java Streams and lambda expressions for efficient searching and filtering.