Experiment 4

Student Name: Harsh Raj Choudhary **UID: 22BCS11231**

Branch: UIE CSE 3rd Year **Section/Group: 22BCS_KPIT-901 A**

Semester: 6th **Date of Performance: 21st February 2025**

Subject Name: Project Based Learning with JAVA **Subject Code: 22CSH-359**

Java ArrayList

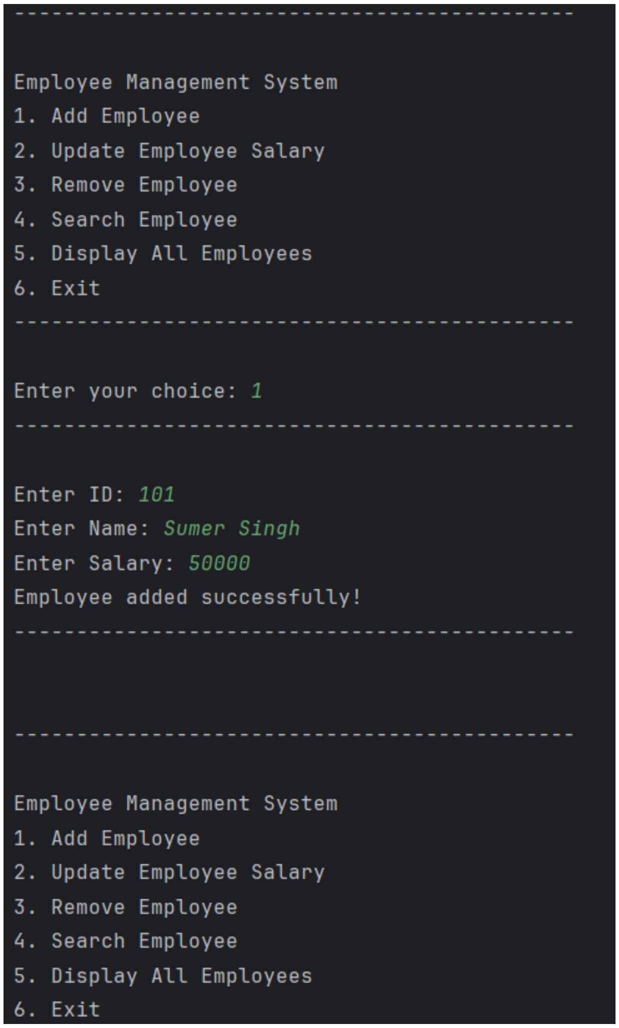
1. **Aim:** Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2. **Objective:** The objective of this Java program is to implement an **ArrayList-based Employee Management System** that allows users to **add, update, remove, and search** employee records. Each employee has an **ID, Name, and Salary** stored as an object. The program provides a **menu-driven interface** for easy interaction, where users can perform operations like adding a new employee, updating salary details, removing an employee, and searching for an employee by ID. This implementation demonstrates **ArrayList operations**, **object manipulation**, and **basic CRUD functionalities** while ensuring efficient storage and retrieval of employee data in a dynamic list structure.

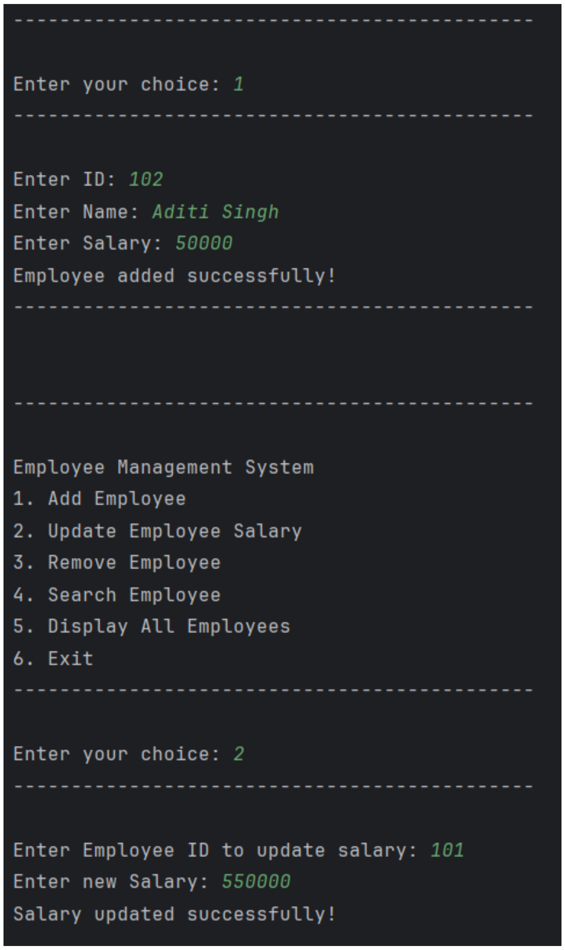
3. **Implementation/Code:**

```
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;
    }

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

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

    public static void main(String[] args) {
        while (true) {
            System.out.println("\nEmployee Management System");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee Salary");
            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();
            switch (choice) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> displayEmployees();
                case 6 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice. Try again.");
            }
        }
    }

    static 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!");
    }

    static void updateEmployee() {
        System.out.print("Enter Employee ID to update salary: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.print("Enter new Salary: ");
                emp.salary = scanner.nextDouble();
                System.out.println("Salary updated successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        employees.removeIf(emp -> emp.id == id);
        System.out.println("Employee removed successfully!");
    }

    static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }
}
```

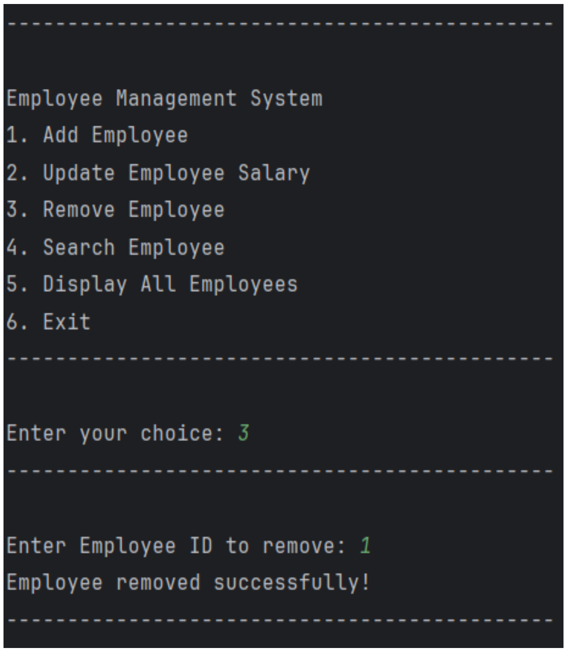
4. Output:



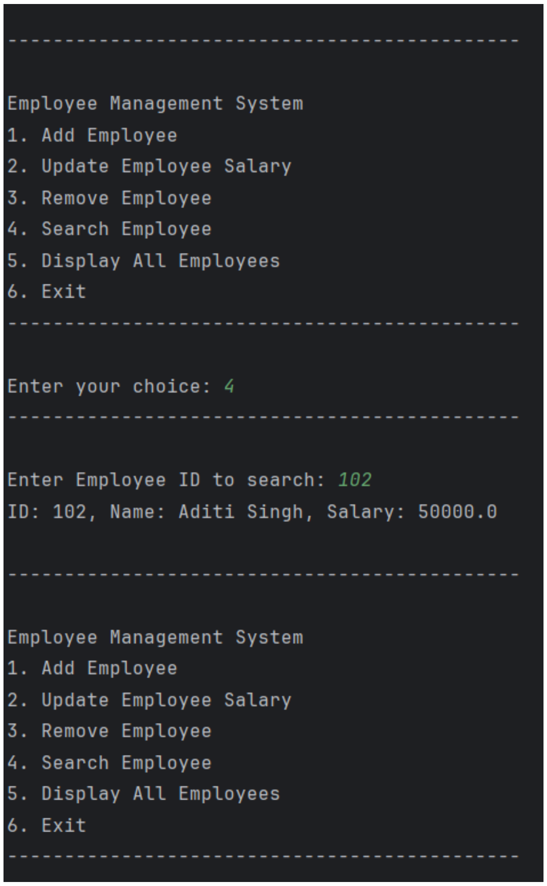
```
--------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
--------------------------------------------

Enter your choice: 1
--------------------------------------------

Enter ID: 101
Enter Name: Sumer Singh
Enter Salary: 50000
Employee added successfully!
--------------------------------------------



--------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
```



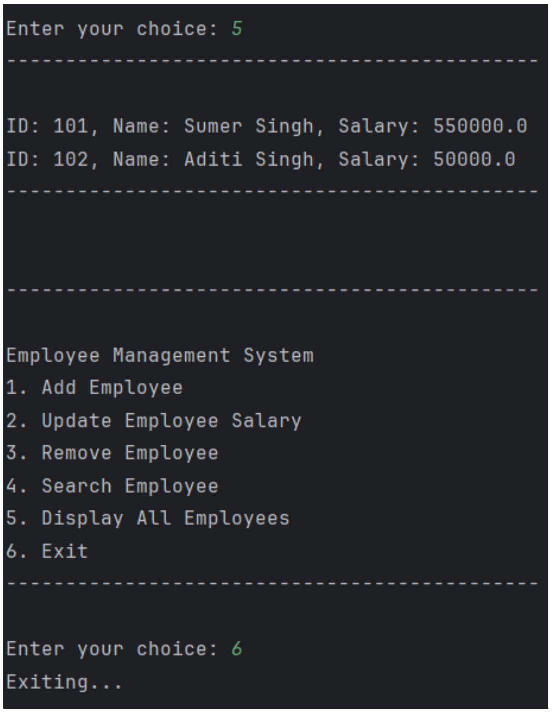
```
---------------------------------------------

Enter your choice: 1
---------------------------------------------

Enter ID: 102
Enter Name: Aditi Singh
Enter Salary: 50000
Employee added successfully!
---------------------------------------------



---------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
---------------------------------------------

Enter your choice: 2
---------------------------------------------

Enter Employee ID to update salary: 101
Enter new Salary: 550000
Salary updated successfully!
```



```
---------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
---------------------------------------------

Enter your choice: 3
---------------------------------------------

Enter Employee ID to remove: 1
Employee removed successfully!
---------------------------------------------
```



```
---------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
---------------------------------------------

Enter your choice: 4
---------------------------------------------

Enter Employee ID to search: 102
ID: 102, Name: Aditi Singh, Salary: 50000.0

---------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
---------------------------------------------
```



```
Enter your choice: 5
---------------------------------------------

ID: 101, Name: Sumer Singh, Salary: 550000.0
ID: 102, Name: Aditi Singh, Salary: 50000.0
---------------------------------------------


---------------------------------------------

Employee Management System
1. Add Employee
2. Update Employee Salary
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
---------------------------------------------

Enter your choice: 6
Exiting...
```

Java Interface

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.

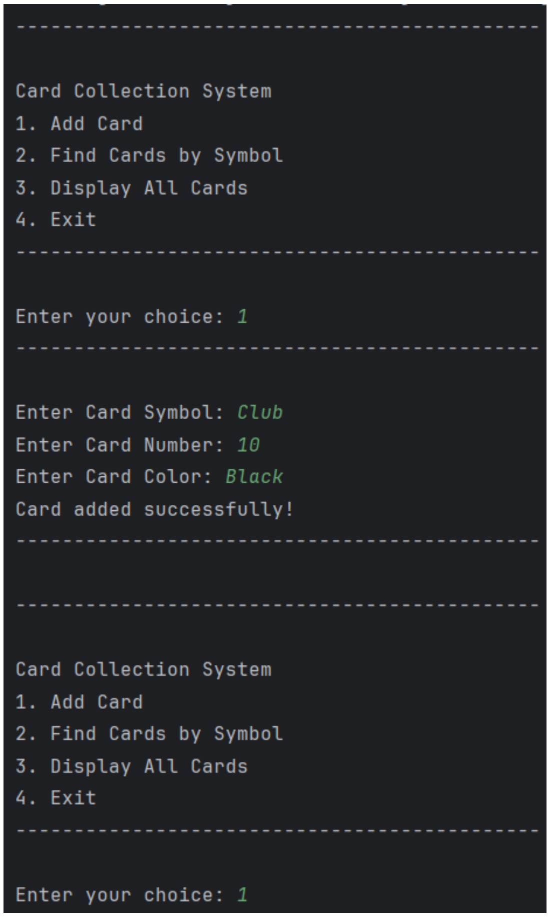
2. **Objective:** The objective of this program is to implement a Card Collection System using Java's Collection interface. The program allows users to store, retrieve, and search for cards based on their symbol. Each card has a symbol, number, and color and is stored dynamically in a collection. Users can add new cards, retrieve all cards of a particular symbol, and display the complete card collection. This program demonstrates Collection framework usage, efficient data storage, and retrieval operations while ensuring a flexible and scalable approach to card management.

3. **Implementation/Code:**

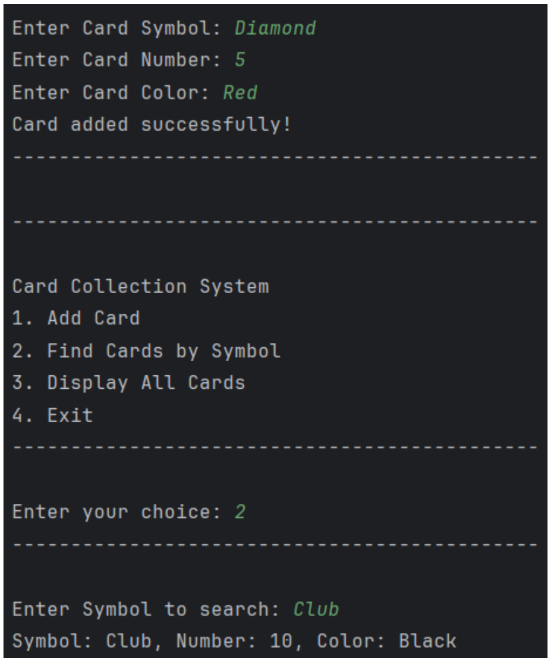
```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Card {
    private String symbol;
    private int number;
    private String color;

    public Card(String symbol, int number, String color) {
        this.symbol = symbol;
        this.number = number;
        this.color = color;
    }

    public String getSymbol() {
        return symbol;
    }

    @Override
    public String toString() {
        return "Symbol: " + symbol + ", Number: " + number + ", Color: " + color;
    }
}

public class CardCollection {
    private static List<Card> cards = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("---------------------------------------------\n");
            System.out.println("Card Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Find Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.println("---------------------------------------------\n");
            System.out.print("Enter your choice: ");

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

            switch (choice) {
                case 1 -> addCard();
            case 2 -> findCardsBySymbol();
            case 3 -> displayAllCards();
            case 4 -> {
                System.out.println("Exiting...");
                return;
            }
            default -> System.out.println("Invalid choice. Try again.");
        }
    }
}

private static void addCard() {
    System.out.println("--------------------------------------------\n");
    System.out.print("Enter Card Symbol: ");
    String symbol = scanner.nextLine();
    System.out.print("Enter Card Number: ");
    int number = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    System.out.print("Enter Card Color: ");
    String color = scanner.nextLine();

    cards.add(new Card(symbol, number, color));
    System.out.println("Card added successfully!");
    System.out.println("--------------------------------------------\n");
}

private static void findCardsBySymbol() {
    System.out.println("--------------------------------------------\n");
    System.out.print("Enter Symbol to search: ");
```

```
        String symbol = scanner.nextLine();
        boolean found = false;

        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                System.out.println(card);
                found = true;
            }
        }

        if (!found) {
            System.out.println("No cards found with the given symbol.");
        }
        System.out.println("--------------------------------------------\n");
    }

    private static void displayAllCards() {
        System.out.println("--------------------------------------------\n");
        if (cards.isEmpty()) {
            System.out.println("No cards available.");
        } else {
            for (Card card : cards) {
                System.out.println(card);
            }
        }
        System.out.println("--------------------------------------------\n");
    }
}
```

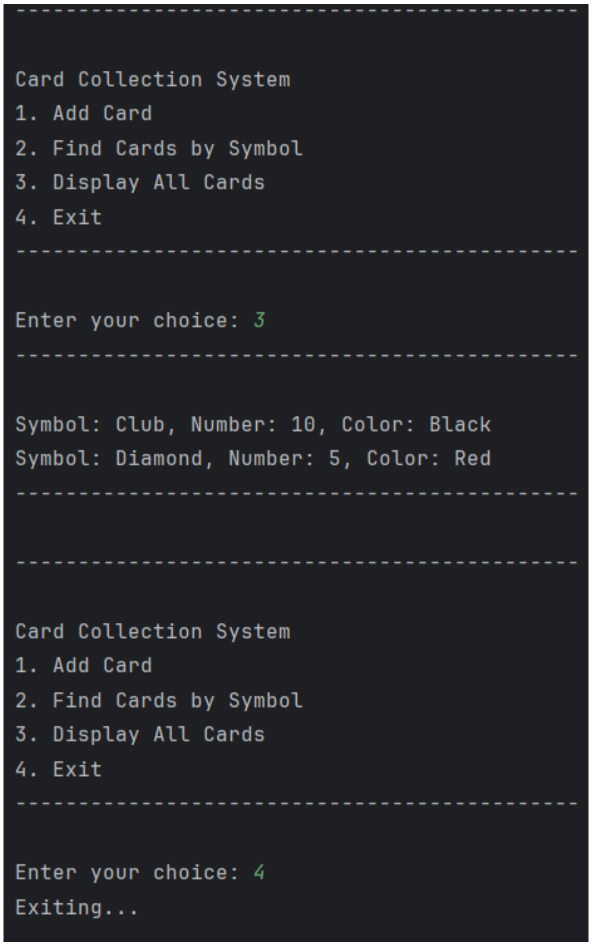
4. Output:



```
---------------------------------------------

Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
---------------------------------------------

Enter your choice: 1
---------------------------------------------

Enter Card Symbol: Club
Enter Card Number: 10
Enter Card Color: Black
Card added successfully!
---------------------------------------------


---------------------------------------------

Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
---------------------------------------------

Enter your choice: 1
```



```
Enter Card Symbol: Diamond
Enter Card Number: 5
Enter Card Color: Red
Card added successfully!
---------------------------------------------


---------------------------------------------

Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
---------------------------------------------

Enter your choice: 2
---------------------------------------------

Enter Symbol to search: Club
Symbol: Club, Number: 10, Color: Black
```



```
---------------------------------------------

Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
---------------------------------------------

Enter your choice: 3
---------------------------------------------

Symbol: Club, Number: 10, Color: Black
Symbol: Diamond, Number: 5, Color: Red
---------------------------------------------


---------------------------------------------

Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
---------------------------------------------

Enter your choice: 4
Exiting...
```

Java Interface

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.

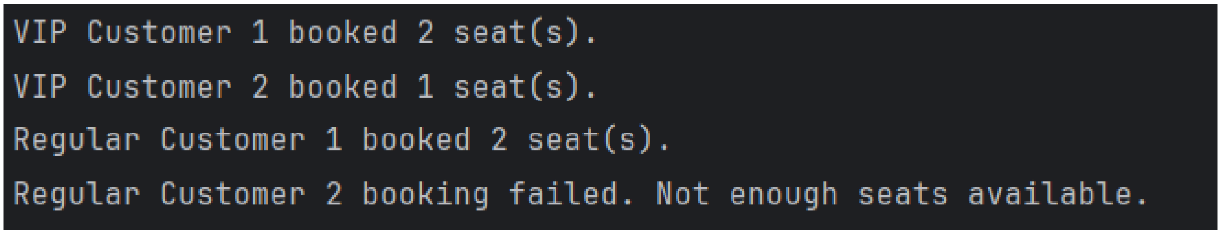
2. **Objective:** The objective of this program is to develop a **multi-threaded ticket booking system** that ensures **synchronized seat allocation** to prevent double booking. The program will use **thread synchronization** to handle concurrent booking requests safely. Additionally, **thread priorities** will be utilized to simulate a real-world scenario where **VIP bookings** are processed first, ensuring a fair and efficient seat allocation process. This implementation will demonstrate **Java's multithreading,**

synchronization mechanisms (synchronized methods/blocks), and thread priorities, showcasing how concurrent systems can be managed securely without data inconsistencies.

3. Implementation/Code:

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class TicketBookingSystem {
    private int availableSeats;
    private final Lock lock = new ReentrantLock(); // Lock to ensure thread safety

    public TicketBookingSystem(int seats) {
        this.availableSeats = seats;
    }

    public void bookSeat(String customerType, int requestedSeats) {
        lock.lock();
        try {
            if (requestedSeats <= availableSeats) {
                System.out.println(customerType + " booked " + requestedSeats + " seat(s).");
                availableSeats -= requestedSeats;
            } else {
                System.out.println(customerType + " booking failed. Not enough seats available.");
            }
        } finally {
            lock.unlock();
        }
    }

    public int getAvailableSeats() {
        return availableSeats;
    }
}

class Customer extends Thread {
    private final TicketBookingSystem system;
    private final String customerType;
    private final int requestedSeats;

    public Customer(TicketBookingSystem system, String customerType, int requestedSeats, int priority)
{
        this.system = system;
        this.customerType = customerType;
        this.requestedSeats = requestedSeats;
        setPriority(priority); // Set thread priority
    }

    @Override
    public void run() {
        system.bookSeat(customerType, requestedSeats);
    }
}

public class TicketBookingApp {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5); // Total available seats
        // Creating customer threads (VIPs have higher priority)
        Customer vip1 = new Customer(system, "VIP Customer 1", 2, Thread.MAX_PRIORITY);
        Customer vip2 = new Customer(system, "VIP Customer 2", 1, Thread.MAX_PRIORITY);
        Customer regular1 = new Customer(system, "Regular Customer 1", 2, Thread.NORM_PRIORITY);
        Customer regular2 = new Customer(system, "Regular Customer 2", 1, Thread.MIN_PRIORITY);
        vip1.start();
        vip2.start();
        regular1.start();
        regular2.start();

    }
}
```

4. Output:



```
VIP Customer 1 booked 2 seat(s).
VIP Customer 2 booked 1 seat(s).
Regular Customer 1 booked 2 seat(s).
Regular Customer 2 booking failed. Not enough seats available.
```

5. Learning Outcome:

- **Understanding Thread Synchronization:** Learned how to use ReentrantLock to prevent race conditions and ensure thread-safe seat booking.
- **Implementing Thread Priorities:** Gained insight into thread priority management to simulate VIP bookings being processed first.
- **Concurrency Handling in Java:** Explored multi-threading concepts to handle multiple booking requests simultaneously.
- **Resource Management:** Learned how to efficiently allocate and manage limited resources (seats) in a concurrent environment.
- **Real-world Application Simulation:** Developed a practical ticket booking system that mimics real-world priority-based seat allocation.