Name: Agrima Sharma

UID:22BCS15314

Section:IOT-620/A

Easy:

Solution:

```
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 "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

public class EmployeeManagementSystem {
    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("\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();

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                updateEmployee();
                break;
            case 3:
                removeEmployee();
                break;
            case 4:
```

```
            searchEmployee();
            break;
        case 5:
            displayEmployees();
            break;
        case 6:
            System.out.println("Exiting...");
            scanner.close();
            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();
   System.out.print("Enter Employee Name: ");
   String name = scanner.nextLine();
   System.out.print("Enter Employee Salary: ");
   double salary = scanner.nextDouble();

   employees.add(new Employee(id, name, salary));
```

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

private static void updateEmployee() {
    System.out.print("Enter Employee ID to update: ");
    int id = scanner.nextInt();
    scanner.nextLine();

    for (Employee emp : employees) {
        if (emp.getId() == id) {
            System.out.print("Enter new name: ");
            String newName = scanner.nextLine();
            System.out.print("Enter new salary: ");
            double newSalary = scanner.nextDouble();
            emp.setName(newName);
            emp.setSalary(newSalary);
            System.out.println("Employee details updated successfully!");
            return;
        }
    }
    System.out.println("Employee not found!");
}

private static void removeEmployee() {
    System.out.print("Enter Employee ID to remove: ");
```

```
    int id = scanner.nextInt();

    for (Employee emp : employees) {
        if (emp.getId() == id) {
            employees.remove(emp);
            System.out.println("Employee removed successfully!");
            return;
        }
    }
    System.out.println("Employee not found!");
}

private static void searchEmployee() {
    System.out.print("Enter Employee ID to search: ");
    int id = scanner.nextInt();

    for (Employee emp : employees) {
        if (emp.getId() == id) {
            System.out.println(emp);
            return;
        }
    }
    System.out.println("Employee not found!");
}
```

```
    private static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found!");
            return;
        }
        System.out.println("\nEmployee List:");
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}
```

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: 101
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 5

Employee List:
ID: 101, Name: Alice, Salary: 50000.0
```

Medium:

Solution:

```
import java.util.*;


class CardCollection {
    private HashMap<String, List<String>> cardMap; // Symbol -> List of Card Names
    private Scanner scanner;


    public CardCollection() {
        cardMap = new HashMap<>();
        scanner = new Scanner(System.in);
    }


    // Add a card to the collection
    public void addCard() {
        System.out.print("Enter card symbol (e.g., Hearts, Spades): ");
        String symbol = scanner.nextLine().trim();
        System.out.print("Enter card name (e.g., Ace, King, Queen): ");
        String name = scanner.nextLine().trim();
```

```
    cardMap.putIfAbsent(symbol, new ArrayList<>()); // Initialize list if not present
    cardMap.get(symbol).add(name);

    System.out.println("Card added successfully!");
}

// Search for cards by symbol
public void searchBySymbol() {
    System.out.print("Enter symbol to search: ");
    String symbol = scanner.nextLine().trim();

    if (cardMap.containsKey(symbol)) {
        System.out.println("Cards with symbol '" + symbol + "': " + cardMap.get(symbol));
    } else {
        System.out.println("No cards found for this symbol.");
    }
}

// Display all stored cards
public void displayAllCards() {
    if (cardMap.isEmpty()) {
        System.out.println("No cards in the collection.");
        return;
    }
    System.out.println("\nCard Collection:");
```

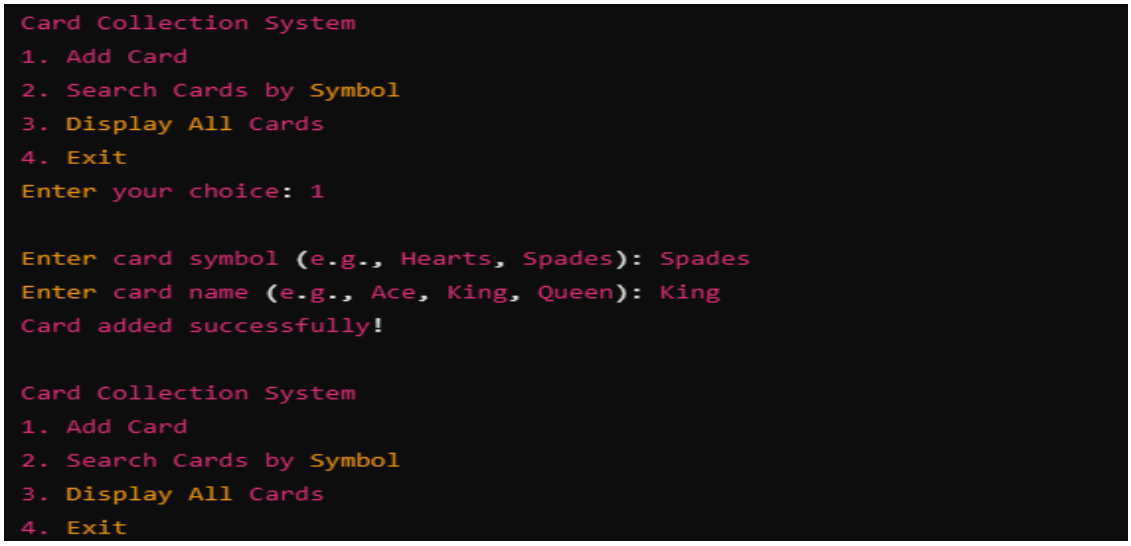
```
    for (Map.Entry<String, List<String>> entry : cardMap.entrySet()) {
        System.out.println("Symbol: " + entry.getKey() + " -> Cards: " + entry.getValue());
    }
}

public void start() {
    while (true) {
        System.out.println("\nCard Collection System");
        System.out.println("1. Add Card");
        System.out.println("2. Search Cards by Symbol");
        System.out.println("3. Display All Cards");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine();  // Consume newline

        switch (choice) {
            case 1:
                addCard();
                break;
            case 2:
                searchBySymbol();
                break;
            case 3:
                displayAllCards();
```

```
                break;
            case 4:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    }
  }
}

public class CardCollectionSystem {
    public static void main(String[] args) {
        CardCollection collection = new CardCollection();
        collection.start();
    }
}
```

Output:

```
Card Collection System
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1

Enter card symbol (e.g., Hearts, Spades): Hearts
Enter card name (e.g., Ace, King, Queen): Ace
Card added successfully!
```

Hard:



Solution:

```
import java.util.*;


class TicketBookingSystem {
    private int availableSeats;
    private final Object lock = new Object(); // Lock for synchronization

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

    public void bookTicket(String name, int seatsRequested) {
        synchronized (lock) {
            if (seatsRequested > availableSeats) {
                System.out.println(name + " requested " + seatsRequested + " seats. Not enough seats available!");
            } else {
```

```
            System.out.println(name + " successfully booked " + seatsRequested + " seat(s).");
            availableSeats -= seatsRequested;
        }
        System.out.println("Seats remaining: " + availableSeats);
    }
  }
}

// Runnable class for booking tickets
class BookingThread extends Thread {
  private TicketBookingSystem bookingSystem;
  private String customerName;
  private int seatsRequested;

  public BookingThread(TicketBookingSystem system, String name, int seats) {
    this.bookingSystem = system;
    this.customerName = name;
    this.seatsRequested = seats;
  }

  @Override
  public void run() {
    bookingSystem.bookTicket(customerName, seatsRequested);
  }
}
```

```
public class TicketBookingSimulation {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(10); // 10 available seats

        // Creating threads for VIP and Regular customers
        BookingThread vip1 = new BookingThread(system, "VIP1", 2);
        BookingThread vip2 = new BookingThread(system, "VIP2", 3);
        BookingThread regular1 = new BookingThread(system, "Regular1", 2);
        BookingThread regular2 = new BookingThread(system, "Regular2", 4);
        BookingThread regular3 = new BookingThread(system, "Regular3", 1);

        // Assigning thread priorities
        vip1.setPriority(Thread.MAX_PRIORITY); // VIP gets the highest priority
        vip2.setPriority(Thread.MAX_PRIORITY);
        regular1.setPriority(Thread.NORM_PRIORITY);
        regular2.setPriority(Thread.NORM_PRIORITY);
        regular3.setPriority(Thread.MIN_PRIORITY); // Lowest priority

        // Starting threads (VIP bookings will be processed first due to priority)
        vip1.start();
        vip2.start();
        regular1.start();
        regular2.start();
        regular3.start();
```

```
    }
}
```

Output:

```
VIP1 successfully booked 2 seat(s).
Seats remaining: 8
VIP2 successfully booked 3 seat(s).
Seats remaining: 5
Regular1 successfully booked 2 seat(s).
Seats remaining: 3
Regular2 successfully booked 4 seats. Not enough seats available!
Seats remaining: 3
Regular3 successfully booked 1 seat(s).
Seats remaining: 2
```