**Q1 Easy Level: Employee Management System Problem Statement 📝 Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to:**

**Add employees**

**Update employee details**

**Remove employees**

**Search for employees**

**Key Concepts Used 🛠️ ArrayList: To store employee objects.**

**Encapsulation: Employee details are stored in a class with private fields and public getters/setters.**

**User Interaction: Using Scanner for input/output operations.**

**Code**

import java.util.ArrayList;

import java.util.Scanner;

class Employee {

private final 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 final ArrayList<Employee> employees = new ArrayList<>();

private static final 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(); // Consume newline

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! Please try again.");

}}} private static void addEmployee() {

System.out.print("Enter Employee ID: ");

int id = scanner.nextInt();

scanner.nextLine(); // Consume newline

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 name = scanner.nextLine();

System.out.print("Enter new Salary: ");

double salary = scanner.nextDouble();

emp.setName(name);

emp.setSalary(salary);

System.out.println("Employee 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();

scanner.nextLine();

employees.removeIf(emp -> emp.getId() == id);

System.out.println("Employee removed successfully!");

} private static void searchEmployee() {

System.out.print("Enter Employee ID to search: ");

int id = scanner.nextInt();

scanner.nextLine();

for (Employee emp : employees) {

if (emp.getId() == id) {

System.out.println("Employee Found: " + emp);

return;}}

System.out.println("Employee not found!");

} private static void displayEmployees() {

if (employees.isEmpty()) {

System.out.println("No employees found!");

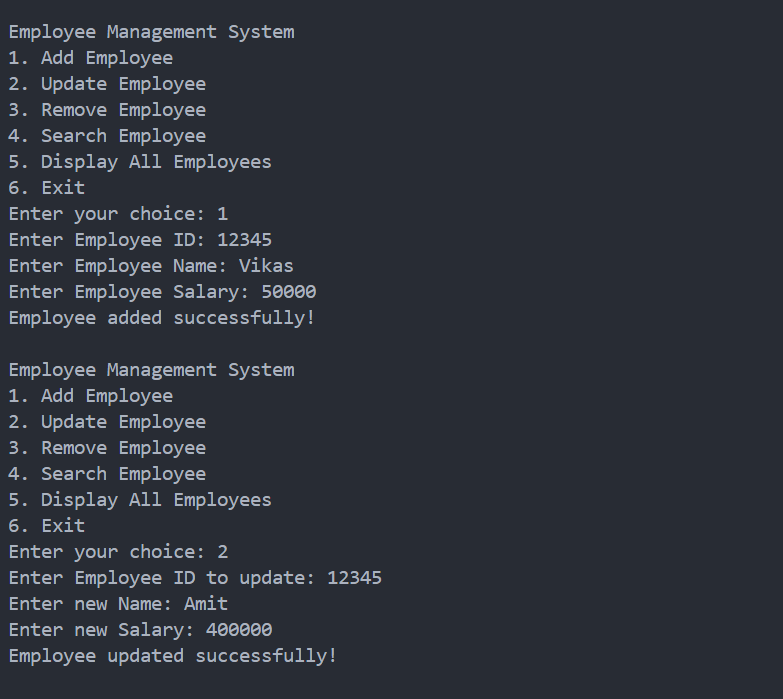
} else {

System.out.println("\nEmployee List:");

for (Employee emp : employees) {

System.out.println(emp);}}}}

**Output**

****

**Q2 Medium Level: Card Collection System Problem Statement 📝 Create a program to collect and store all the cards (e.g., playing cards) and assist users in finding all the cards of a given symbol using the Collection interface.**

**Key Concepts Used 🛠️ HashMap: To store cards with their symbols as keys.**

**Collection Interface: To manage and manipulate the card data.**

**User Interaction: Allow users to search for cards by symbol.**

**How to Run � Navigate to the Medium/ folder.**

**Compile and run the CardCollection.java file.**

**Enter the symbol (e.g., "Hearts", "Spades") to find all cards of that symbol.**

.

**Code**

import java.util.\*;

public class CardCollectionSystem {

    private Map<String, List<String>> cardCollection;

    public CardCollectionSystem() {

        cardCollection = new HashMap<>();

    }    public void addCard(String *symbol*, String *cardName*) {

        cardCollection.computeIfAbsent(*symbol*, *k* -> new ArrayList<>()).add(*cardName*);

    }    public Collection<String> getCardsBySymbol(String *symbol*) {

        return cardCollection.getOrDefault(*symbol*, Collections.emptyList());}

    public void displayAllCards() {

        for (Map.Entry<String, List<String>> entry : cardCollection.entrySet()) {

            System.out.println("Symbol: " + entry.getKey() + " -> Cards: " + entry.getValue());} }

    public static void main(String[] *args*) {

        CardCollectionSystem cardSystem = new CardCollectionSystem();

        Scanner scanner = new Scanner(System.in);

        cardSystem.addCard("Hearts", "Ace of Hearts");

        cardSystem.addCard("Hearts", "King of Hearts");

        cardSystem.addCard("Spades", "Queen of Spades");

        cardSystem.addCard("Diamonds", "Jack of Diamonds");

        cardSystem.addCard("Clubs", "10 of Clubs");

        System.out.println("Card Collection System");

        while (true) {

            System.out.println("\n1. Add a 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 choice: ");

            int choice = scanner.nextInt();

            scanner.nextLine();

            switch (choice) {

                case 1:

                    System.out.print("Enter card symbol (e.g., Hearts, Spades): ");

                    String symbol = scanner.nextLine();

                    System.out.print("Enter card name: ");

                    String cardName = scanner.nextLine();

                    cardSystem.addCard(symbol, cardName);

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

                    break;

                case 2:

                    System.out.print("Enter symbol to search: ");

                    String searchSymbol = scanner.nextLine();

                    Collection<String> cards = cardSystem.getCardsBySymbol(searchSymbol);

                    if (cards.isEmpty()) {

                        System.out.println("No cards found for symbol: " + searchSymbol);

                    } else {

                        System.out.println("Cards for " + searchSymbol + ": " + cards); }

                    break;

                case 3:

                    cardSystem.displayAllCards();

                    break;

                case 4:

                    System.out.println("Exiting program.");

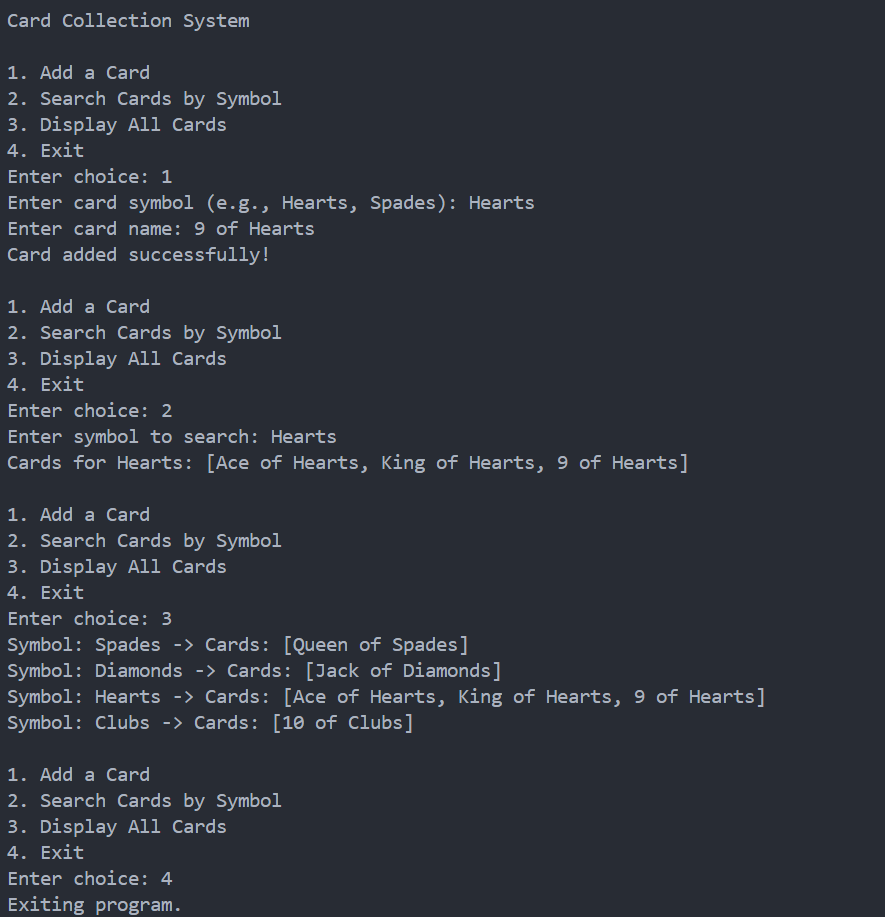
                    scanner.close();

                    return;

                default:

                    System.out.println("Invalid choice. Please try again.");}}}}

**Output**

****

**Q3 Hard Level: Ticket Booking System with Multithreading Problem Statement 📝 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.**

**Key Concepts Used 🛠️ Multithreading: To handle multiple booking requests simultaneously.**

**Synchronization: To prevent double booking of seats.**

**Thread Priorities: To prioritize VIP bookings over regular bookings.**

**Code**

import java.util.\*;

class TicketBookingSystem {

    private final int totalSeats;

    private final boolean[] seats;

    public TicketBookingSystem(int *totalSeats*) {

        this.totalSeats = *totalSeats*;

        this.seats = new boolean[*totalSeats*]; *// False = available, True = booked* }

    public synchronized boolean bookSeat(int *seatNumber*, String *customerType*) {

        if (*seatNumber* < 0 || *seatNumber* >= totalSeats) {

            System.out.println(*customerType* + " requested an invalid seat: " + *seatNumber*);

            return false; }

        if (!seats[*seatNumber*]) {

            seats[*seatNumber*] = true;

            System.out.println(*customerType* + " booked Seat " + *seatNumber*);

            return true;

        } else {

            System.out.println(*customerType* + " failed to book Seat " + *seatNumber* + " (Already Booked)");

            return false;}}}

class BookingThread extends Thread {

    private final TicketBookingSystem bookingSystem;

    private final int seatNumber;

    private final String customerType;

    public BookingThread(TicketBookingSystem *bookingSystem*, int *seatNumber*, String *customerType*, int *priority*) {

        this.bookingSystem = *bookingSystem*;

        this.seatNumber = *seatNumber*;

        this.customerType = *customerType*;

        this.setPriority(*priority*); *// Assign thread priority (higher value for VIP)* }

    @Override

    public void run() {

        bookingSystem.bookSeat(seatNumber, customerType);  }}

public class TicketBookingMain {

    public static void main(String[] *args*) {

        TicketBookingSystem bookingSystem = new TicketBookingSystem(5); *// 5 available seats*

        List<BookingThread> threads = new ArrayList<>();

        threads.add(new BookingThread(bookingSystem, 1, "VIP User", Thread.MAX\_PRIORITY));

        threads.add(new BookingThread(bookingSystem, 2, "Regular User", Thread.NORM\_PRIORITY));

        threads.add(new BookingThread(bookingSystem, 1, "Regular User", Thread.NORM\_PRIORITY)); *// Attempt to double book*

        threads.add(new BookingThread(bookingSystem, 3, "VIP User", Thread.MAX\_PRIORITY));

        threads.add(new BookingThread(bookingSystem, 4, "Regular User", Thread.NORM\_PRIORITY));

        threads.add(new BookingThread(bookingSystem, 0, "VIP User", Thread.MAX\_PRIORITY));

        for (BookingThread thread : threads) {

            thread.start();  }

        for (BookingThread thread : threads) {

            try {

                thread.join();

            } catch (InterruptedException *e*) {

                e.printStackTrace(); }}

        System.out.println("All booking requests processed."); }}

**Output**

