ASSIGNMENT - 4

Name : Amit Kumar Jha

UID: 22BCS10510

Q. 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 ge ers/se ers.

User Interaction: Using Scanner for input/output operations.

How to Run Navigate to the Easy/ folder.

Compile and run the EmployeeManagement.java file.

Follow the on-screen instructions to manage employee details.

Code:

```
import java.u l.*;


// Employee class with encapsula on 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 void setName(String name)
{          this.name = name;}      public
double getSalary() {
        return salary;
   }
   public void setSalary(double salary) {
        this.salary = salary;
   }
   @Override
   public String toString() {         return "ID: " + id + ", Name:
" + name + ", Salary: " + salary;
   }
}
public class EmployeeManagement {     private sta c
ArrayList<Employee> employees = new ArrayList<>();     private
sta c Scanner scanner = new Scanner(System.in);
```

```
    public sta c 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();
                    break;
case 2:
                    updateEmployee();
                    break;
case 3:
                    removeEmployee();
                    break;
case 4:
                    searchEmployee();
                    break;
case 5:
```

```
                displayEmployees();
                break;
case 6:
                System.out.println("Exi ng...");
                return;
default:
                System.out.println("Invalid choice! Please try again.");
   private sta c void addEmployee() {
System.out.print("Enter ID: ");
int id = scanner.nextInt();
scanner.nextLine();
       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!");
   }

   private sta c 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: ");
emp.setName(scanner.nextLine());
```

```
System.out.print("Enter new Salary: ");
emp.setSalary(scanner.nextDouble());
                System.out.println("Employee updated successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }

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

Q. 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.

Collec on 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.u l.*; class CardCollection {
private Map<String, List<String>> cardMap;
public CardCollec on() {        cardMap = new
HashMap<>();
}    public void addCard(String symbol, String card) {
cardMap.putIfAbsent(symbol, new ArrayList<>());
cardMap.get(symbol).add(card);
    }
```

```
    public Collec on<String> getCardsBySymbol(String symbol) {
return cardMap.getOrDefault(symbol, Collec ons.emptyList());
    }
    public sta c void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CardCollec on collec on = new CardCollec on();

        // Sample cards
        collec on.addCard("Hearts", "Ace of Hearts");
        collec on.addCard("Hearts", "King of Hearts");
        collec on.addCard("Spades", "Queen of Spades");
        collec on.addCard("Diamonds", "Jack of Diamonds");
        collec on.addCard("Clubs", "10 of Clubs");

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

        Collec on<String> cards = collec on.getCardsBySymbol(symbol);
        if (cards.isEmpty()) {
            System.out.println("No cards found for symbol: " + symbol);
        } else {
            System.out.println("Cards of " + symbol + ": " + cards);
        }

        scanner.close();
    }
}
```

Q) Hard Level: Ticket Booking System with Mul threading Problem Statement Develop a bascket booking system with synchronized threads to ensure no double booking of seats. Use thread priori es to simulate VIP bookings being processed first.

Key Concepts Used Mul threading: To handle multiple booking requests simultaneously.

Synchronization: To prevent double booking of seats.

Thread Priori es: To prioritize VIP bookings over regular bookings.

How to Run Navigate to the Hard/ folder.

Compile and run the TicketBookingSystem.java file.

Observe how VIP bookings are priori zed and how synchronization prevents double booking.

Codde:

```
import java.u l.concurrent.locks.*;


class TicketBookingSystem {
   private int availableSeats = 5;     private final Lock lock = new
ReentrantLock(true); // Fair lock to ensure priority order


   public void bookTicket(String customerName) {
      lock.lock();
```

```
        try {
            if (availableSeats > 0) {
                System.out.println(customerName + " successfully booked a seat. Remaining
seats: " + (--availableSeats));
            } else {
                System.out.println(customerName + " failed to book. No seats available.");
            }
        } finally {
lock.unlock();
        }
    }
}

class BookingThread extends Thread {
private TicketBookingSystem system;
private String customerName;

    public BookingThread(TicketBookingSystem system, String customerName, int priority)
{
        this.system = system;
        this.customerName = customerName;
        setPriority(priority);
    }

    @Override
    public void run() {
        try {
            Thread.sleep(100); // Simulate processing delay
        } catch (InterruptedExcep on e) {
```

```
            Thread.currentThread().interrupt();
        }
        system.bookTicket(customerName);
    }
}

public class TicketBooking {     public
sta c void main(String[] args) {
TicketBookingSystem system = new
TicketBookingSystem();


        Thread vip1 = new BookingThread(system, "VIP1", Thread.MAX_PRIORITY);
        Thread vip2 = new BookingThread(system, "VIP2", Thread.MAX_PRIORITY);
        Thread regular1 = new BookingThread(system, "Regular1",
Thread.NORM_PRIORITY);
        Thread regular2 = new BookingThread(system, "Regular2",
Thread.NORM_PRIORITY);
                        Thread regular3 = new BookingThread(system, "Regular3",
      Thread.NORM_PRIORITY);         Thread regular4 = new BookingThread(system,
                                "Regular4", Thread.NORM_PRIORITY);


        vip1.start();
vip2.start();
regular1.start();
regular2.start();
regular3.start();
regular4.start();
```

```
        try {
vip1.join();
vip2.join();
regular1.join();
regular2.join();
regular3.join();
regular4.join();
        } catch (InterruptedExcep on e) {
            Thread.currentThread().interrupt();
        }

        System.out.println("All booking a empts are completed.");
    }
}
```