Assignment 4

Name: Trimaanpreet Kaur

UID: 22BCS13971

Section: 22BCS-IOT-605

Group: A

Easy

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. 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.util.ArrayList;

import java.util.Scanner;


// Employee class with encapsulation

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;
}

// Getters and Setters
public int getId() {
return id;
}

public void setId(int id) {
this.id = id;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public double getSalary() {
return salary;
}
```

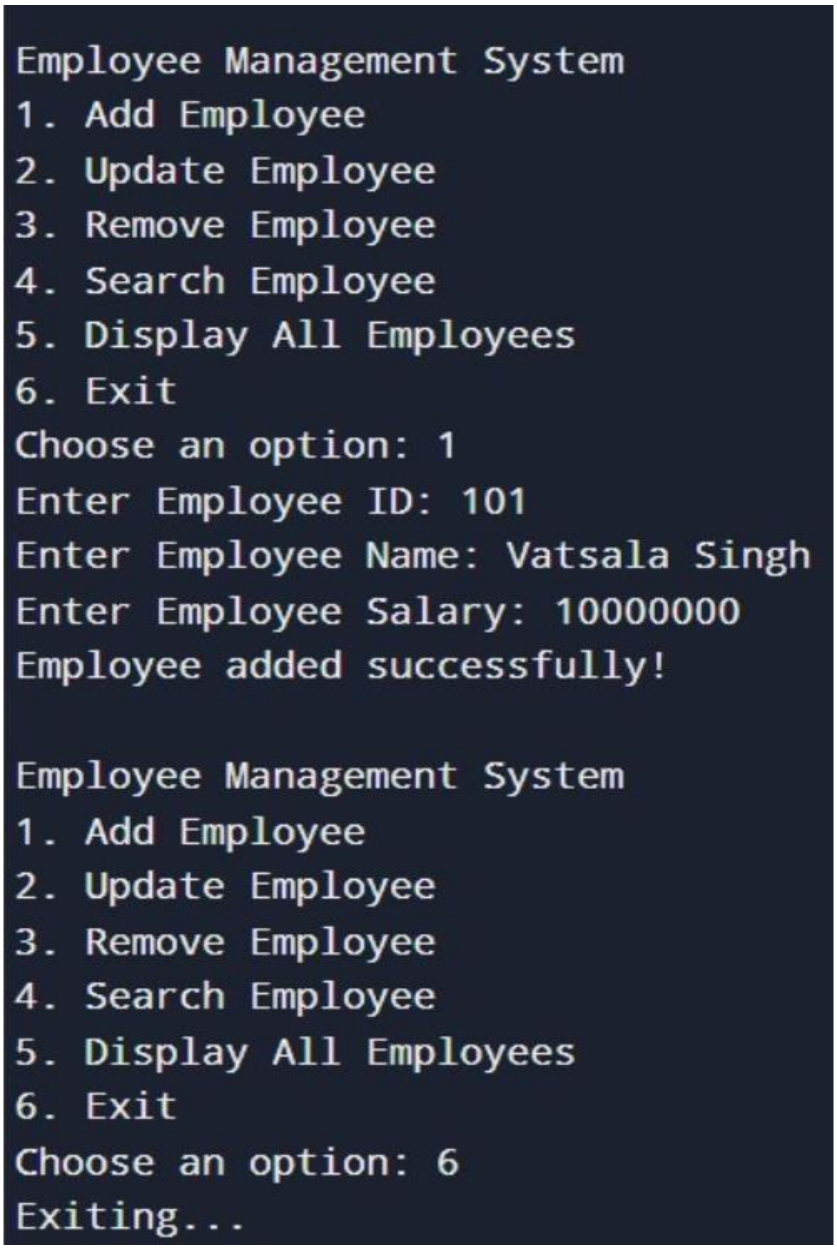
```
public void setSalary(double salary) {
this.salary = salary;
}

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

// Employee Management System
public class EmployeeManagement {
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("Choose an option: ");

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.");
}
}
}

// Add Employee
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!");
```

```
// Update Employee
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 details updated!");
return;
}
}
System.out.println("Employee not found!");
}

// Remove Employee
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!");
}

// Search Employee
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("Employee Found: " + emp);
return;
}
}
System.out.println("Employee not found!");
}

// Display All Employees
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



```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 101
Enter Employee Name: Vatsala Singh
Enter Employee Salary: 10000000
Employee added successfully!

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 6
Exiting...
```

Medium

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.

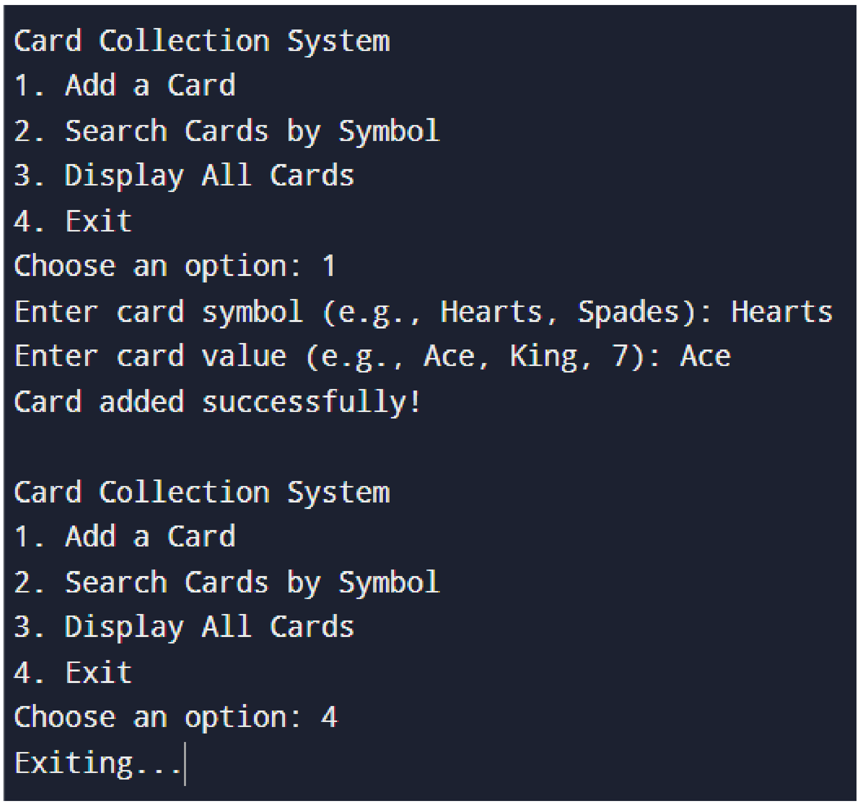
Code

```
import java.util.*;

public class CardCollection {

private static final Map<String, List<String>> cardCollection = new HashMap<>();

private static final Scanner scanner = new Scanner(System.in);


public static void main(String[] args)

{ while (true) {

System.out.println("\nCard Collection System");

System.out.println("1. 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("Choose an option: ");


int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline


switch (choice) {
```

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

// Method to add a card
private static void addCard() {
System.out.print("Enter card symbol (e.g., Hearts, Spades): ");
String symbol = scanner.nextLine().trim();
System.out.print("Enter card value (e.g., Ace, King, 7):
"); String value = scanner.nextLine().trim();

cardCollection.putIfAbsent(symbol, new ArrayList<>());
cardCollection.get(symbol).add(value);

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

// Method to search for cards by symbol private
static void searchCardsBySymbol() {
System.out.print("Enter symbol to search (e.g., Hearts, Spades): ");
```

```
String symbol = scanner.nextLine().trim();


List<String> cards =
cardCollection.get(symbol); if (cards != null
&& !cards.isEmpty()) {
System.out.println("Cards with symbol " + symbol + ": " + cards);
} else {
System.out.println("No cards found for the symbol: " + symbol);
}
}


// Method to display all stored cards
private static void displayAllCards()
{ if (cardCollection.isEmpty()) {
System.out.println("No cards in the collection.");
} else {
System.out.println("\nAll Stored Cards:");
for (Map.Entry<String, List<String>> entry : cardCollection.entrySet()) {
System.out.println(entry.getKey() + " -> " + entry.getValue());
}
}
}
}
```

Output



```
Card Collection System
1. Add a Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 1
Enter card symbol (e.g., Hearts, Spades): Hearts
Enter card value (e.g., Ace, King, 7): Ace
Card added successfully!

Card Collection System
1. Add a Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 4
Exiting...
```

Hard

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.

How to Run 🔋 Navigate to the Hard/ folder.

Compile and run the TicketBookingSystem.java file.

Observe how VIP bookings are prioritized and how synchronization prevents double booking.

Code

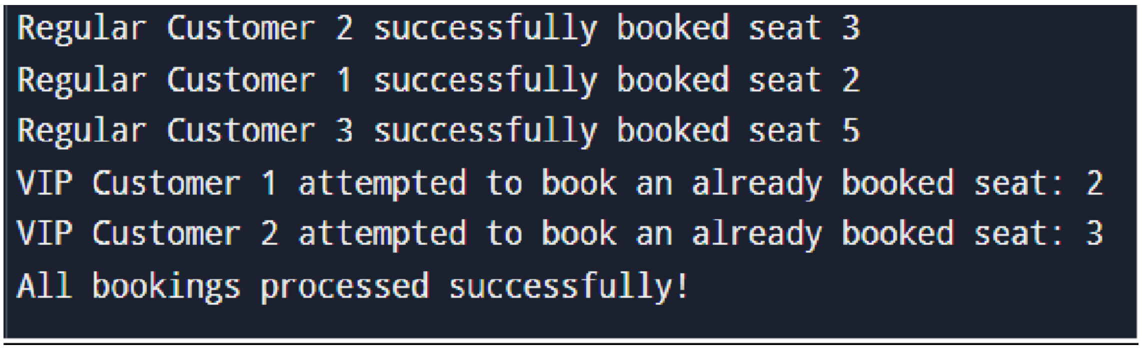
```
import java.util.*;

class TicketBookingSystem {
```

```
private static final int TOTAL_SEATS = 10; // Total available seats

private static final boolean[] seats = new boolean[TOTAL_SEATS]; // Seat availability

private static final Object lock = new Object(); // Synchronization lock


// Booking method

public static void bookSeat(String customerType, int seatNumber) {

synchronized (lock) { // Ensure thread safety

if (seatNumber < 0 || seatNumber >= TOTAL_SEATS) {

System.out.println(customerType + " attempted to book an invalid seat: " +
seatNumber);

return;

}

if (!seats[seatNumber]) {

seats[seatNumber] = true; // Mark seat as booked

System.out.println(customerType + " successfully booked seat " + seatNumber);

} else {

System.out.println(customerType + " attempted to book an already booked seat: " +
seatNumber);

}

}

}

}


// Booking Thread Class

class BookingThread extends Thread {

private final String customerType;

private final int seatNumber;


public BookingThread(String customerType, int seatNumber, int priority) {
```

```
this.customerType = customerType;

this.seatNumber = seatNumber;

this.setPriority(priority); // Set thread priority (higher for VIPs)

}


@Override

public void run() {

TicketBookingSystem.bookSeat(customerType, seatNumber);

}

}


// Main Class

public class TicketBookingMain {

public static void main(String[] args)

{

List<Thread> bookings = new ArrayList<>();


// Creating bookings with VIP customers having higher priority

bookings.add(new BookingThread("VIP Customer 1", 2, Thread.MAX_PRIORITY));

bookings.add(new BookingThread("VIP Customer 2", 3, Thread.MAX_PRIORITY));

bookings.add(new BookingThread("Regular Customer 1", 2,
Thread.MIN_PRIORITY));

bookings.add(new BookingThread("Regular Customer 2", 3,
Thread.MIN_PRIORITY));

bookings.add(new BookingThread("Regular Customer 3", 5,
Thread.NORM_PRIORITY));


// Shufle the list to simulate random booking requests

Collections.shufle(bookings);


// Start all threads
```

```
for (Thread booking : bookings) {
booking.start();
}

// Wait for all threads to finish
for (Thread booking : bookings) {
try {
booking.join();
} catch (InterruptedException e) {
e.printStackTrace();
}
}

System.out.println("All bookings processed successfully!");
}
}
```

<u>Output</u>



```
Regular Customer 2 successfully booked seat 3
Regular Customer 1 successfully booked seat 2
Regular Customer 3 successfully booked seat 5
VIP Customer 1 attempted to book an already booked seat: 2
VIP Customer 2 attempted to book an already booked seat: 3
All bookings processed successfully!
```