Experiment 4

Student Name: Piyushree Rani Sinha **UID: 22BCS13178**
Branch: BE-CSE **Section/Group: EPAM_801-B**
Semester: 6th **Date of Performance: 17/2/25**
Subject Name: Project Based Java Lab **Subject Code: 22CSP-359**

1. **Aim:**

   A- Easy Level: 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.

   B- Medium Level: 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.

   C- Hard Level: 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:**

   A- The objective of this program is to develop a Java-based employee management system using core data structures such as ArrayList. The system allows users to add, update, remove, search, and display employee details efficiently as it dynamically manages employee records without requiring predefined storage limits and ensures user-friendly interaction, making it easy to perform CRUD (Create, Read, Update, Delete) operations on employee data.

   B- The objective of this program is to develop a Java application that allows users to store and retrieve all cards based on their symbols using the Collection Framework. Users can add new cards, search for all cards under a particular symbol, and display all stored cards.

   C- The objective of this Ticket Booking System is to develop a concurrent, thread-safe mechanism for booking seats while ensuring that no double booking occurs. This system utilizes multithreading with synchronized

methods to prevent race conditions when multiple passengers attempt to book seats simultaneously, such as airline or railway reservation systems.

3. **Implementation/Code:**

A-
```
import java.util.Scanner;
import java.util.ArrayList;
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 empDetails{
    static ArrayList<Employee> empList = new ArrayList<>();
    static Scanner sc = new Scanner(System.in);
    public static void addEmployee(){
        System.out.print("Enter Employee ID: ");
        int id = sc.nextInt();
        System.out.print("Enter Employee Name: ");
        String name = sc.next();
        System.out.print("Enter Employee Salary: ");
        double salary = sc.nextDouble();
        empList.add(new Employee(id, name, salary));
        System.out.println("Employee Added Successfully.");
    }
    public static void removeEmployee(){
```

```
        System.out.print("Enter employee ID to be removed: ");
        int id = sc.nextInt();
        for(int i = 0; i < empList.size(); i++){
            if(empList.get(i).id == id){
                empList.remove(i);
                System.out.println("Employee removed successfully.");
                return;
            }
        }
        System.out.println("Employee not found by this ID.");
    }
    public static void updateEmployee(){
        System.out.print("Enter employee ID to be updated: ");
        int id = sc.nextInt();
        for(int i = 0; i < empList.size(); i++){
            if(empList.get(i).id == id){
                System.out.print("Enter new employee name: ");
                String name = sc.next();
                System.out.print("Enter its salary: ");
                double salary = sc.nextDouble();
                empList.get(i).name = name;
                empList.get(i).salary = salary;
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found by this ID.");
    }
    public static void searchEmployee(){
        System.out.print("Enter employee ID to be searched: ");
        int id = sc.nextInt();
        for(int i = 0; i < empList.size(); i++){
```

```
            if(empList.get(i).id == id){
                System.out.println(empList.get(i));
                return;
            }
        }
        System.out.println("Employee not found with this ID.");
    }
    public static void displayEmployees(){
        if(empList.isEmpty()){
            System.out.println("No employees found.");
        }
        for(Employee e : empList){
            System.out.println((e));
        }
    }
    public static void main(String[] args){
        while(true){
            System.out.println(".....Employee Management System.....");
            System.out.print("1- Add Employee:\n2- Remove Employee:\n3-
Update Employee:\n4- Search Employee:\n5- Display all Employees:\n6-
Exit!!!\n");
            System.out.print("Enter the choice: ");
            int choice = sc.nextInt();
            switch(choice){
                case 1:
                    addEmployee();
                    break;
                case 2:
                    removeEmployee();
                    break;
                case 3:
                    updateEmployee();
```

```
                    break;
                case 4:
                    searchEmployee();
                    break;
                case 5:
                    displayEmployees();
                    break;
                case 6:
                    System.out.println("Exiting the program.");
                    sc.close();
                default:
                    System.out.println("Invalid choice made, try again.");
            }
        }
    }
}
```

B-

```
import java.util.*;
class Card{
    String symbol;
    String value;
    public Card(String symbol, String value){
        this.symbol = symbol;
        this.value = value;
    }
    public void print(){
        System.out.println("Symbol: " + symbol + " Value: " + value);
    }
}
public class CardCollection{
    public static void main(String[] args){
        Map<String, List<Card>> cardCollection = new HashMap<>();
        Scanner sc = new Scanner(System.in);
        while(true){
            System.out.println(".....Card Collection System.....");
            System.out.print("1- Add Card:\n2- Search Cards by
```

```
symbol:\n3- Display all cards:\n4- Exit!!!\n");
            System.out.print("Enter the choice: ");
            int ch = sc.nextInt();
            switch(ch){
                case 1:
                    System.out.print("Enter the symbol of the card: ");
                    String symbol = sc.next();
                    System.out.print("Enter the value of the card: ");
                    String value = sc.next();
                    Card card = new Card(symbol, value);
                    if(cardCollection.containsKey(symbol)){
                        System.out.println("Card already exists.");

System.out.println(cardCollection.get(symbol).add(card));
                    }
                    else{
                        List<Card> cards = new ArrayList<>();
                        cards.add(card);
                        cardCollection.put(symbol, cards);
                        System.out.println("Card added successfully!");
                    }
                    break;
                case 2:
                    System.out.print("Enter the symbol of the card to be
searched: ");
                    String symbolToSearch = sc.next();
                    if(cardCollection.containsKey(symbolToSearch)) {
                        System.out.println("Cards found.");
                        for (Card c : cardCollection.get(symbolToSearch)) {
                            c.print();
                        }
                    }
                    else{
                        System.out.println("No cards are found for this
symbol.");
                    }
                    break;
```

```
                case 3:
                    System.out.println("All cards in Collection.");
                    for(String s : cardCollection.keySet()){
                        for(Card c : cardCollection.get(s)){
                            c.print();
                        }
                    }
                    break;
                case 4:
                    System.out.println("Exiting....");
                    sc.close();
                    return;
                default:
                    System.out.println("Invalid choice done!!");
            }
        }
    }
}
```

C-

```
import java.util.concurrent.*;
class BookTicket{
    private static int availableSeats = 10;
    public synchronized void bookTicket(String passenger, int
noOfSeats){
        if(availableSeats >= noOfSeats){
            System.out.println(passenger + " has successfully booked " +
noOfSeats + " seats.");
            availableSeats = availableSeats - noOfSeats;
        }
        else{
            System.out.println("Not enough seats for " + passenger);
        }
    }
}
class Passenger extends Thread{
    private BookTicket bookingSystem;
```

```
    private int seats;
    private String passengerName;
    public Passenger(BookTicket bookingSystem, String passengerName,
int seats, int priority){
        this.bookingSystem = bookingSystem;
        this.seats = seats;
        this.passengerName = passengerName;
        setPriority(priority);
    }
    public void run(){
        bookingSystem.bookTicket(passengerName, seats);
    }
}
public class TicketBookingSystem{
    public static void main(String[] args){
        BookTicket bookingSystem = new BookTicket();
        Passenger p1 = new Passenger(bookingSystem,"Andrew", 2,
Thread.MAX_PRIORITY);
        Passenger p2 = new Passenger(bookingSystem,"Jenny", 1,
Thread.MAX_PRIORITY);
        Passenger p3 = new Passenger(bookingSystem,"Mike", 3,
Thread.MAX_PRIORITY);
        Passenger p4 = new Passenger(bookingSystem,"Annie", 4,
Thread.MAX_PRIORITY);
        Passenger p5 = new Passenger(bookingSystem,"Ben", 2,
Thread.MAX_PRIORITY);
        p1.start();
        p2.start();
        p3.start();
        p4.start();
        p5.start();
```

```
    }
  }
```

4. **Output:**

A-

```
Employee Added Successfully.
.........Employee Management System.........
1- Add Employee:
2- Remove Employee:
3- Update Employee:
4- Search Employee:
5- Display all Employees:
6- Exit!!!
Enter the choice: 1
Enter Employee ID: 104
Enter Employee Name: Hiteshi
Enter Employee Salary: 4100000
Employee Added Successfully.
.........Employee Management System.........
1- Add Employee:
2- Remove Employee:
3- Update Employee:
4- Search Employee:
5- Display all Employees:
6- Exit!!!
Enter the choice: 1
Enter Employee ID: 105
Enter Employee Name: Shivani
Enter Employee Salary: 1500000
Employee Added Successfully.
```

```
Run        empDetails
6- Exit!!!
Enter the choice: 4
Enter employee ID to be searched: 102
ID: 102 || Name: Piyushree || Salary: 6000000.0
..........Employee Management System..........
1- Add Employee:
2- Remove Employee:
3- Update Employee:
4- Search Employee:
5- Display all Employees:
6- Exit!!!
Enter the choice: 5
ID: 101 || Name: Aayushi || Salary: 4500000.0
ID: 102 || Name: Piyushree || Salary: 6000000.0
ID: 103 || Name: Sanya || Salary: 4000000.0
ID: 104 || Name: Gavy || Salary: 4200000.0
..........Employee Management System..........
1- Add Employee:
2- Remove Employee:
3- Update Employee:
4- Search Employee:
5- Display all Employees:
6- Exit!!!
Enter the choice: 6
Exiting the program.
Invalid choice made, try again.
```

B-

```
C:\Users\HP\.jdks\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
3- Display all cards:
4- Exit!!!
Enter the choice: 1
Enter the symbol of the card: Spade
Enter the value of the card: A
Card added successfully!
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
3- Display all cards:
4- Exit!!!
Enter the choice: 1
Enter the symbol of the card: Diamond
Enter the value of the card: 2
Card added successfully!
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
3- Display all cards:
4- Exit!!!
Enter the choice: 2
Enter the symbol of the card to be searched: Spade
Cards found.
Symbol: Spade Value: A
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
```

```
Enter the value of the card: 2
Card added successfully!
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
3- Display all cards:
4- Exit!!!
Enter the choice: 2
Enter the symbol of the card to be searched: Spade
Cards found.
Symbol: Spade Value: A
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
3- Display all cards:
4- Exit!!!
Enter the choice: 3
All cards in Collection.
Symbol: Spade Value: A
Symbol: Diamond Value: 2
..........Card Collection System..........
1- Add Card:
2- Search Cards by symbol:
3- Display all cards:
4- Exit!!!
Enter the choice: 4
Exiting....

Process finished with exit code 0
```

C-

```
C:\Users\HP\.jdks\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
Jenny has successfully booked 1 seats.
Ben has successfully booked 2 seats.
Annie has successfully booked 4 seats.
Mike has successfully booked 3 seats.
Not enough seats for Andrew

Process finished with exit code 0
```

5. Complexity:

A- Since the operations involve searching for an employee by ID before updating or deleting, the overall worst-case complexity is O(n). However, adding an employee is O(1), making it efficient for inserting new records.

B- For hashmap insertion, hashmap lookup and adding to arrayList, complexity will be O(1). Iterating through all symbols (M symbols), TC will be O(M) and for iterating through all cards (N total cards across all symbols) TC will be O(N). Total Complexity: O(N + M) O(N) (Since N is usually much larger).

C- O(1) to book the ticket and O(n) each thread processes one booking.

6. **Learning Outcomes**

A- Understanding Java Collections.
B- CRUD operations.
C- Handling duplicates.
D- Sorting and Filtering.
E- Concurrency Handling.
F- Multithreading in java.