Experiment 4

Student Name: Lakshay	**UID: 22BCS16259**
Branch: B.E CSE	**Section: IOT-643-A**
Semester: 6th	**DOP:24/02/25**
Subject: PBLJ	**Subject Code: 22CSH-359**

Aim:

Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

Problem Statement :

1) 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.

2) 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.

3) 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.

Algorithm:

1. Employee Management (Using ArrayList)

- Initialize an ArrayList to store employees.
- Display a menu with options: Add, Update, Remove, Search, and Exit.
- **Add Employee**:
  - Take user input for ID, Name, and Salary.
  - Create an Employee object and add it to the list.
- **Update Employee**:
  - Ask for the Employee ID.
  - If found, update Name and Salary.
- **Remove Employee**:
  - Ask for the Employee ID.
  - Remove matching employee from the list.
- **Search Employee**:
  - Ask for the Employee ID.
  - If found, display details.
- Repeat until the user chooses to exit.

2. Card Collection (Using Collections)

- Initialize an ArrayList to store Card objects.
- Display a menu with options: Add Card, Find Cards by Symbol, and Exit.
- **Add Card**:
  - Ask for card symbol (e.g., Hearts, Diamonds).
  - Ask for card value (A, 2, 3, ... J, Q, K).
  - Create a Card object and store it in the list.
- **Find Cards by Symbol**:
  - Ask for a symbol.
  - Search and display all cards with that symbol.
- Repeat until the user chooses to exit.

3. Ticket Booking System (Multithreading)

- Create a TicketBookingSystem with a limited number of seats.
- Implement synchronized booking to prevent double booking.
- Create Customer threads with different priorities (VIP first).
- **Each Customer thread**:
  - Tries to book a ticket.
  - If seats are available, booking is confirmed, and the seat count decreases.
  - If not, booking fails.
- Start all customer threads and process bookings.
- Stop when all threads have completed execution.

Program :

1. Employee Management:

```
import java.util.ArrayList;
import java.util.Scanner;
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 EmployeeManager {
    static ArrayList<Employee> employees = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);
    public static 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!");
    }
    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                scanner.nextLine();
                System.out.print("Enter New Name: ");
                emp.name = scanner.nextLine();
                System.out.print("Enter New Salary: ");
                emp.salary = scanner.nextDouble();
                System.out.println("Employee updated successfully!");
                return;
       } }
        System.out.println("Employee not found.");
    }
    public static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        employees.removeIf(emp -> emp.id == id);
        System.out.println("Employee removed successfully!");
    }
    public static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found.");
    }
    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> System.exit(0);
                default -> System.out.println("Invalid choice! Try again.");
      } } } }
```

2. Card Collection :

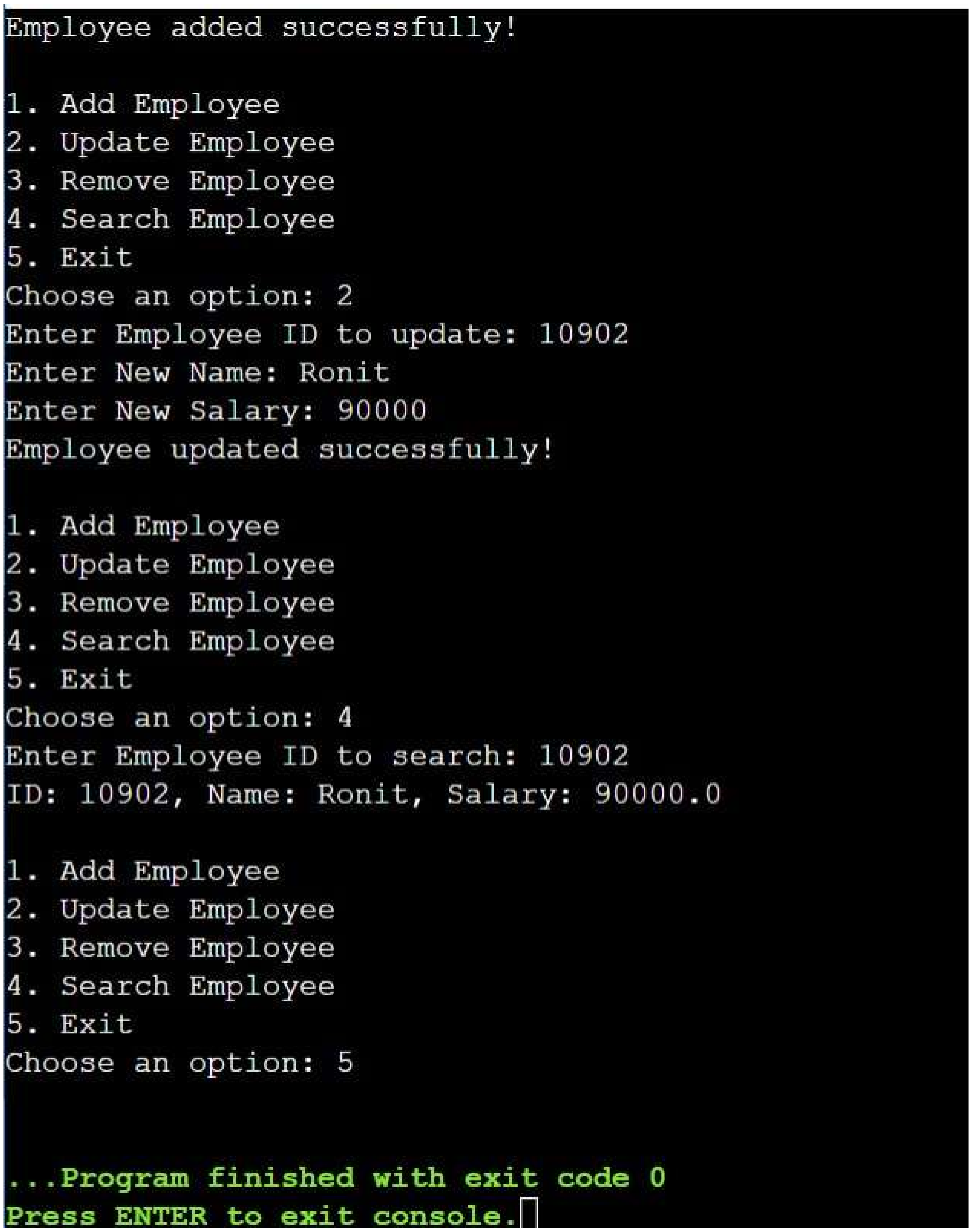
```
import java.util.*;
class Card {
    String symbol;
    String value;
    Card(String symbol, String value) {
        this.symbol = symbol;
        this.value = value;
    }
    public String toString() {
        return value + " of " + symbol;
    }
}
public class CardCollection {
    static ArrayList<Card> deck = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);
    public static void addCard() {
        System.out.print("Enter Symbol (Hearts, Diamonds, etc.): ");
        String symbol = scanner.next();
        System.out.print("Enter Value (A, 2, 3, ... J, Q, K): ");
        String value = scanner.next();
        deck.add(new Card(symbol, value));
        System.out.println("Card added successfully!");
    }
    public static void findCardsBySymbol() {
        System.out.print("Enter Symbol to search for: ");
        String symbol = scanner.next();
        System.out.println("Cards found:");
        for (Card card : deck) {
            if (card.symbol.equalsIgnoreCase(symbol)) {
                System.out.println(card);
    } }  }
    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Card\n2. Find Cards by Symbol\n3. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1 -> addCard();
                case 2 -> findCardsBySymbol();
                case 3 -> System.exit(0);
                default -> System.out.println("Invalid choice! Try again.");
        } } } }
```

3. Ticket Booking System:

```
import java.util.*;
class TicketBookingSystem {
    private int vipSeats, regularSeats;
    public TicketBookingSystem(int vipSeats, int regularSeats) {
        this.vipSeats = vipSeats;
        this.regularSeats = regularSeats;
    }
    public synchronized boolean bookTicket(String customerName, String type) {
        if (type.equals("VIP") && vipSeats > 0) {
          System.out.println(customerName + " booked a VIP ticket. Remaining VIP seats: "
  + (--vipSeats));
            return true;
        }
else if (type.equals("Regular") && regularSeats > 0) {
            System.out.println(customerName + " booked a Regular ticket. Remaining
            Regular seats: " + (--regularSeats));
            return true;
        }
 else {
            System.out.println(customerName + " failed to book a " + type + " ticket
             (No " + type + " seats available).");
            return false;
        } } }
class Customer extends Thread {
    private TicketBookingSystem bookingSystem;
    private String name, type;
    public Customer(TicketBookingSystem bookingSystem, String name, String type,
    int priority) {
        this.bookingSystem = bookingSystem;
        this.name = name;
        this.type = type;
        this.setPriority(priority);
    }
    public void run() {
        bookingSystem.bookTicket(name, type);
    } }
public class TicketBooking {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of VIP seats: ");
        int vipSeats = scanner.nextInt();
        System.out.print("Enter number of Regular seats: ");
        int regularSeats = scanner.nextInt();
        TicketBookingSystem system = new TicketBookingSystem(vipSeats, regularSeats);
        System.out.print("Enter number of customers: ");
        int n = scanner.nextInt();
        scanner.nextLine();
        Customer[] customers = new Customer[n];
        for (int i = 0; i < n; i++) {
            System.out.print("Enter customer name: ");
            String name = scanner.nextLine();
            System.out.print("Enter priority (1 for VIP, 2 for Regular): ");
            int priority = scanner.nextInt();
            scanner.nextLine();
            String type = (priority == 1) ? "VIP" : "Regular";
            int threadPriority = (priority == 1) ? Thread.MAX_PRIORITY :
    Thread.NORM_PRIORITY;
            customers[i] = new Customer(system, name, type, threadPriority);
        }
        System.out.println("\nStarting booking process...");
        for (Customer c : customers) c.start();
        scanner.close();
     } }
```

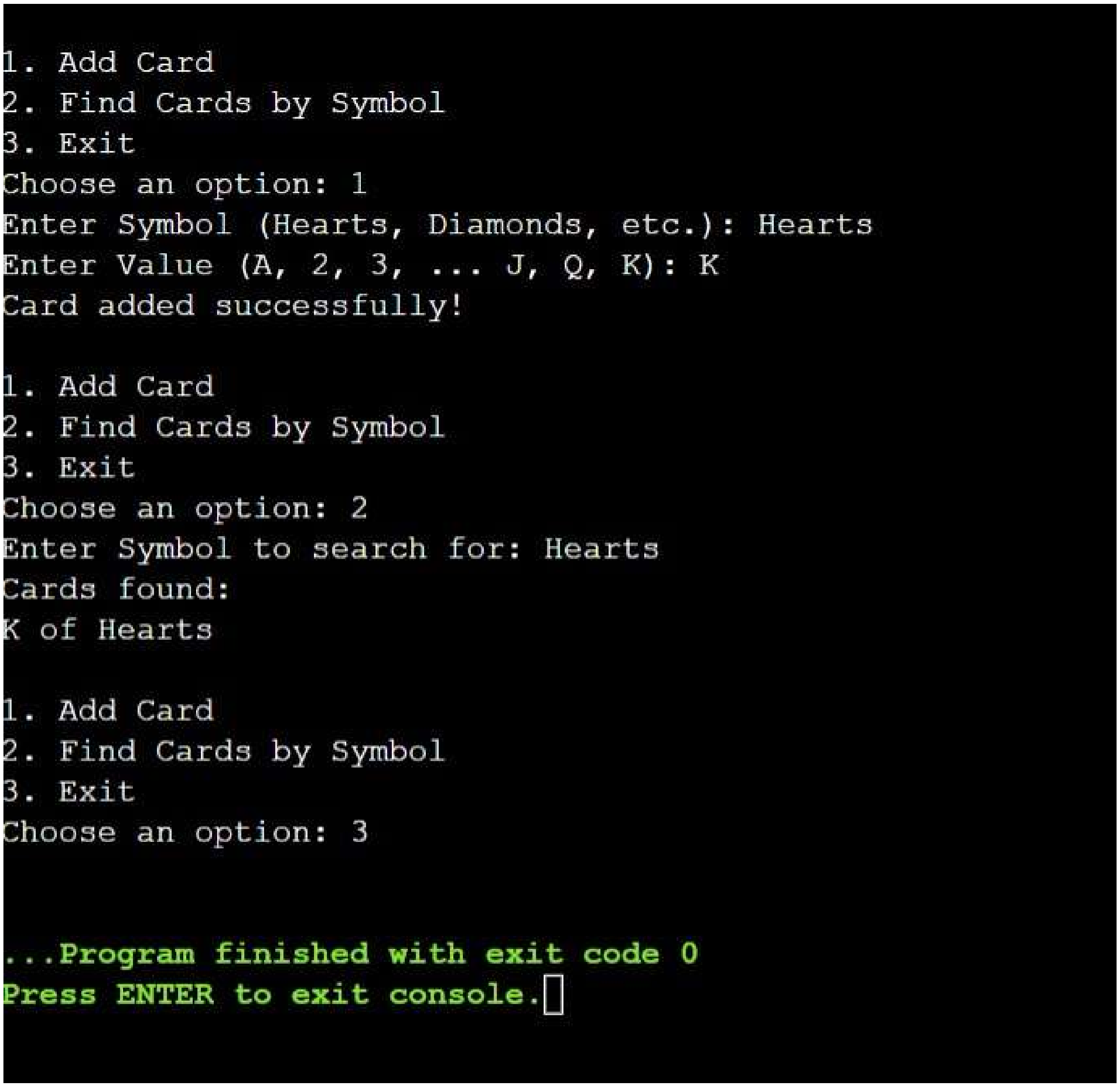
OUTPUT :

1. **Employee Management:**



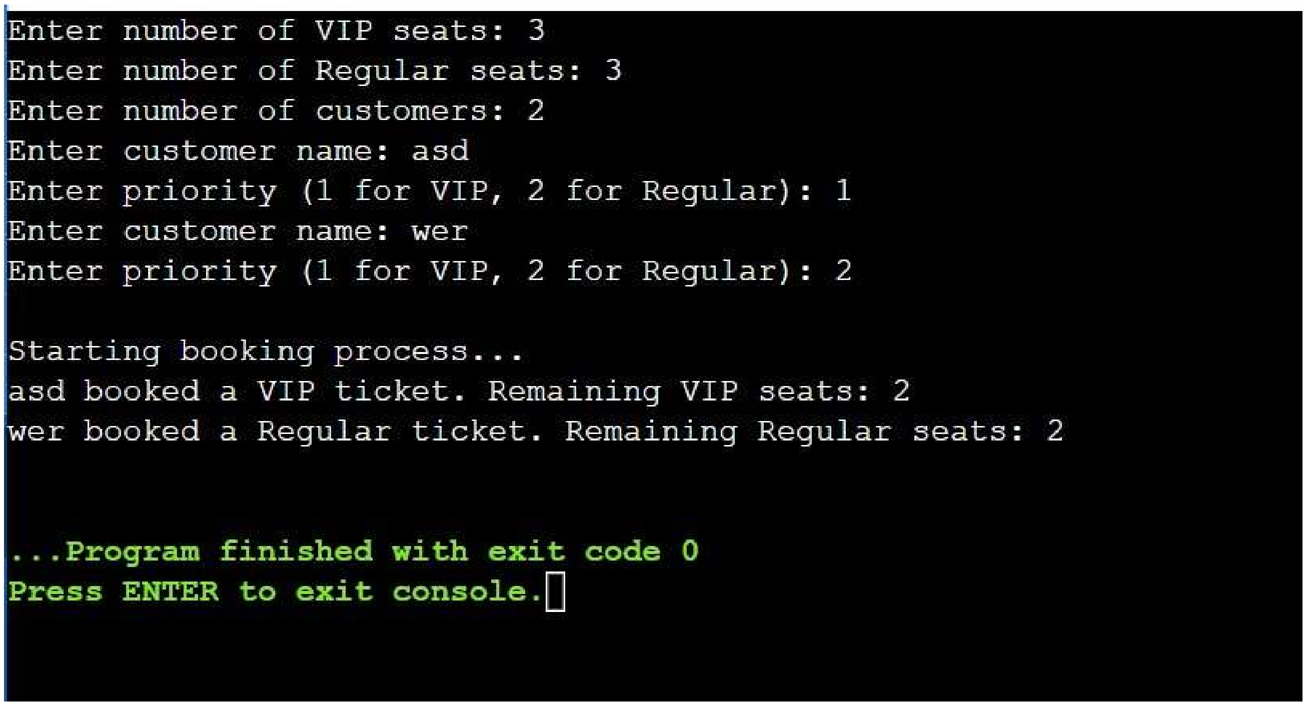
```
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 2
Enter Employee ID to update: 10902
Enter New Name: Ronit
Enter New Salary: 90000
Employee updated successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 4
Enter Employee ID to search: 10902
ID: 10902, Name: Ronit, Salary: 90000.0

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 5


...Program finished with exit code 0
Press ENTER to exit console.
```

2. **Card Collection :**



```
1. Add Card
2. Find Cards by Symbol
3. Exit
Choose an option: 1
Enter Symbol (Hearts, Diamonds, etc.): Hearts
Enter Value (A, 2, 3, ... J, Q, K): K
Card added successfully!

1. Add Card
2. Find Cards by Symbol
3. Exit
Choose an option: 2
Enter Symbol to search for: Hearts
Cards found:
K of Hearts

1. Add Card
2. Find Cards by Symbol
3. Exit
Choose an option: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

3. **Ticket Booking System:**



```
Enter number of VIP seats: 3
Enter number of Regular seats: 3
Enter number of customers: 2
Enter customer name: asd
Enter priority (1 for VIP, 2 for Regular): 1
Enter customer name: wer
Enter priority (1 for VIP, 2 for Regular): 2

Starting booking process...
asd booked a VIP ticket. Remaining VIP seats: 2
wer booked a Regular ticket. Remaining Regular seats: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

- **Object-Oriented Design** (Classes for real-world entities)
- **Core Programming Skills** (Loops, conditionals, methods for inventory operations)
- **Data Structure Usage** (ArrayList for dynamic data management)
- **User-Friendly Systems** (Intuitive interface with error handling)