Experiment 4

StudentName:Harsh Tandon **UID:22BCS14789**
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.*;

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 List<Employee> employees = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);

    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.");
    }

    static void updateEmployee() {
        System.out.print("Enter ID: ");
        int id = scanner.nextInt();
        for (Employee e : employees) {
            if (e.id == id) {
                scanner.nextLine();
                System.out.print("New Name: ");
                e.name = scanner.nextLine();
                System.out.print("New Salary: ");
                e.salary = scanner.nextDouble();
                System.out.println("Updated.");
                return;
            }
        }
        System.out.println("Not found.");
    }

    static void removeEmployee() {
        System.out.print("Enter ID: ");
        int id = scanner.nextInt();
        employees.removeIf(e -> e.id == id);
        System.out.println("Removed.");
    }

    static void searchEmployee() {
```

```
        System.out.print("Enter ID: ");
        int id = scanner.nextInt();
        employees.stream().filter(e -> e.id == id).findFirst()
                .ifPresentOrElse(System.out::println, () -> System.out.println("Not found."));
    }

    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add  2. Update  3. Remove  4. Search  5. Exit");
            System.out.print("Choice: ");
            switch (scanner.nextInt()) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> {
                    System.out.println("Goodbye!");
                    return;
                }
                default -> System.out.println("Invalid choice.");
            }  }}}
```

2. Card Collection :

```
import java.util.*;

class Employee {
    private final int id;
    private String name;
    private double salary;

    Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    public int getId() {
        return id;
    }

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

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

    @Override
    public String toString() {
        return String.format("ID: %d, Name: %s, Salary: %.2f", id, name, salary);
    }
}

public class EmployeeManager {
    private static final List<Employee> employees = new ArrayList<>();
    private static final Scanner scanner = new Scanner(System.in);

    private 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.");
    }

    private static void updateEmployee() {
        System.out.print("Enter ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        Employee employee = findEmployeeById(id);
        if (employee != null) {
            System.out.print("Enter New Name: ");
            employee.setName(scanner.nextLine());
            System.out.print("Enter New Salary: ");
            employee.setSalary(scanner.nextDouble());
            System.out.println("Employee updated successfully.");
        } else {
            System.out.println("Employee not found.");
        }
    }

    private static void removeEmployee() {
        System.out.print("Enter ID to remove: ");
```

```
        int id = scanner.nextInt();
        if (employees.removeIf(e -> e.getId() == id)) {
            System.out.println("Employee removed successfully.");
        } else {
            System.out.println("Employee not found.");
        }
    }

    private static void searchEmployee() {
        System.out.print("Enter ID to search: ");
        int id = scanner.nextInt();
        Employee employee = findEmployeeById(id);
        if (employee != null) {
            System.out.println(employee);
        } else {
            System.out.println("Employee not found.");
        }
    }

    private static Employee findEmployeeById(int id) {
        return employees.stream().filter(e -> e.getId() == id).findFirst().orElse(null);
    }

    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Employee  2. Update Employee  3. Remove Employee  4.
Search Employee  5. 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.out.println("Exiting program. Goodbye!");
                    return;
                }
                default -> System.out.println("Invalid choice. Please try again.");
            }}}}
```

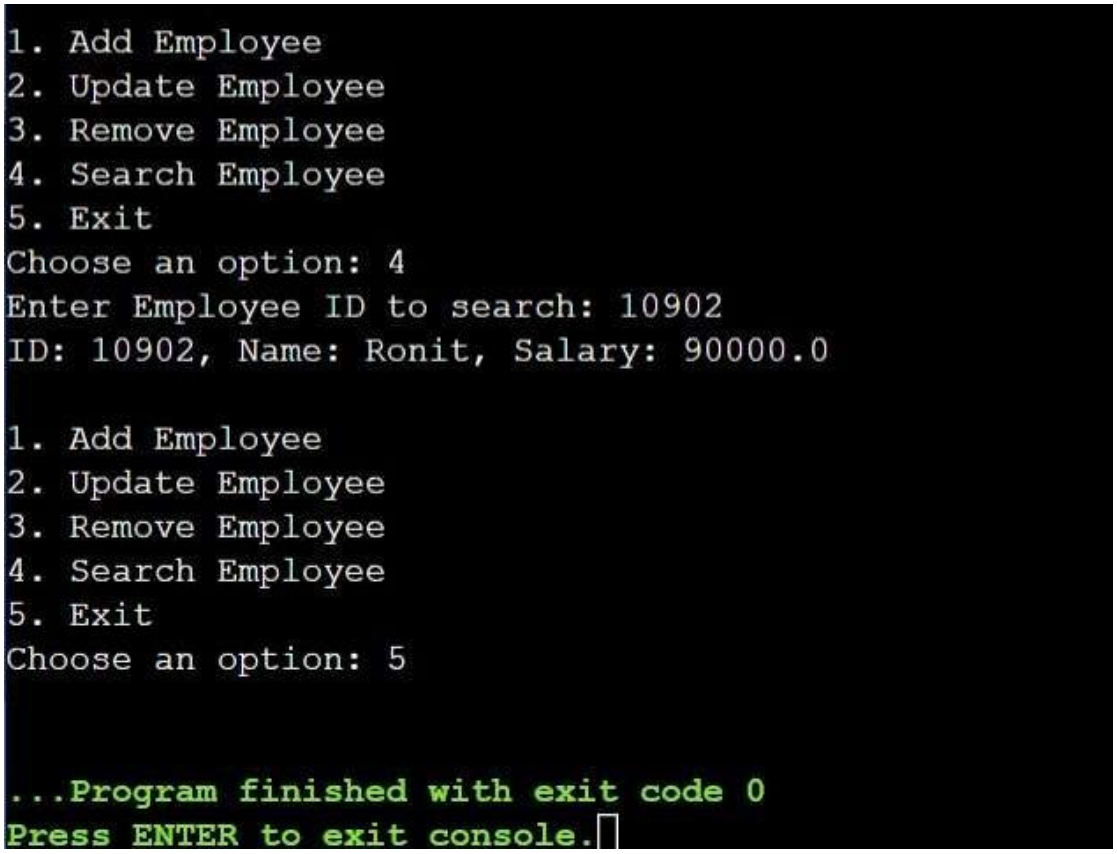
3. Ticket Booking System:

```
    import java.util.*;
```

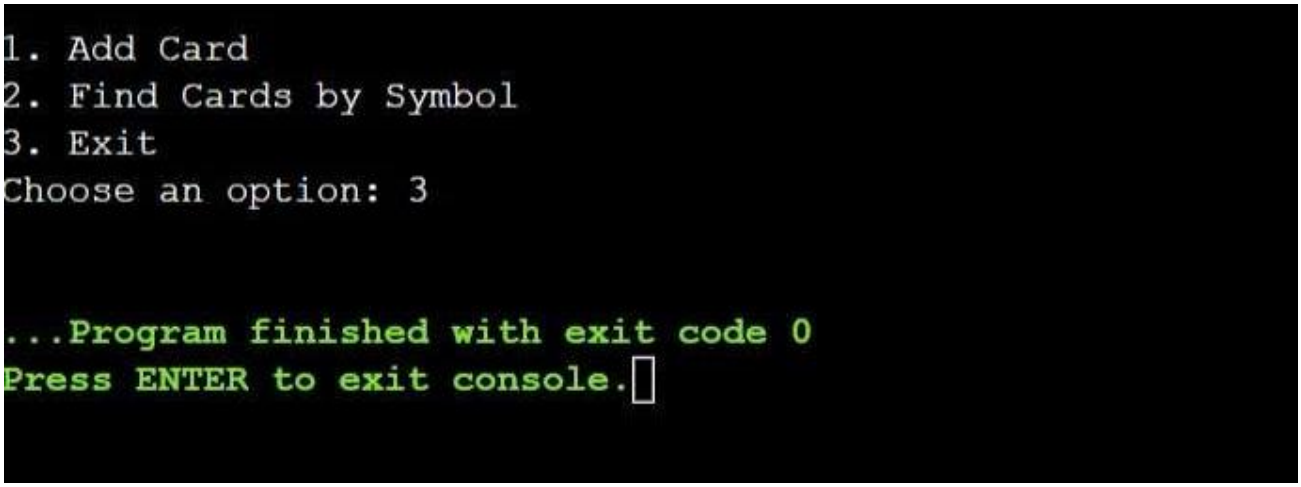
```
class TicketBookingSystem {
    private int vipSeats, regularSeats;

    public TicketBookingSystem(int vipSeats, int regularSeats) {
        this.vipSeats = vipSeats;
        this.regularSeats = regularSeats;
    }

    public synchronized void bookTicket(String name, String type) {
        if (type.equalsIgnoreCase("VIP") && vipSeats > 0) {
            vipSeats--;
            System.out.println(name + " booked a VIP ticket. Remaining: " + vipSeats);
        } else if (type.equalsIgnoreCase("Regular") && regularSeats > 0) {
            regularSeats--;
            System.out.println(name + " booked a Regular ticket. Remaining: " +
regularSeats);
        } else {
            System.out.println(name + " failed to book a " + type + " ticket.");
        }
    }
}

public class TicketBooking {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter VIP seats: ");
        int vipSeats = scanner.nextInt();
        System.out.print("Enter Regular seats: ");
        int regularSeats = scanner.nextInt();
        scanner.nextLine();

        TicketBookingSystem system = new TicketBookingSystem(vipSeats, regularSeats);

        System.out.print("Enter number of customers: ");
        int n = scanner.nextInt();
        scanner.nextLine();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter customer name: ");
            String name = scanner.nextLine();
            System.out.print("Enter ticket type (VIP/Regular): ");
            String type = scanner.nextLine();
            system.bookTicket(name, type);
        }

        scanner.close();}}
```

OUTPUT :

1. **Employee Management:**



2. **Card Collection :**



3. **Ticket Booking System:**

```
Enter VIP seats: 2
Enter Regular seats: 3
Enter number of customers: 5
Enter customer name: Alice
Enter ticket type (VIP/Regular): VIP
Alice booked a VIP ticket. Remaining: 1
Enter customer name: Bob
Enter ticket type (VIP/Regular): Regular
Bob booked a Regular ticket. Remaining: 2
Enter customer name: Charlie
Enter ticket type (VIP/Regular): VIP
Charlie booked a VIP ticket. Remaining: 0
Enter customer name: Dave
Enter ticket type (VIP/Regular): Regular
Dave booked a Regular ticket. Remaining: 1
Enter customer name: Eve
Enter ticket type (VIP/Regular): VIP
Eve failed to book a VIP ticket.
```

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)