Experiment 2.1

Student Name: Sushant Singh **UID: 22BCS10100**
Branch: CSE **Section: 640-A**
Semester: 6th **DOP:13-01-2025**
Subject: Java **Subject Code: 22CSH-359**

1. Aim: Implement and manage data using Java Collections and Threads.

2. Problem Statements:

- **Easy Level:** Implement an ArrayList to store employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

- **Medium Level:** Create a program to collect and store all the cards to assist users in finding all the cards in a given symbol using the Collection interface.

- **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.

3. Implementation/Code:

Problem 1.1: Employee Management System

```
import java.util.*;
class Employee {
    int id;
    String name;
    double salary;

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

    @Override
    public String toString() {
```

```
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

public class EmployeeManagement {
    private static List<Employee> employees = new ArrayList<>();

    public static void addEmployee(int id, String name, double salary) {
        employees.add(new Employee(id, name, salary));
    }

    public static void updateEmployee(int id, String newName, double newSalary) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                emp.name = newName;
                emp.salary = newSalary;
                return;
            }
        }
    }

    public static void removeEmployee(int id) {
        employees.removeIf(emp -> emp.id == id);
    }

    public static Employee searchEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.id == id) return emp;
        }
        return null;
    }
}
```

Problem 1.2: Card Collection Management

```
import java.util.*;

class CardCollection {
    private List<Card> cards = new ArrayList<>();

    class Card {
        String symbol;
        String cardName;

        Card(String symbol, String cardName) {
            this.symbol = symbol;
```

```
        this.cardName = cardName;
    }

    @Override
    public String toString() {
        return cardName;
    }
}

public void addCard(String symbol, String cardName) {
    cards.add(new Card(symbol, cardName));
}

public List<String> getCardsBySymbol(String symbol) {
    List<String> result = new ArrayList<>();
    for (Card card : cards) {
        if (card.symbol.equals(symbol)) {
            result.add(card.cardName);
        }
    }
    return result;
}

public static void main(String[] args) {
    CardCollection collection = new CardCollection();


    collection.addCard("Hearts", "Ace of Hearts");
    collection.addCard("Hearts", "2 of Hearts");
    collection.addCard("Diamonds", "Ace of Diamonds");

    System.out.println("Hearts: " + collection.getCardsBySymbol("Hearts"));
    System.out.println("Diamonds: " + collection.getCardsBySymbol("Diamonds"));
}
```

Problem 1.3: Ticket Booking System with Synchronized Threads

```
class TicketBookingSystem {
    private static int availableSeats = 10;
    private static Lock lock = new ReentrantLock();

    public void bookSeat(String customer) {
        lock.lock();
        try {
```

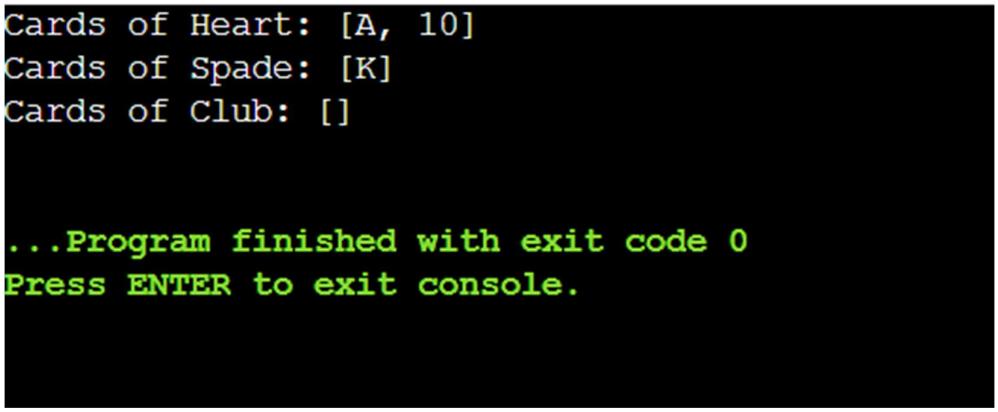
```
            if (availableSeats > 0) {
                System.out.println(customer + " booked a seat. Remaining seats: " +
(--availableSeats));
            } else {
                System.out.println("No seats available for " + customer);
            }
        } finally {
            lock.unlock();
        }
    }
}

public class TicketBooking {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        Thread vip = new Thread(() -> system.bookSeat("VIP Customer"));
        Thread normal = new Thread(() -> system.bookSeat("Normal
Customer"));
        vip.setPriority(Thread.MAX_PRIORITY);
        normal.setPriority(Thread.MIN_PRIORITY);
        vip.start();
        normal.start();
    }
}
```

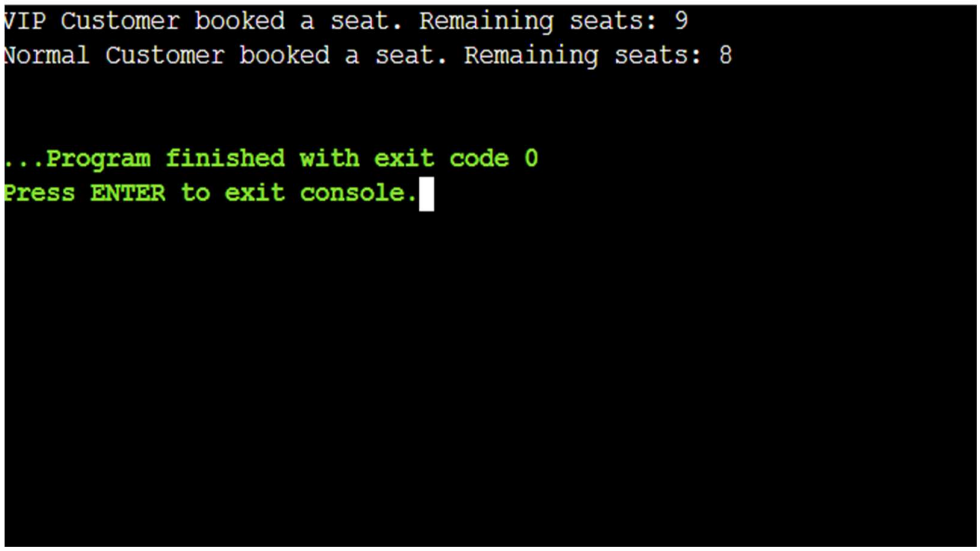
4. Output:

```
Choose an option: 1. Add Employee 2. Update Employee 3. Remove Employee 4. Search Employee 5. Exit
1
Enter ID: 15377
Enter Name: Jobanjeet
Enter Salary: 2000000
Choose an option: 1. Add Employee 2. Update Employee 3. Remove Employee 4. Search Employee 5. Exit
4
Enter ID: 15377
ID: 15377, Name: Jobanjeet, Salary: 2000000.0
Choose an option: 1. Add Employee 2. Update Employee 3. Remove Employee 4. Search Employee 5. Exit
```

(Fig. 1 - Employee Management Output)



(Fig. 2 - Card Collection Output)



(Fig. 3 - Ticket Booking System Output)

5. Learning Outcome:

1. Learn to manage collections dynamically using ArrayList and HashMap.
2. Understand thread synchronization and priority handling in Java.
3. Develop real-world applications with efficient data management techniques.