Experiment 2.1

Student Name: Nikhil Kumar **UID: 22BCS15501**
Branch: CSE **Section: 22KPIT-901/B**
Semester: 6th **Date of Performance:20/02/2025**
Subject: Project Based Learning in Java **Subject Code: 22CSH-359**

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

2. **Objective:**

   a) 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) 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) 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. **Code/Implementation:**

a)

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

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

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

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

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("1. Add Employee\n2. Update Employee\n3. Remove Employee\n4.
   Search Employee\n5. Exit");
        int choice = scanner.nextInt();
        if (choice == 5) break;
        switch (choice) {
            case 1:
                System.out.println("Enter ID, Name, Salary:");
                addEmployee(scanner.nextInt(), scanner.next(), scanner.nextDouble());
                break;
            case 2:
                System.out.println("Enter ID, New Name, New Salary:");
                updateEmployee(scanner.nextInt(), scanner.next(), scanner.nextDouble());
                break;
            case 3:
                System.out.println("Enter ID to Remove:");
                removeEmployee(scanner.nextInt());
                break;
            case 4:
                System.out.println("Enter ID to Search:");
                Employee emp = searchEmployee(scanner.nextInt());
                System.out.println(emp != null ? emp : "Employee not found");
```

```
                break;
            }
        }
        scanner.close();
    }
}
```

b)

```
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 Collection<Card> cards = new ArrayList<>();

    static void addCard(String symbol, String value) {
        cards.add(new Card(symbol, value));
    }

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("1. Add Card\n2. Find Cards by Symbol\n3. Exit");
            int choice = scanner.nextInt();
            if (choice == 3) break;
            switch (choice) {
                case 1:
                    System.out.println("Enter Symbol and Value:");
                    addCard(scanner.next(), scanner.next());
                    break;
```

```
                case 2:
                    System.out.println("Enter Symbol to Search:");
                    List<Card> foundCards = getCardsBySymbol(scanner.next());
                    System.out.println(foundCards.isEmpty() ? "No cards found" : foundCards);
                    break;
            }
        }
        scanner.close();
    }
}
```

c)

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

class TicketBookingSystem {
    private final boolean[] seats;
    private final Object lock = new Object();

    public TicketBookingSystem(int seatCount) {
        seats = new boolean[seatCount];
    }

    public boolean bookSeat(int seatNumber, String customer) {
        synchronized (lock) {
            if (seatNumber < 0 || seatNumber >= seats.length || seats[seatNumber]) {
                return false;
            }
            seats[seatNumber] = true;
            System.out.println(customer + " successfully booked seat " + seatNumber);
            return true;
        }
    }
}

class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final String customer;

    public BookingThread(TicketBookingSystem system, int seatNumber, String customer, int
      priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.customer = customer;
        setPriority(priority);
    }

    public void run() {
        if (!system.bookSeat(seatNumber, customer)) {
            System.out.println(customer + " failed to book seat " + seatNumber + ". Already taken.");
        }
```

```
    }
}

public class TicketBookingDemo {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(10);
        ExecutorService executor = Executors.newFixedThreadPool(5);

        executor.execute(new BookingThread(system, 2, "VIP Customer 1",
       Thread.MAX_PRIORITY));
        executor.execute(new BookingThread(system, 2, "Regular Customer 1",
       Thread.MIN_PRIORITY));
        executor.execute(new BookingThread(system, 5, "VIP Customer 2",
       Thread.MAX_PRIORITY));
        executor.execute(new BookingThread(system, 5, "Regular Customer 2",
       Thread.MIN_PRIORITY));
        executor.execute(new BookingThread(system, 7, "Regular Customer 3",
       Thread.NORM_PRIORITY));

        executor.shutdown();
    }
}
```

4. Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERM

PS C:\Users\Dell\OneDrive\Desktop\Pr
loyeeManager.java } ; if ($?) { java
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
1
Enter ID, Name, Salary:
101 Nikhil 1500000
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
1
```

```
Enter ID, Name, Salary:
102 Gaurav 150000
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
4
Enter ID to Search:
101
ID: 101, Name: Nikhil, Salary: 1500000.0
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
```

(a)

```
Enter Symbol and Value:
Spade A
1. Add Card
2. Find Cards by Symbol
3. Exit
1
Enter Symbol and Value:
Club King
1. Add Card
2. Find Cards by Symbol
3. Exit
2
Enter Symbol to Search:
Spade
[A of Spade]
1. Add Card
2. Find Cards by Symbol
3. Exit
```

(b)

```
PS C:\Users\Dell\OneDrive\Desktop\Project Based Learning in Java with Lab\Lab> cd
ketBookingDemo.java } ; if ($?) { java TicketBookingDemo }
VIP Customer 2 successfully booked seat 5
Regular Customer 1 successfully booked seat 2
Regular Customer 3 successfully booked seat 7
Regular Customer 2 failed to book seat 5. Already taken.
VIP Customer 1 failed to book seat 2. Already taken.
PS C:\Users\Dell\OneDrive\Desktop\Project Based Learning in Java with Lab\Lab>
```

(c)

5. Learning Outcomes:

- Develop proficiency in Java programming using fundamental concepts such as **data structures, collections, and multithreading**.
- Understand the significance of **thread synchronization, collections, and data structures** in real-world scenarios.
- Implement a program to **store and retrieve cards** using the **Collection interface**.
- Develop an efficient mechanism to find all cards associated with a given symbol.
- Enhance understanding of data organization and retrieval techniques in Java.