Experiment 4

Name: Tarun Kumar **UID:22BCS15293**
Branch: BE-CSE **Section/Group:22BCS618-B**
Semester: 6th **DateofPerformance:20/02/2025**
Subject Name: Project Based Learning in Java with Lab **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 : Develop Java programs using ArrayList, Collection Interface, and Thread Synchronization to efficiently manage employee records, store and search card details, and implement a synchronized ticket booking system with prioritized VIP bookings.

3. Implementation/Code:

3.1 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.

```
import java.util.ArrayList;
 employee details
class Employee {
    private int id;
    private String name;
    private double salary;
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
```

```
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
class EmployeeManagementSystem {
    private ArrayList<Employee> employees;
    public EmployeeManagementSystem() {
        employees = new ArrayList<>();
    }
    public void addEmployee(int id, String name, double salary) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                System.out.println("Error: Employee with ID " + id + " already exists.");
                return;
            }
        }
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee Added: " + employees.get(employees.size() - 1));
    }
    public void updateEmployee(int id, double newSalary) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                emp.setSalary(newSalary);
                System.out.println("Employee ID " + id + " updated successfully.");
                return;
            }
        }
        System.out.println("Employee with ID " + id + " not found.");
    }
    public void removeEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                employees.remove(emp);
                System.out.println("Employee ID " + id + " removed successfully.");
                return;
            }
        }
        System.out.println("Employee with ID " + id + " not found.");
    }
    public void searchEmployeeById(int id) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                System.out.println("Employee Found: " + emp);
```

```
                return;
            }
        }
        System.out.println("Employee with ID " + id + " not found.");
    }
    public void displayAllEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            for (Employee emp : employees) {
                System.out.println(emp);
            }}}}
public class Main {
    public static void main(String[] args) {
        EmployeeManagementSystem ems = new EmployeeManagementSystem();
        ems.displayAllEmployees(); // Test Case 1
        ems.addEmployee(101, "Anish", 50000); // Test Case 2
        ems.addEmployee(102, "Bobby", 60000);
        ems.updateEmployee(101, 55000); // Test Case 3
        ems.searchEmployeeById(102); // Test Case 4
        ems.removeEmployee(101); // Test Case 5
        ems.displayAllEmployees(); // Test Case 6
        ems.addEmployee(101, "Charlie", 70000);}} // Test Case 7
```

Output:

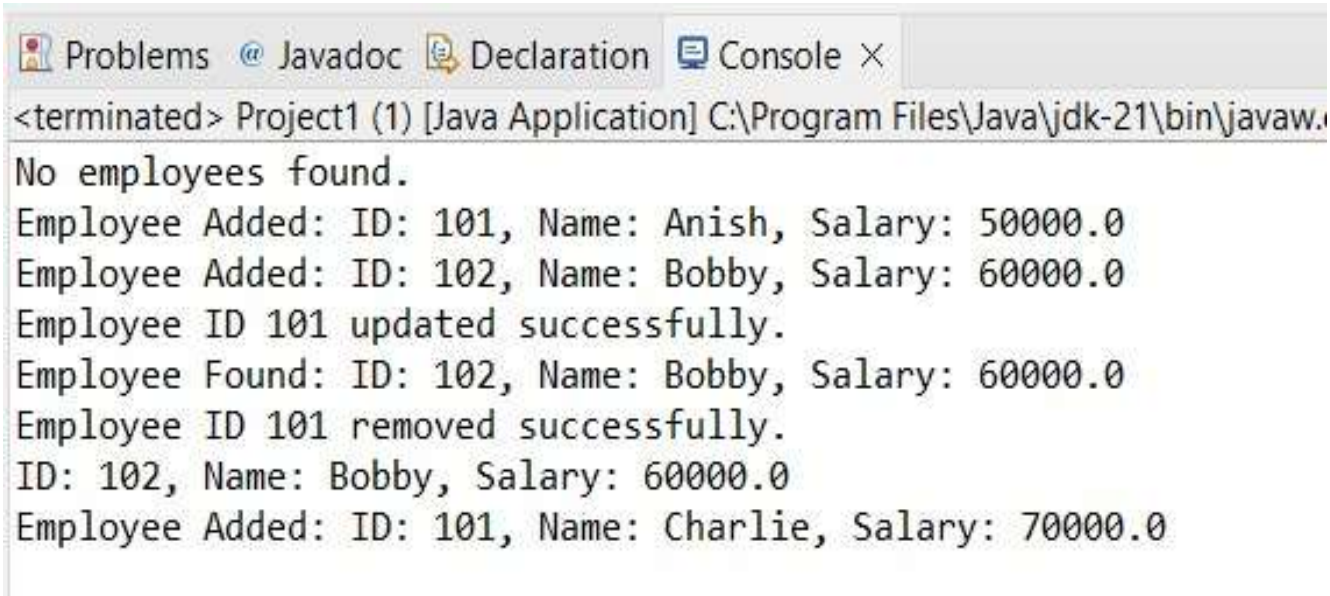

Fig. 1 (Output 3.1)

3.2 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.

```
import java.util.*;
class Card {
    private String symbol;
    private String value;
    public Card(String symbol, String value) {
        this.symbol = symbol;
        this.value = value;
    }
    public String getSymbol() {
        return symbol;
    }
    public String getValue() {
        return value;
    }
    @Override
    public String toString() {
        return value + " of " + symbol;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Card card = (Card) obj;
        return Objects.equals(symbol, card.symbol) && Objects.equals(value, card.value);
    }
    @Override
    public int hashCode() {
        return Objects.hash(symbol, value);
    }
}
class CardCollection {
    private Map<String, List<Card>> cardMap;
    private Set<Card> cardSet;
    public CardCollection() {
        cardMap = new HashMap<>();
        cardSet = new HashSet<>();
    }
    public void addCard(String symbol, String value) {
        Card newCard = new Card(symbol, value);
        if (cardSet.contains(newCard)) {
            System.out.println("Error: Card \"" + newCard + "\" already exists.");
            return;
        }
        cardSet.add(newCard);
```

```
            cardMap.computeIfAbsent(symbol, k -> new ArrayList<>()).add(newCard);
            System.out.println("Card added: " + newCard);
        }
        public void findCardsBySuit(String symbol) {
            if (!cardMap.containsKey(symbol) || cardMap.get(symbol).isEmpty()) {
                System.out.println("No cards found for " + symbol + ".");
                return;
            }
            for (Card card : cardMap.get(symbol)) {
                System.out.println(card);
            }
        }
        public void displayAllCards() {
            if (cardSet.isEmpty()) {
                System.out.println("No cards found.");
                return;
            }
            for (Card card : cardSet) {
                System.out.println(card);
            }
        }
        public void removeCard(String symbol, String value) {
            Card card = new Card(symbol, value);
            if (!cardSet.contains(card)) {
                System.out.println("Card not found: " + card);
                return;
            }
            cardSet.remove(card);
            cardMap.get(symbol).remove(card);
            if (cardMap.get(symbol).isEmpty()) {
                cardMap.remove(symbol);
            }
            System.out.println("Card removed: " + card);
        }
    }
    public class Main {
        public static void main(String[] args) {
            CardCollection collection = new CardCollection();
            // Test Case 1: No Cards Initially
            collection.displayAllCards();
            // Test Case 2: Adding Cards
            collection.addCard("Spades", "Ace");
            collection.addCard("Hearts", "King");
            collection.addCard("Diamonds", "10");
            collection.addCard("Clubs", "5");
```

```
        // Test Case 3: Finding Cards by Suit
        collection.findCardsBySuit("Hearts");

        // Test Case 4: Searching Suit with No Cards
        collection.findCardsBySuit("Diamonds");

        // Test Case 5: Displaying All Cards
        collection.displayAllCards();

        // Test Case 6: Preventing Duplicate Cards
        collection.addCard("Hearts", "King");

        // Test Case 7: Removing a Card
        collection.removeCard("Diamonds", "10");
    }
}
```

Output:

Fig. 2 (Output 3.2)

3.3 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.

```
import java.util.Arrays;
class TicketBookingSystem {
    private final boolean[] seats;
    public TicketBookingSystem(int numSeats) {
        seats = new boolean[numSeats];
        Arrays.fill(seats, false);
    }
    public synchronized void bookSeat(String user, int seatNumber, boolean isVIP) {
        if (seatNumber < 1 || seatNumber > seats.length) {
            System.out.println("Invalid seat number!");
            return;
        }
        if (seats[seatNumber - 1]) {
            System.out.println(user + ": Seat " + seatNumber + " is already booked!");
            return;
        }
        seats[seatNumber - 1] = true;
        System.out.println(user + " booked seat " + seatNumber);
    }
}
class UserThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final boolean isVIP;
    public UserThread(TicketBookingSystem system, String name, int seatNumber, boolean isVIP) {
        super(name);
        this.system = system;
        this.seatNumber = seatNumber;
        this.isVIP = isVIP;
        if (isVIP) setPriority(MAX_PRIORITY);
        else setPriority(NORM_PRIORITY);
    }
    public void run() {
        system.bookSeat(getName(), seatNumber, isVIP);
    }
}
public class Main {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5);
```

```
// Test Case 1: No Seats Available Initially
// Expected Output: No bookings yet. (No users attempting)
// Test Case 2: Successful Booking
new UserThread(system, "Anish (VIP)", 1, true).start();
new UserThread(system, "Bobby (Regular)", 2, false).start();
new UserThread(system, "Charlie (VIP)", 3, true).start();
// Test Case 3: Thread Priorities (VIP First)
new UserThread(system, "Bobby (Regular)", 4, false).start();
new UserThread(system, "Anish (VIP)", 4, true).start();
// Test Case 4: Preventing Double Booking
new UserThread(system, "Anish (VIP)", 1, true).start();
new UserThread(system, "Bobby (Regular)", 1, false).start();
// Test Case 5: Booking After All Seats Are Taken
new UserThread(system, "New User (Regular)", 3, false).start();
// Test Case 6: Invalid Seat Selection
new UserThread(system, "User1", 0, false).start();
new UserThread(system, "User2", 6, false).start();
// Test Case 7: Simultaneous Bookings (Concurrency Test)
for (int i = 0; i < 10; i++) {
    int seat = (i % 5) + 1;
    new UserThread(system, "User" + (i + 1), seat, i % 2 == 0).start();}}}
```

Output:

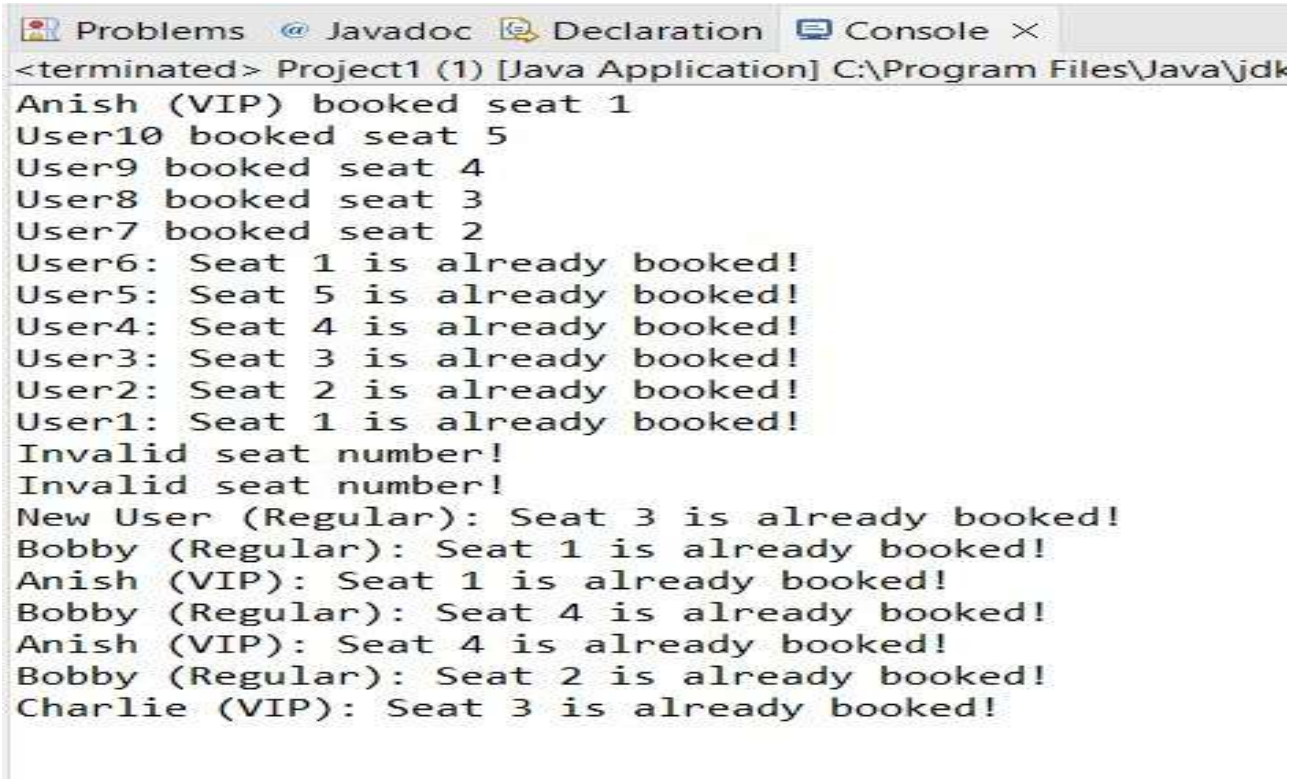

Fig. 3 (Output 3.3)

4. Learning Outcomes:

- Understand ArrayList operations for storing and managing structured data dynamically.
- Gain hands-on experience with Collection interfaces for efficient data storage and retrieval.
- Learn thread synchronization to prevent data inconsistency in concurrent environments.
- Implement thread priorities to manage task execution order in multi-threaded applications.
- Develop real-world problem-solving skills by handling data structures, collections, and concurrency in Java