Experiment 4

Student Name: Swati Joshi **UID: 22BCS10183**
Branch: BE-CSE **Section/Group: 22BCS 618 - A**
Semester: 6th **Date of Performance:14/02/2025**
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;
import java.util.Scanner;
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 setName(String name) { this.name = name; }
    public void setSalary(double salary) { this.salary = salary; }
    @Override
    public String toString() {
        return "Employee [ID=" + id + ", Name=" + name + ", Salary=" + salary + "]";
    }
}
public class Main {
    private static ArrayList<Employee> employees = new ArrayList<>();
```

```
    private static Scanner scanner = new Scanner(System.in);
    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit");
            switch (scanner.nextInt()) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> displayAllEmployees();
                case 6 -> { System.out.println("Exiting..."); return; }
                default -> System.out.println("Invalid choice.");
            }
        }
    }
    private static void addEmployee() {
        System.out.print("ID: "); int id = scanner.nextInt();
        System.out.print("Name: "); String name = scanner.next();
        System.out.print("Salary: "); double salary = scanner.nextDouble();
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added.");
    }
    private static void updateEmployee() {
        System.out.print("ID to update: "); int id = scanner.nextInt();
        for (Employee e : employees) {
            if (e.getId() == id) {
                System.out.print("New Name: "); e.setName(scanner.next());
                System.out.print("New Salary: "); e.setSalary(scanner.nextDouble());
                System.out.println("Employee updated."); return;
            }
        }
        System.out.println("Employee not found.");
    }
    private static void removeEmployee() {
        System.out.print("ID to remove: "); int id = scanner.nextInt();
        employees.removeIf(e -> e.getId() == id);
        System.out.println("Employee removed.");
    }
    private static void searchEmployee() {
        System.out.print("ID to search: "); int id = scanner.nextInt();
        employees.stream().filter(e -> e.getId() == id).forEach(System.out::println);
    }
```

```
 private static void displayAllEmployees() {
        if (employees.isEmpty()) System.out.println("No employees found.");
        else employees.forEach(System.out::println);
    }
}
```

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.*;
interface CardOperations {
    void addCard();
    void searchCardsBySymbol();
    void displayAllCards();
}
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;
    }
    @Override
    public String toString() {
        return "Card{Symbol='" + symbol + "', Value='" + value + "'}";
    }
}
class CardCollection implements CardOperations {
    private Collection<Card> cards = new ArrayList<>();
    private Scanner scanner = new Scanner(System.in);
    @Override
    public void addCard() {
        System.out.print("Enter Card Symbol: ");
        String symbol = scanner.next();
        System.out.print("Enter Card Value: ");
        String value = scanner.next();
        cards.add(new Card(symbol, value));
        System.out.println("Card added successfully!");
    }
    @Override
    public void searchCardsBySymbol() {
```

```
        System.out.print("Enter symbol to search: ");
        String symbol = scanner.next();
        cards.stream()
                .filter(card -> card.getSymbol().equalsIgnoreCase(symbol))
                .forEach(System.out::println);
    }
    @Override
    public void displayAllCards() {
        if (cards.isEmpty()) {
            System.out.println("No cards in the collection.");
        } else {
            cards.forEach(System.out::println);
        }
    }
    public static void main(String[] args) {
        CardOperations cardCollection = new CardCollection();
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\n1. Add Card");
            System.out.println("2. Search Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Enter choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1: cardCollection.addCard(); break;
                case 2: cardCollection.searchCardsBySymbol(); break;
                case 3: cardCollection.displayAllCards(); break;
                case 4: System.exit(0);
                default: System.out.println("Invalid choice!");
            }
        }
    }
}
```

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.*;
class TicketBookingSystem {
    private int availableSeats;
    public TicketBookingSystem(int seats) {
        this.availableSeats = seats;    }
```

```
    public synchronized boolean bookTicket(String name) {
        if (availableSeats > 0) {
            System.out.println(name + " successfully booked a ticket. Seats left: " + (--
availableSeats));
            return true;
        } else {
            System.out.println(name + " failed to book a ticket. No seats available.");
            return false;
        }
    }
}
class BookingThread extends Thread {
    private TicketBookingSystem system;
    private String userName;
    public BookingThread(TicketBookingSystem system, String userName, int priority) {
        this.system = system;
        this.userName = userName;
        setPriority(priority); // Set thread priority
    }
    @Override
    public void run() {
        system.bookTicket(userName);
    }
}
public class TicketBookingMain {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5); // 5 seats available
        List<BookingThread> threads = new ArrayList<>();
        threads.add(new BookingThread(system, "VIP_Hardik", Thread.MAX_PRIORITY));  //
VIP Booking (High Priority)
        threads.add(new BookingThread(system, "User_Rahul", Thread.NORM_PRIORITY));
        threads.add(new BookingThread(system, "VIP_Virat", Thread.MAX_PRIORITY)); // VIP
Booking (High Priority)
        threads.add(new BookingThread(system, "User_Rohit", Thread.NORM_PRIORITY));
        threads.add(new BookingThread(system, "User_Shubman", Thread.MIN_PRIORITY)); //
Low priority
        // Start all threads
        for (BookingThread thread : threads) {
            thread.start();
        }
        // Wait for all threads to finish
        for (BookingThread thread : threads) {
```

```
            try {
                thread.join(); }
            catch (InterruptedException e) {
                e.printStackTrace();  }
        }
        System.out.println("All bookings completed!");
    }
}
```

4. Output:

4.1

```
1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
1
ID: 101
Name: ABC
Salary: 1234
Employee added.

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
5
Employee [ID=101, Name=ABC, Salary=1234.0]
```

4.2

```
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 1
Enter Card Symbol:  ♥
Enter Card Value: A
Card added successfully!

1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 3
Card{Symbol='♥', Value='A'}
```

4.3

```
VIP_Hardik successfully booked a ticket. Seats left: 4
User_Shubman successfully booked a ticket. Seats left: 3
User_Rohit successfully booked a ticket. Seats left: 2
User_Rahul successfully booked a ticket. Seats left: 1
VIP_Virat successfully booked a ticket. Seats left: 0
All bookings completed!

Process finished with exit code 0
```

5. 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