EXPERIMENT- 4

Student Name: Shraddha Shinde **UID:** 22BCS16617

Branch: CSE **Section/Group:** 22BCS_IOT-618-B

Semester: 6 **Date of Performance:** 21.02.25

Subject Name: Project Based Learning in Java **Subject Code:**22CSH-359

1. **Aim**: Collection Framework and MultiThreading

2. **Objective:**

   Use of Collections in Java. Array List, Linked List, Hash Map, Tree Map, Hash Set in Java. Multithreading in Java. Thread Synchronization. Thread Priority, Thread Lifecycle.

Problem 1

3. **Problem:** 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.

4. **Implementation/Code:**

New Project | Online Java Compiler IDE

```
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;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

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

    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();

        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println("Error: Employee with ID " + id + " already exists.");
                return;
            }
        }
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee Added: ID=" + id + ", Name=" + name + ", Salary=" + salary);
    }
```

```
    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        System.out.print("Enter new Salary: ");
        double newSalary = scanner.nextDouble();

        for (Employee emp : employees) {
            if (emp.id == id) {
                emp.salary = newSalary;
                System.out.println("Employee ID " + id + " updated successfully.");
                return;
            }
        }
        System.out.println("Error: Employee ID " + id + " not found.");
    }

    public static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();

        Iterator<Employee> iterator = employees.iterator();
        while (iterator.hasNext()) {
            if (iterator.next().id == id) {
                iterator.remove();
                System.out.println("Employee ID " + id + " removed successfully.");
                return;
            }
        }
        System.out.println("Error: Employee ID " + id + " not found.");
    }
```

```
    public static void searchEmployeeById() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();

        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println("Employee Found: " + emp);
                return;
            }
        }
        System.out.println("Error: Employee ID " + id + " not found.");
    }

    public static void displayAllEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }
```

```
    public static void main(String[] args) {
        while (true) {
            System.out.println("\nEmployee Management System");
            System.out.println("1. Display Employees");
            System.out.println("2. Add Employee");
            System.out.println("3. Update Employee Salary");
            System.out.println("4. Search Employee by ID");
            System.out.println("5. Remove Employee");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    displayAllEmployees();
                    break;
                case 2:
                    addEmployee();
                    break;
                case 3:
                    updateEmployee();
                    break;
                case 4:
                    searchEmployeeById();
                    break;
                case 5:
                    removeEmployee();
                    break;
                case 6:
                    System.out.println("Exiting Employee Management System...");
                    return;
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
    }
}
```

5. **Output:**

```
Employee Management System
1. Display Employees
2. Add Employee
3. Update Employee Salary
4. Search Employee by ID
5. Remove Employee
6. Exit
Enter your choice: 1
No employees found.

Employee Management System
1. Display Employees
2. Add Employee
3. Update Employee Salary
4. Search Employee by ID
5. Remove Employee
6. Exit
Enter your choice: 2
Enter Employee ID: 1
Enter Employee Name: shraddha
Enter Employee Salary: 400000
Employee Added: ID=1, Name=shraddha, Salary=400000.0

Employee Management System
1. Display Employees
2. Add Employee
3. Update Employee Salary
4. Search Employee by ID
5. Remove Employee
6. Exit
Enter your choice: 1
ID: 1, Name: shraddha, Salary: 400000.0
```

Problem 2

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

7. **Implementation/Code:**

New Project ⋮ Online Java Compiler IDE

```java
import java.util.*;

class Card {
    private String rank;
    private String suit;

    public Card(String rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }

    public String getRank() {
        return rank;
    }

    public String getSuit() {
        return suit;
    }

    @Override
    public String toString() {
        return rank + " of " + suit;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Card card = (Card) obj;
        return rank.equals(card.rank) && suit.equals(card.suit);
    }

    @Override
    public int hashCode() {
        return Objects.hash(rank, suit);
    }
}
```

```
class CardCollection {
    private Map<String, Set<Card>> cardsBySuit;

    public CardCollection() {
        cardsBySuit = new HashMap<>();
    }

    // Add a card to the collection
    public String addCard(String rank, String suit) {
        cardsBySuit.putIfAbsent(suit, new HashSet<>());

        Card newCard = new Card(rank, suit);
        if (cardsBySuit.get(suit).contains(newCard)) {
            return "Error: Card \"" + newCard + "\" already exists.";
        }

        cardsBySuit.get(suit).add(newCard);
        return "Card added: " + newCard;
    }

    // Find all cards of a given suit
    public String findCardsBySuit(String suit) {
        if (!cardsBySuit.containsKey(suit) || cardsBySuit.get(suit).isEmpty()) {
            return "No cards found for " + suit + ".";
        }
        StringBuilder result = new StringBuilder();
        for (Card card : cardsBySuit.get(suit)) {
            result.append(card).append("\n");
        }
        return result.toString().trim();
    }

    // Display all stored cards
    public String displayAllCards() {
        if (cardsBySuit.isEmpty()) {
            return "No cards found.";
        }
        StringBuilder result = new StringBuilder();
        for (Set<Card> cardSet : cardsBySuit.values()) {
            for (Card card : cardSet) {
                result.append(card).append("\n");
            }
        }
        return result.toString().trim();
    }

    // Remove a card from the collection
    public String removeCard(String rank, String suit) {
        if (!cardsBySuit.containsKey(suit) || !cardsBySuit.get(suit).remove(new Card(rank, suit))) {
            return "Card \"" + rank + " of " + suit + "\" not found.";
        }

        // Remove the suit entry if it's empty after removal
        if (cardsBySuit.get(suit).isEmpty()) {
            cardsBySuit.remove(suit);
        }
        return "Card removed: " + rank + " of " + suit;
    }
}

// Main class with Switch Case Menu
public class CardCollectionSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CardCollection collection = new CardCollection();
        int choice;

        do {
            System.out.println("\n===== Card Collection System =====");
            System.out.println("1. Add Card");
            System.out.println("2. Find Cards by Suit");
            System.out.println("3. Display All Cards");
            System.out.println("4. Remove Card");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1: // Add Card
                    System.out.print("Enter card rank (e.g., Ace, King, 10): ");
                    String rank = scanner.nextLine();
                    System.out.print("Enter card suit (e.g., Spades, Hearts): ");
                    String suit = scanner.nextLine();
                    System.out.println(collection.addCard(rank, suit));
                    break;

                case 2: // Find Cards by Suit
                    System.out.print("Enter suit to find (e.g., Hearts): ");
                    String findSuit = scanner.nextLine();
                    System.out.println(collection.findCardsBySuit(findSuit));
                    break;

                case 3: // Display All Cards
                    System.out.println(collection.displayAllCards());
                    break;
```

```
                case 4: // Remove Card
                    System.out.print("Enter rank of card to remove: ");
                    String removeRank = scanner.nextLine();
                    System.out.print("Enter suit of card to remove: ");
                    String removeSuit = scanner.nextLine();
                    System.out.println(collection.removeCard(removeRank, removeSuit));
                    break;

                case 5: // Exit
                    System.out.println("Exiting the Card Collection System. Goodbye!");
                    break;

                default:
                    System.out.println("Invalid choice! Please enter a valid option.");
            }
        } while (choice != 5);

        scanner.close();
    }
}
```

8. Output:

Output Generated files

```
===== Card Collection System =====
1. Add Card
2. Find Cards by Suit
3. Display All Cards
4. Remove Card
5. Exit
Enter your choice: 1
Enter card rank (e.g., Ace, King, 10): 10
Enter card suit (e.g., Spades, Hearts): hearts
Card added: 10 of hearts

===== Card Collection System =====
1. Add Card
2. Find Cards by Suit
3. Display All Cards
4. Remove Card
5. Exit
Enter your choice: 2
Enter suit to find (e.g., Hearts): 2
No cards found for 2.
```

Problem 3

9. **Problem:** 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.

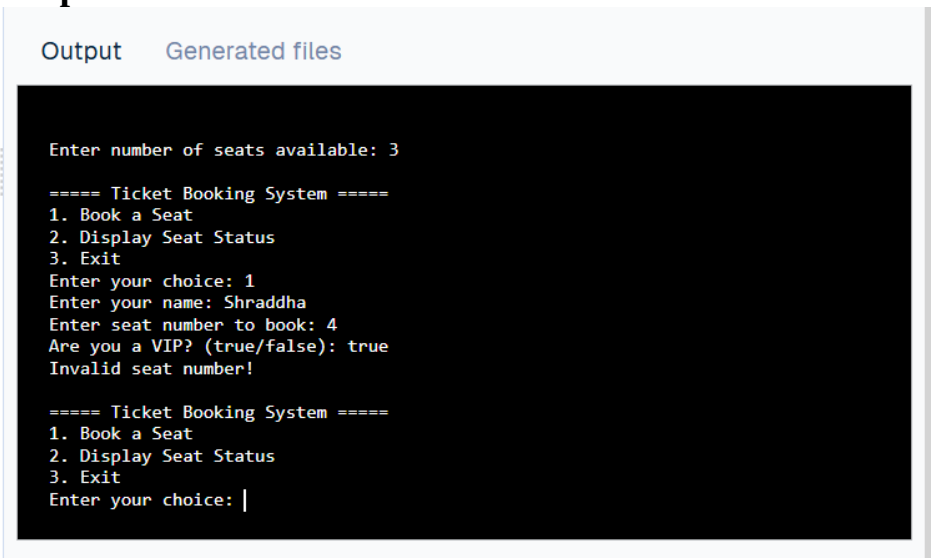
10. **Implementation/Code:**

New Project ⋮ Online Java Compiler IDE

```
import java.util.Scanner;

class TicketBookingSystem {
    private boolean[] seats;

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

    // Synchronized method to prevent double booking
    public synchronized String bookSeat(int seatNumber, String user, boolean isVIP) {
        if (seatNumber < 1 || seatNumber > seats.length) {
            return "Invalid seat number!";
        }
        if (seats[seatNumber - 1]) {
            return user + ": Seat " + seatNumber + " is already booked!";
        }

        seats[seatNumber - 1] = true;
        return user + " (" + (isVIP ? "VIP" : "Regular") + ") booked seat " + seatNumber;
    }

    // Display available seats
    public synchronized void displaySeats() {
        System.out.println("\nCurrent Seat Status:");
        for (int i = 0; i < seats.length; i++) {
            System.out.println("Seat " + (i + 1) + ": " + (seats[i] ? "Booked" : "Available"));
        }
    }
}

// Thread class for users booking seats
class BookingThread extends Thread {
    private TicketBookingSystem system;
    private int seatNumber;
    private String user;
    private boolean isVIP;

    public BookingThread(TicketBookingSystem system, int seatNumber, String user, boolean isVIP) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.user = user;
        this.isVIP = isVIP;
        setPriority(isVIP ? Thread.MAX_PRIORITY : Thread.NORM_PRIORITY); // VIP gets high priority
    }

    @Override
    public void run() {
        System.out.println(system.bookSeat(seatNumber, user, isVIP));
    }
}

// Main class with switch-case menu
public class TicketBookingApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter number of seats available: ");
        int totalSeats = scanner.nextInt();

        TicketBookingSystem system = new TicketBookingSystem(totalSeats);

        while (true) {
            System.out.println("\n===== Ticket Booking System =====");
            System.out.println("1. Book a Seat");
            System.out.println("2. Display Seat Status");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
```

```
        switch (choice) {
            case 1: // Book a seat
                System.out.print("Enter your name: ");
                String user = scanner.next();
                System.out.print("Enter seat number to book: ");
                int seatNumber = scanner.nextInt();
                System.out.print("Are you a VIP? (true/false): ");
                boolean isVIP = scanner.nextBoolean();

                BookingThread booking = new BookingThread(system, seatNumber, user, isVIP);
                booking.start();

                try {
                    booking.join(); // Ensure booking completes before moving on
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                break;

            case 2: // Display seats
                system.displaySeats();
                break;

            case 3: // Exit
                System.out.println("Exiting Ticket Booking System. Goodbye!");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice! Please select a valid option.");
        }
```

11.Output:



12.Learning Outcomes:

(i) Implement abstract class and method overriding, Use an abstract class Account and override in subclasses.

(ii) Apply conditional logic for interest Handle different problems on adding the data and arranging it.