DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment 4

Name: ANUKRITI
Uid: 22BET10023
Section/Group: EPAM_802/B
Subject Name: Project Based Learning In Java Lab
Branch: BE-CSEA
Semester: 6TH
Date Of Performance: 17/2/2025
Subject Code: 22CSP-359

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

- **Objectives**
  - **Ticket Booking System (Multithreading)**
    1. Implement a multi-user ticket booking system using Java threads.
    2. Use thread synchronization to prevent race conditions when booking seats.
    3. Allow users to specify priority levels for booking requests.
    4. Demonstrate concurrent execution and thread scheduling in Java.
  - **Employee Management System**
    1. Provide CRUD operations (Create, Read, Update, Delete) for employee records.
    2. Use an ArrayList to store and manage employee objects.
    3. Implement a search function to retrieve employee details by ID.
    4. Allow user interaction through a menu-driven console interface.
  - **Card Collection System**
    1. Enable users to add cards with a symbol and value.
    2. Store and manage card objects using a collection framework.
    3. Provide a search function to find cards based on their symbol.
    4. Implement a menu-driven system for adding, displaying, and searching cards.

- **Code:**

1.

```
import java.util.ArrayList;
import java.util.Scanner;
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 EmployeeManagement {
    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("\nEmployee Management System");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
            System.out.println("4. Search Employee");
            System.out.println("5. Display All Employees");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1: addEmployee(); break;
                case 2: updateEmployee(); break;
                case 3: removeEmployee(); break;
                case 4: searchEmployee(); break;
                case 5: displayEmployees(); break;
                case 6: System.exit(0);
                default: System.out.println("Invalid option. Try again.");
            }
        }
    }
    private static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully.");
    }
    private static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                scanner.nextLine();
                System.out.print("Enter new Name: ");
                emp.name = scanner.nextLine();
```

```
                System.out.print("Enter new Salary: ");
                emp.salary = scanner.nextDouble();
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }
    private static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        employees.removeIf(emp -> emp.id == id);
        System.out.println("Employee removed if found.");
    }
    private static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found.");
    }
    private static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees to display.");
        } else {
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }
}
```

2.

```
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;
    }
```

```
    @Override
    public String toString() {
        return "Card{" + "symbol='" + symbol + '\'' + ", value='" + value + '\'' + '}';
    }
}
public class CardCollection {
    private static Collection<Card> cards = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);
    public static void main(String[] args) {
        while (true) {
            System.out.println("\nCard Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Display All Cards");
            System.out.println("3. Search Cards by Symbol");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
            switch (choice) {
                case 1:
                    addCard();
                    break;
                case 2:
                    displayCards();
                    break;
                case 3:
                    searchBySymbol();
                    break;
                case 4:
                    System.exit(0);
                default:
                    System.out.println("Invalid option. Try again.");
            }
        }
    }
    private static void addCard() {
        System.out.print("Enter Card Symbol: ");
        String symbol = scanner.nextLine();
        System.out.print("Enter Card Value: ");
        String value = scanner.nextLine();
        cards.add(new Card(symbol, value));
        System.out.println("Card added successfully.");
    }
    private static void displayCards() {
        if (cards.isEmpty()) {
            System.out.println("No cards to display.");
        } else {
            for (Card card : cards) {
```

```
                System.out.println(card);
            }
        }
    }
    private static void searchBySymbol() {
        System.out.print("Enter Symbol to search: ");
        String symbol = scanner.nextLine();
        boolean found = false;

        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                System.out.println(card);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No cards found with the given symbol.");
        }
    }
}
```

3.

```
import java.util.*;
class TicketBookingSystem {
    private final int totalSeats;
    private final boolean[] seats;
    public TicketBookingSystem(int totalSeats) {
        this.totalSeats = totalSeats;
        this.seats = new boolean[totalSeats];
    }
    public synchronized boolean bookSeat(int seatNumber, String customerName) {
        if (seatNumber < 0 || seatNumber >= totalSeats) {
            System.out.println("Invalid seat number: " + seatNumber);
            return false;
        }
        if (!seats[seatNumber]) {
            seats[seatNumber] = true;
            System.out.println("Seat " + seatNumber + " booked successfully by " +
customerName);
            return true;
        } else {
            System.out.println("Seat " + seatNumber + " is already booked.");
            return false;
        }
    }
}
class BookingThread extends Thread {
    private final TicketBookingSystem system;
```

```
    private final int seatNumber;
    private final String customerName;
    public BookingThread(TicketBookingSystem system, int seatNumber, String
customerName, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.customerName = customerName;
        setPriority(priority);
    }
    @Override
    public void run() {
        system.bookSeat(seatNumber, customerName);
    }
}
public class TicketBookingApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        TicketBookingSystem system = new TicketBookingSystem(10);
        System.out.print("Enter number of users: ");
        int numUsers = scanner.nextInt();
        scanner.nextLine();
        Thread[] threads = new Thread[numUsers];
        for (int i = 0; i < numUsers; i++) {
            System.out.print("Enter name for user " + (i + 1) + ": ");
            String name = scanner.nextLine();
            System.out.print("Enter seat number for " + name + ": ");
            int seatNumber = scanner.nextInt();
            System.out.print("Enter priority (1-10, where 10 is highest) for " + name + ": ");
            int priority = scanner.nextInt();
            scanner.nextLine();
            threads[i] = new BookingThread(system, seatNumber, name, priority);
        }
        for (Thread thread : threads) {
            thread.start();
        }
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        scanner.close();
    }
}
```

- **Output**
  1. **Ticket Booking System:**

```
akanshasonker@Akanshas-MacBook-Pro exp_4 % cd "/Users/akanshasonker/Desktop/untitled folder/exp_4/" && javac EmployeeManagement.java
 && java EmployeeManagement

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 101
Enter Name: Akansha
Enter Salary: 2000000
Employee added successfully.

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 102
Enter Name: Alice
Enter Salary: 20000
Employee added successfully.

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 5
ID: 101, Name: Akansha, Salary: 2000000.0
ID: 102, Name: Alice, Salary: 20000.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 3
Enter Employee ID to remove: 102
Employee removed if found.

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 5
ID: 101, Name: Akansha, Salary: 2000000.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option:
```

2. Attempting to Book an Already Taken Seat

```
akanshasonker@Akanshas-MacBook-Pro exp_4 % cd "/Users/akanshasonker/Desktop/untitled folder/exp_4/" && javac CardCollection.java &&
java CardCollection

Card Collection System
1. Add Card
2. Display All Cards
3. Search Cards by Symbol
4. Exit
Choose an option: 1
Enter Card Symbol: Heart
Enter Card Value: Ace
Card added successfully.

Card Collection System
1. Add Card
2. Display All Cards
3. Search Cards by Symbol
4. Exit
Choose an option: 1
Enter Card Symbol: Spade
Enter Card Value: King
Card added successfully.

Card Collection System
1. Add Card
2. Display All Cards
3. Search Cards by Symbol
4. Exit
Choose an option: 2
Card{symbol='Heart', value='Ace'}
Card{symbol='Spade', value='King'}

Card Collection System
1. Add Card
2. Display All Cards
3. Search Cards by Symbol
4. Exit
Choose an option: 3
Enter Symbol to search: Diamond
No cards found with the given symbol.

Card Collection System
1. Add Card
2. Display All Cards
3. Search Cards by Symbol
4. Exit
Choose an option: 4
akanshasonker@Akanshas-MacBook-Pro exp_4 %
```

3. Invalid Seat Number

```
akanshasonker@Akanshas-MacBook-Pro exp_4 % cd "/Users/akanshasonker/Desktop/untitled folder/exp_4/" && javac TicketBookingApp.java &
& java TicketBookingApp
Enter number of users: 3
Enter name for user 1: Akansha
Enter seat number for Akansha: 3
Enter priority (1-10, where 10 is highest) for Akansha: 8
Enter name for user 2: Alex
Enter seat number for Alex: 7
Enter priority (1-10, where 10 is highest) for Alex: 5
Enter name for user 3: Pox
Enter seat number for Pox: 3
Enter priority (1-10, where 10 is highest) for Pox: 3
Seat 3 booked successfully by Akansha
Seat 7 booked successfully by Alex
Seat 3 is already booked.
akanshasonker@Akanshas-MacBook-Pro exp_4 %
```

- **Learning Outcomes:**
  - **Ticket Booking System (Multithreading)**
    1. Understand the fundamentals of multithreading and concurrent execution in Java.
    2. Learn how to use the synchronized keyword to prevent data inconsistencies.
    3. Gain experience in managing thread priorities and their impact on execution.
    4. Develop problem-solving skills for handling real-world race conditions in concurrent applications.
  - **Employee Management System**
    1. Learn how to implement CRUD operations using object-oriented programming.
    2. Understand the use of ArrayLists for storing and managing dynamic data.
    3. Develop skills in user input handling and data validation.
    4. Gain experience in implementing search and update functionalities within a collection.
  - **Card Collection System**
    1. Understand the basics of collections and their practical applications.
    2. Learn how to use loops and conditions to process user inputs efficiently.
    3. Gain experience in implementing search functionalities based on user-defined criteria.
    4. Develop a menu-driven application to interact with stored objects dynamically.