# Experiment 4

**Student Name: Akshit** **UID: 22BCS15229**
**Branch: CSE** **Section: 903_DL – B**
**Subject Code: 22CSH-352**
**Semester: 6th**
**Subject: Project Based learning with Java**

**Aim:** Use of Collections in Java. ArrayList, LinkedList, HashMap, TreeMap, HashSet in Java. Multithreading in Java. Thread Synchronization. Thread Priority, Thread LifeCycle.

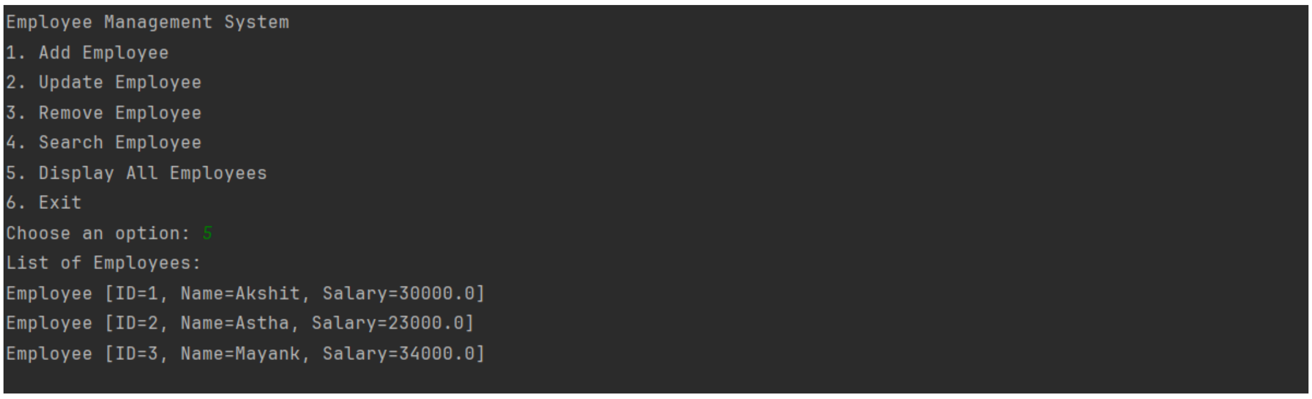
**Objective:** To implement and Use of Collections in Java. ArrayList, LinkedList, HashMap, TreeMap, HashSet in Java. Multithreading in Java. Thread Synchronization. Thread Priority, Thread LifeCycle.

## Code:

**Q1: 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 void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

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

public class EmployeeManagementSystem {
    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();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    updateEmployee();
                    break;
                case 3:
                    removeEmployee();
                    break;
                case 4:
                    searchEmployee();
                    break;
                case 5:
                    displayAllEmployees();
                    break;
                case 6:
                    System.out.println("Exiting the system...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }

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

        Employee employee = new Employee(id, name, salary);
        employees.add(employee);
        System.out.println("Employee added successfully!");
    }

    private static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        Employee employee = findEmployeeById(id);
        if (employee != null) {
            System.out.print("Enter new Employee Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter new Employee Salary: ");
            double salary = scanner.nextDouble();
            scanner.nextLine(); // Consume newline

            employee.setName(name);
            employee.setSalary(salary);
            System.out.println("Employee updated successfully!");
        } else {
            System.out.println("Employee with ID " + id + " not found.");
        }
    }

    private static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        Employee employee = findEmployeeById(id);
        if (employee != null) {
            employees.remove(employee);
            System.out.println("Employee removed successfully!");
        } else {
            System.out.println("Employee with ID " + id + " not found.");
        }
    }

    private static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        Employee employee = findEmployeeById(id);
        if (employee != null) {
            System.out.println("Employee found: " + employee);
        } else {
            System.out.println("Employee with ID " + id + " not found.");
        }
    }

    private static void displayAllEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            System.out.println("List of Employees:");
            for (Employee employee : employees) {
                System.out.println(employee);
            }
        }
    }

    private static Employee findEmployeeById(int id) {
        for (Employee employee : employees) {
            if (employee.getId() == id) {
                return employee;
            }
        }
        return null;
    }
}
```

**OUTPUT:**

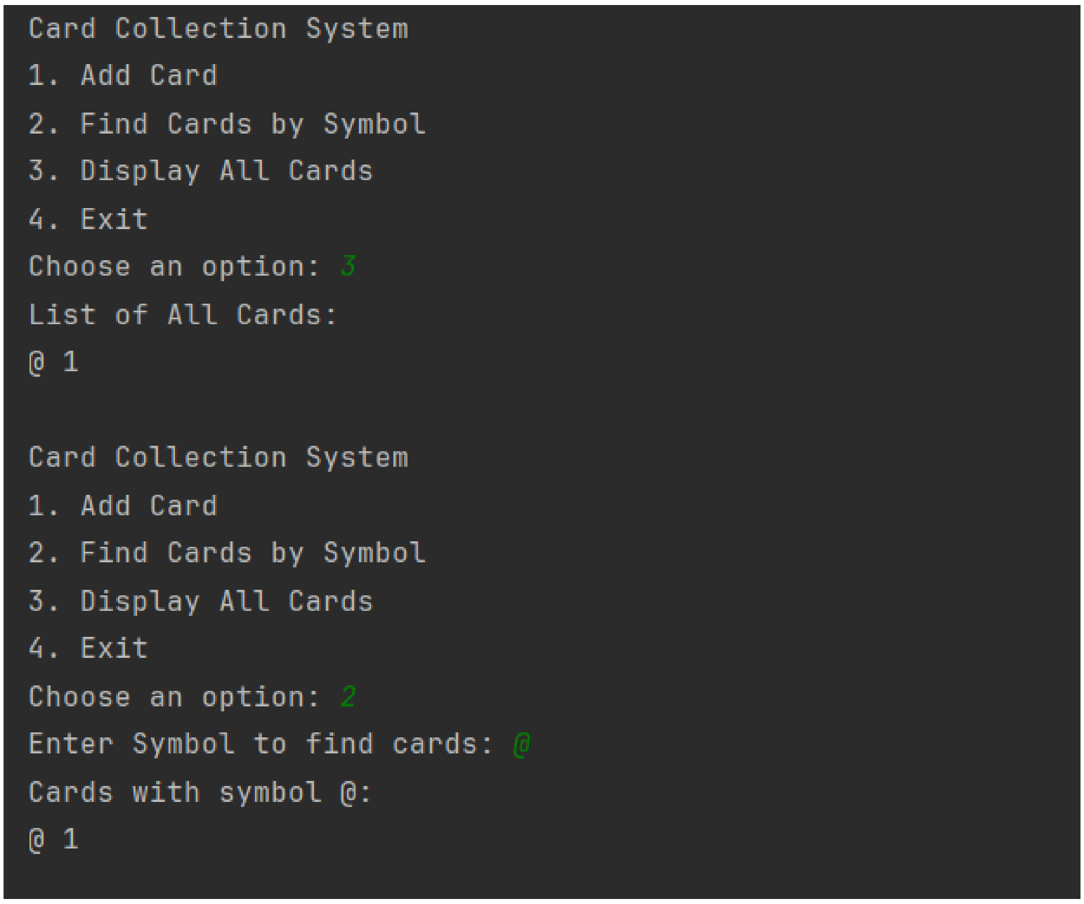


```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 5
List of Employees:
Employee [ID=1, Name=Akshit, Salary=30000.0]
Employee [ID=2, Name=Astha, Salary=23000.0]
Employee [ID=3, Name=Mayank, Salary=34000.0]
```

**Q2: 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 int number;

    public Card(String symbol, int number) {
        this.symbol = symbol;
        this.number = number;
    }

    public String getSymbol() {
        return symbol;
    }

    public int getNumber() {
        return number;
    }

    @Override
    public String toString() {
        return symbol + " " + number;
    }
}

public class CardCollectionSystem {
    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. Find Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    addCard();
                    break;
                case 2:
                    findCardsBySymbol();
                    break;
                case 3:
                    displayAllCards();
                    break;
                case 4:
                    System.out.println("Exiting the system...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }

    private static void addCard() {
        System.out.print("Enter Card Symbol: ");
        String symbol = scanner.nextLine();
        System.out.print("Enter Card Number: ");
        int number = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        Card card = new Card(symbol, number);
        cards.add(card);
        System.out.println("Card added successfully!");
    }

    private static void findCardsBySymbol() {
        System.out.print("Enter Symbol to find cards: ");
        String symbol = scanner.nextLine();

        List<Card> foundCards = new ArrayList<>();
        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                foundCards.add(card);
            }
        }

        if (foundCards.isEmpty()) {
            System.out.println("No cards found with symbol: " + symbol);
        } else {
            System.out.println("Cards with symbol " + symbol + ":");
            for (Card card : foundCards) {
                System.out.println(card);
            }
        }
    }

    private static void displayAllCards() {
        if (cards.isEmpty()) {
            System.out.println("No cards found.");
        } else {
            System.out.println("List of All Cards:");
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }
}
```

**OUTPUT:**



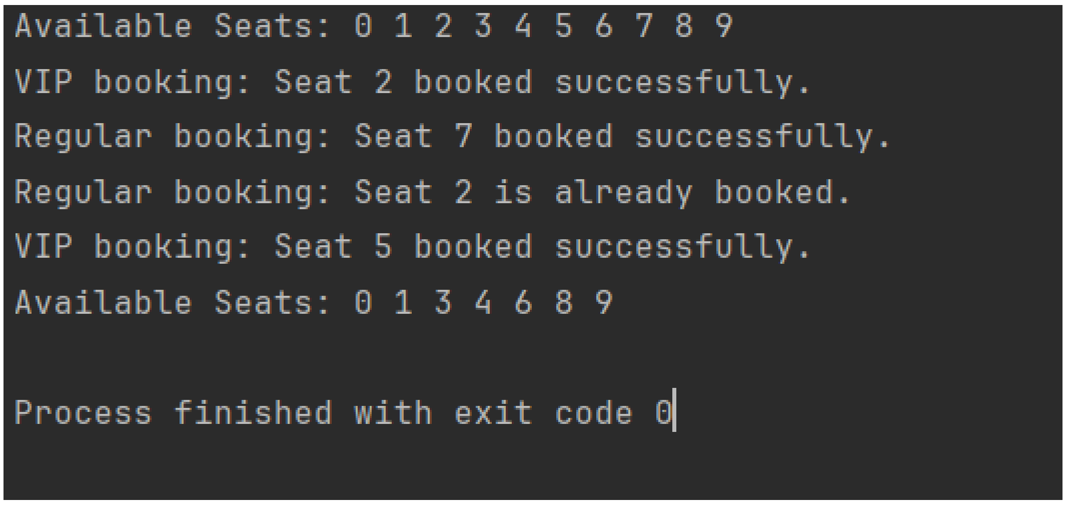
```
Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 3
List of All Cards:
@ 1

Card Collection System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 2
Enter Symbol to find cards: @
Cards with symbol @:
@ 1
```

## Q3: 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.

```
class TicketBookingSystem {
    private boolean[] seats; // Represents seats (true = booked, false = available)
    private int totalSeats;

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

    // Synchronized method to book a seat
    public synchronized boolean bookSeat(int seatNumber, String customerType) {
        if (seatNumber < 0 || seatNumber >= totalSeats) {
            System.out.println("Invalid seat number: " + seatNumber);
            return false;
        }
        if (!seats[seatNumber]) {
            seats[seatNumber] = true; // Book the seat
            System.out.println(customerType + " booking: Seat " + seatNumber + " booked
successfully.");
            return true;
        } else {
            System.out.println(customerType + " booking: Seat " + seatNumber + " is
already booked.");
            return false;
        }
    }

    // Method to display available seats
    public synchronized void displayAvailableSeats() {
        System.out.print("Available Seats: ");
        for (int i = 0; i < totalSeats; i++) {
            if (!seats[i]) {
                System.out.print(i + " ");
            }
        }
        System.out.println();
    }
}

class BookingThread extends Thread {
    private TicketBookingSystem bookingSystem;
    private String customerType;
    private int seatNumber;

    public BookingThread(TicketBookingSystem bookingSystem, String customerType,
int seatNumber) {
        this.bookingSystem = bookingSystem;
        this.customerType = customerType;
        this.seatNumber = seatNumber;
    }

    @Override
    public void run() {
        if (customerType.equals("VIP")) {
            this.setPriority(Thread.MAX_PRIORITY); // VIP bookings have higher
priority
        } else {
            this.setPriority(Thread.MIN_PRIORITY); // Regular bookings have lower
```

```
priority
        }
        bookingSystem.bookSeat(seatNumber, customerType);
    }
}

public class TicketBookingApp {
    public static void main(String[] args) {
        int totalSeats = 10; // Total number of seats
        TicketBookingSystem bookingSystem = new TicketBookingSystem(totalSeats);

        // Display initial available seats
        bookingSystem.displayAvailableSeats();

        // Create booking threads for VIP and regular customers
        BookingThread vip1 = new BookingThread(bookingSystem, "VIP", 2);
        BookingThread vip2 = new BookingThread(bookingSystem, "VIP", 5);
        BookingThread regular1 = new BookingThread(bookingSystem, "Regular", 2);
        BookingThread regular2 = new BookingThread(bookingSystem, "Regular", 7);

        // Start the threads
        vip1.start();
        vip2.start();
        regular1.start();
        regular2.start();

        // Wait for all threads to finish
        try {
            vip1.join();
            vip2.join();
            regular1.join();
            regular2.join();
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        }

        // Display final available seats
        bookingSystem.displayAvailableSeats();
    }
}
```

**Output**:



```
Available Seats: 0 1 2 3 4 5 6 7 8 9
VIP booking: Seat 2 booked successfully.
Regular booking: Seat 7 booked successfully.
Regular booking: Seat 2 is already booked.
VIP booking: Seat 5 booked successfully.
Available Seats: 0 1 3 4 6 8 9

Process finished with exit code 0
```

## Learning Outcomes:

1. **Learn how to use the synchronized keyword to ensure thread-safe access to shared resources (e.g., the seats array).**

2. **Understand the importance of synchronization in preventing race conditions and ensuring data consistency in multi-threaded environments.**

3. **Understand how higher-priority threads (e.g., VIP bookings) are given precedence over lower-priority threads (e.g., regular bookings).**

4. **Learn how to manage shared resources (e.g., the seats array) in a multi-threaded environment.**

5. **Understand the challenges of concurrent access to shared data and how to implement solutions to avoid conflicts**