Experiment- 04

Student Name: Pargat Singh UID: 22ICS10002
Branch: BE-CSE Section/Group: IOT-641/B
Semester: 6th Date of Performance: 05/03/2025
Subject Name: Project Based Learning in JAVA with Lab. Code: 22CSH-359

1. **Aim(EASY LEVEL)** : 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. **Objective:** The objective of this program is to implement an `ArrayList` in Java to manage employee records, allowing users to add, update, remove, and search employees efficiently..

3. **Implementation/Code:**

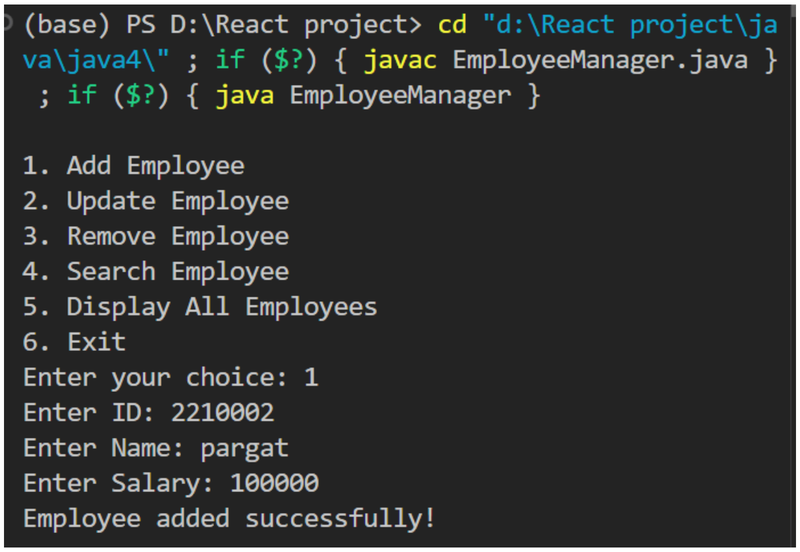
```
import java.util.ArrayList;
import java.util.Scanner;

class Employee {
    int id;
    String name;
    double salary;

    Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

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

```
public class EmployeeManager {
    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 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("Enter your choice: ");
            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.out.println("Exiting..."); return;
                default: System.out.println("Invalid choice! Try again.");
            }
        }
    }

    private static void addEmployee() {
        System.out.print("Enter 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 successfully!");
}

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 found.");
    } else {
```

```
                for (Employee emp : employees) {
                    System.out.println(emp);
                }
            }
        }
    }
```

4. **Output:**



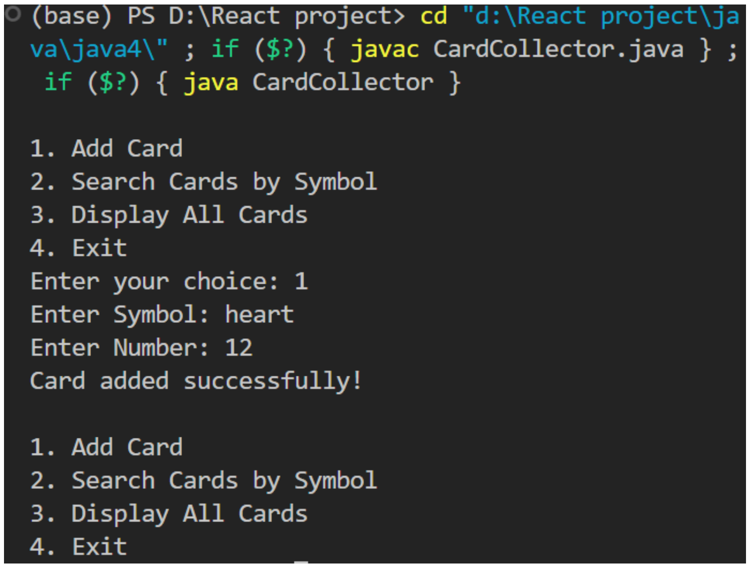
```
(base) PS D:\React project> cd "d:\React project\ja
va\java4\" ; if ($?) { javac EmployeeManager.java }
 ; if ($?) { java EmployeeManager }

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter ID: 2210002
Enter Name: pargat
Enter Salary: 100000
Employee added successfully!
```

AIM(MEDIUM LEVEL)- 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.

Implementation/Code:

```
import java.util.*;

class Card {
    String symbol;
    int number;

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

    public String toString() {
        return "Symbol: " + symbol + ", Number: " + number;
    }
}

public class CardCollector {
    private static Map<String, List<Card>> cardCollection = new HashMap<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        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 your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
            switch (choice) {
                case 1: addCard(); break;
                case 2: searchCardsBySymbol(); break;
                case 3: displayAllCards(); break;
                case 4: System.out.println("Exiting..."); return;
                default: System.out.println("Invalid choice! Try again.");
            }
        }
    }
    private static void addCard() {
        System.out.print("Enter Symbol: ");
        String symbol = scanner.nextLine();
        System.out.print("Enter Number: ");
        int number = scanner.nextInt();
        scanner.nextLine();

        cardCollection.putIfAbsent(symbol, new ArrayList<>());
        cardCollection.get(symbol).add(new Card(symbol, number));

        System.out.println("Card added successfully!");
    }

    private static void searchCardsBySymbol() {
        System.out.print("Enter Symbol to search: ");
        String symbol = scanner.nextLine();
        if (cardCollection.containsKey(symbol)) {
            for (Card card : cardCollection.get(symbol)) {
                System.out.println(card);
            }
        } else {
            System.out.println("No cards found for this symbol.");
        }
    }

    private static void displayAllCards() {
        if (cardCollection.isEmpty()) {
            System.out.println("No cards in the collection.");
        } else {
            for (List<Card> cards : cardCollection.values()) {
                for (Card card : cards) {
                    System.out.println(card);
                }
            }
        }
    }
}
```

Output:



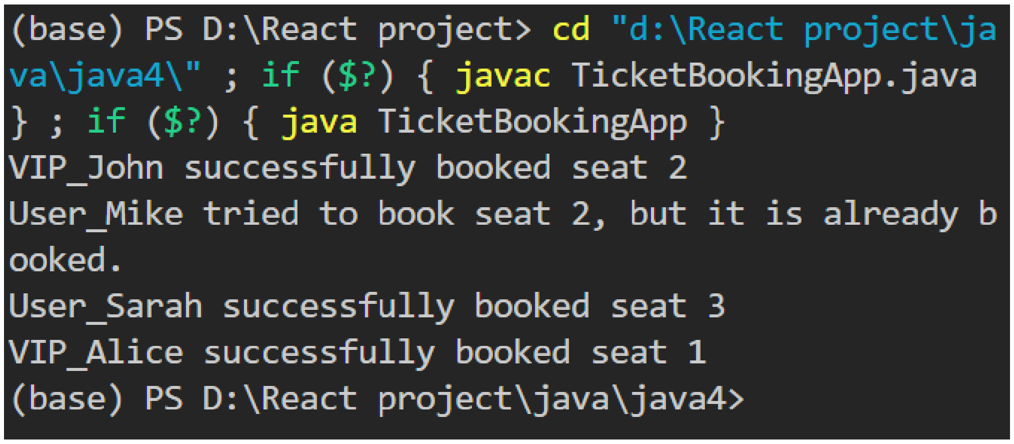
```
(base) PS D:\React project> cd "d:\React project\java\java4\" ; if ($?) { javac CardCollector.java } ; if ($?) { java CardCollector }

1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol: heart
Enter Number: 12
Card added successfully!

1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
```

Aim(HARD LEVEL): 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.

Implementation/Code:

```
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 customer) {
        if (seatNumber < 0 || seatNumber >= totalSeats) {
            System.out.println(customer + " tried to book an invalid seat.");
            return false;
        }
        if (!seats[seatNumber]) {
            seats[seatNumber] = true;
            System.out.println(customer + " successfully booked seat " + seatNumber);
            return true;
        } else {
                System.out.println(customer + " tried to book seat " + seatNumber + ", but it is
already booked.");
            return false;
        }
    }
}

class Customer extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final String customerName;

    public Customer(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) {
        TicketBookingSystem system = new TicketBookingSystem(5);

        Customer vip1 = new Customer(system, 2, "VIP_John", Thread.MAX_PRIORITY);
        Customer vip2 = new Customer(system, 1, "VIP_Alice", Thread.MAX_PRIORITY);
        Customer user1 = new Customer(system, 2, "User_Mike", Thread.NORM_PRIORITY);
        Customer user2 = new Customer(system, 3, "User_Sarah", Thread.MIN_PRIORITY);

        vip1.start();
        vip2.start();
        user1.start();
        user2.start();
    }
}
```

Output:



```
(base) PS D:\React project> cd "d:\React project\ja
va\java4\" ; if ($?) { javac TicketBookingApp.java
} ; if ($?) { java TicketBookingApp }
VIP_John successfully booked seat 2
User_Mike tried to book seat 2, but it is already b
ooked.
User_Sarah successfully booked seat 3
VIP_Alice successfully booked seat 1
(base) PS D:\React project\java\java4>
```

5. **Learning Outcomes:**

1. Java Collections – Practical use of `ArrayList`, `HashMap`, and `List`.
2. User Input Handling – Using `Scanner` to interact with users dynamically.
3. Concurrency Control – Managing multiple threads safely using `synchronized`.
4. Real-World Applications – Applying concepts to scenarios like employee records, card collections, and ticket bookings.