# Experiment 4

**Student Name: Ishika Thakur** **UID: 22BCS10765**
**Branch: BE/CSE** **Section/Group: 22BCS_IOT-618/B**
**Semester: 6th** **Date of Performance: 21/02/25**
**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:** The objective of this Java program is to develop applications using core Java concepts such as data structures, collections, and multithreading to efficiently store, manage, manipulate, and process data.
3. **Implementation/Code:**

**4.1: 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.**

**Code:**

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

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

    public 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 {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            try {
                System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Exit");
                System.out.print("Enter choice: ");
                int choice = scanner.nextInt();

                switch (choice) {
                    case 1:
                        addEmployee(employees, scanner);
                        break;
                    case 2:
                        updateEmployee(employees, scanner);
                        break;
                    case 3:
                        removeEmployee(employees, scanner);
                        break;
                    case 4:
                        searchEmployee(employees, scanner);
                        break;
                    case 5:
                        System.out.println("Exiting...");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Invalid choice! Please enter a number between 1 and 5.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input! Please enter a valid number.");
                scanner.nextLine(); // Clear the invalid input
            }
        }
    }

    private static void addEmployee(ArrayList<Employee> employees, Scanner scanner) {
        try {
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            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!");
        } catch (InputMismatchException e) {
            System.out.println("Invalid input! Please enter correct data types.");
            scanner.nextLine(); // Clear the invalid input
        }
    }

    private static void updateEmployee(ArrayList<Employee> employees, Scanner scanner) {
        try {
            System.out.print("Enter Employee ID to update: ");
            int updateId = scanner.nextInt();
            boolean found = false;

            for (Employee emp : employees) {
                if (emp.id == updateId) {
                    scanner.nextLine(); // Consume newline
                    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.");
                    found = true;
                    break;
                }
            }

            if (!found) {
                System.out.println("Employee ID not found!");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input! Please enter correct data types.");
            scanner.nextLine(); // Clear invalid input
        }
    }

    private static void removeEmployee(ArrayList<Employee> employees, Scanner scanner) {
        try {
            System.out.print("Enter Employee ID to remove: ");
            int removeId = scanner.nextInt();
            boolean removed = employees.removeIf(emp -> emp.id == removeId);

            if (removed) {
                System.out.println("Employee removed successfully.");
            } else {
                System.out.println("Employee ID not found!");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input! Please enter a valid Employee ID.");
            scanner.nextLine();
        }
    }

    private static void searchEmployee(ArrayList<Employee> employees, Scanner scanner) {
        try {
            System.out.print("Enter Employee ID to search: ");
            int searchId = scanner.nextInt();
            boolean found = false;

            for (Employee emp : employees) {
                if (emp.id == searchId) {
                    System.out.println(emp);
                    found = true;
                    break;
                }
            }

            if (!found) {
                System.out.println("Employee ID not found!");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input! Please enter a valid Employee ID.");
            scanner.nextLine();
        }
    }
}
```

**4.2: 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**.

**Code**:

```
import java.util.*;

class Card {
    private String suit;
    private String rank;

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

    public String getSuit() {
        return suit;
    }

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

class CardCollectionSystem {
    private Map<String, List<Card>> cardMap;

    public CardCollectionSystem() {
        cardMap = new HashMap<>();
    }

    public void addCard(String suit, String rank) {
        cardMap.putIfAbsent(suit, new ArrayList<>());
        List<Card> cards = cardMap.get(suit);

        // Check for duplicate
        for (Card card : cards) {
            if (card.getRank().equals(rank)) {
                System.out.println("Error: Card \"" + rank + " of " + suit + "\" already exists.");
                return;
            }
        }

        cards.add(new Card(suit, rank));
        System.out.println("Card added: " + rank + " of " + suit);
    }

    public void findCardsBySuit(String suit) {
        if (cardMap.containsKey(suit) && !cardMap.get(suit).isEmpty()) {
            System.out.println("Cards of " + suit + ":");
            for (Card card : cardMap.get(suit)) {
                System.out.println(card);
            }
        } else {
            System.out.println("No cards found for " + suit + ".");
        }
    }

    public void displayAllCards() {
        if (cardMap.isEmpty()) {
            System.out.println("No cards found.");
            return;
        }

        System.out.println("All Cards:");
        for (List<Card> cards : cardMap.values()) {
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }

    public void removeCard(String suit, String rank) {
        if (cardMap.containsKey(suit)) {
            List<Card> cards = cardMap.get(suit);
            for (Iterator<Card> iterator = cards.iterator(); iterator.hasNext();) {
                Card card = iterator.next();
                if (card.getRank().equals(rank)) {
                    iterator.remove();
                    System.out.println("Card removed: " + rank + " of " + suit);
                    return;
                }
            }
        }
        System.out.println("Error: Card \"" + rank + " of " + suit + "\" not found.");
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CardCollectionSystem ccs = new CardCollectionSystem();
        int choice;
```

```
do {
    System.out.println("\nCard 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:
            System.out.print("Enter Suit: ");
            String suit = scanner.nextLine();
            System.out.print("Enter Rank: ");
            String rank = scanner.nextLine();
            ccs.addCard(suit, rank);
            break;

        case 2:
            System.out.print("Enter Suit to search: ");
            String searchSuit = scanner.nextLine();
            ccs.findCardsBySuit(searchSuit);
            break;

        case 3:
            ccs.displayAllCards();
            break;

        case 4:
            System.out.print("Enter Suit: ");
            String removeSuit = scanner.nextLine();
            System.out.print("Enter Rank: ");
            String removeRank = scanner.nextLine();
            ccs.removeCard(removeSuit, removeRank);
            break;
```

```
                case 5:
                    System.out.println("Exiting the system.");
                    break;

                default:
                    System.out.println("Invalid choice. Please try again.");
                    break;
            }
        } while (choice != 5);

        scanner.close();
    }
}
```

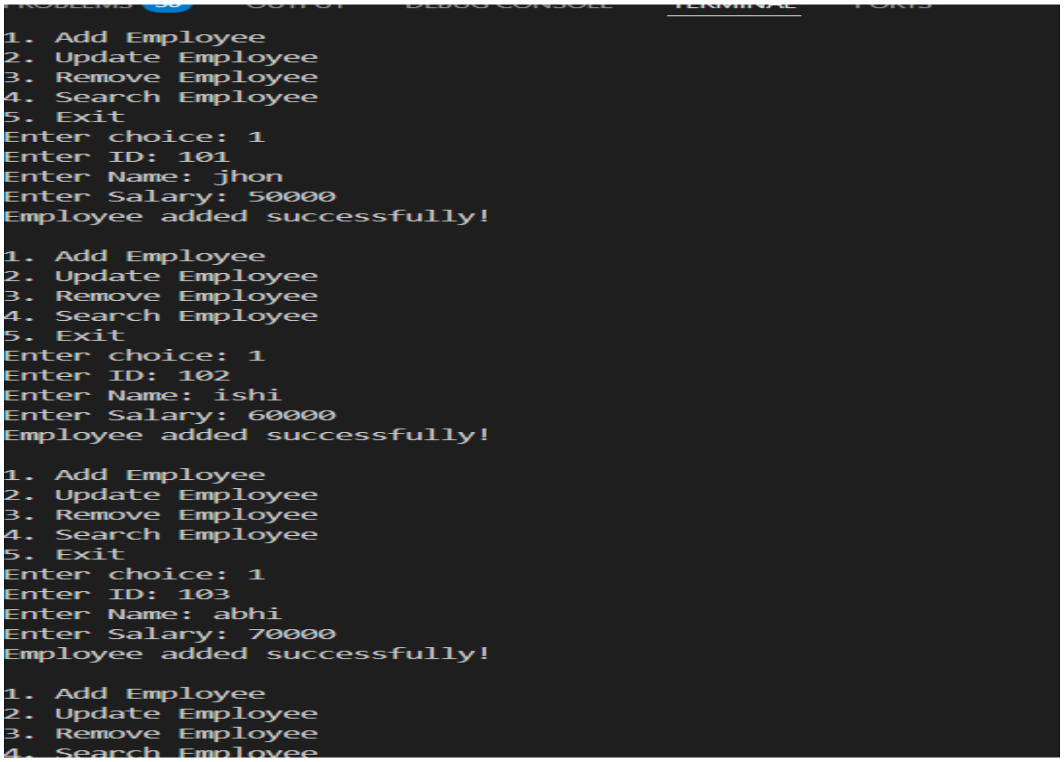
**4.3: Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.**

**Code:**

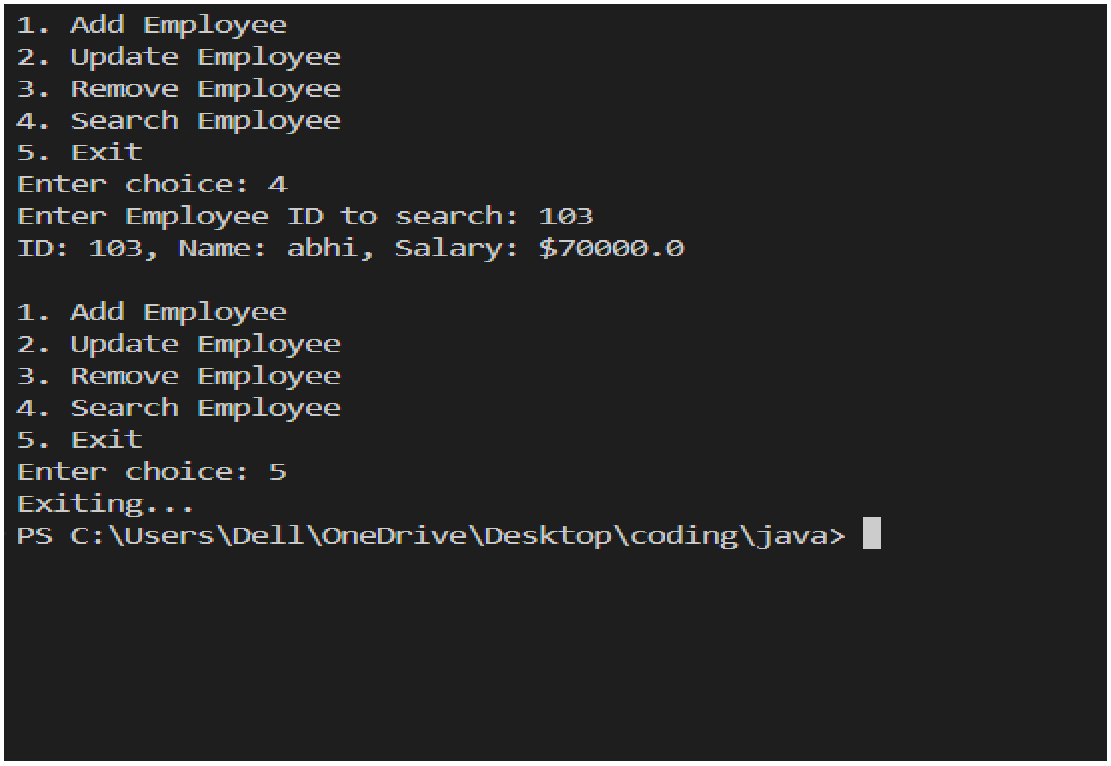
```
import java.util.concurrent.atomic.AtomicInteger;

class TicketBookingSystem {
    private int totalSeats;
    private AtomicInteger availableSeats;

    public TicketBookingSystem(int seats) {
        this.totalSeats = seats;
        this.availableSeats = new AtomicInteger(seats);
    }

    public synchronized boolean bookSeat(String userType, String userName) {
        if (availableSeats.get() > 0) {
            int seatNumber = totalSeats - availableSeats.addAndGet(-1);
            System.out.println(userType + " User " + userName + " successfully booked seat #" +
seatNumber);
            return true;
        } else {
            System.out.println("Sorry, " + userName + "! No seats available.");
            return false;
        }
    }}
class User extends Thread {
    private TicketBookingSystem system;
    private String userType;
    private String userName;

    public User(TicketBookingSystem system, String userType, String userName, int priority)
{
        this.system = system;
        this.userType = userType;
        this.userName = userName;
        setPriority(priority);
    }
    @Override
    public void run() {
        system.bookSeat(userType, userName);
    }}
public class TicketBookingApp {
    public static void main(String[] args) {
        System.out.println("Welcome to the Ticket Booking System!\n");

        TicketBookingSystem bookingSystem = new TicketBookingSystem(5);

        User u1 = new User(bookingSystem, "VIP", "John", Thread.MAX_PRIORITY);
        User u2 = new User(bookingSystem, "Regular", "Alice", Thread.NORM_PRIORITY);
        User u3 = new User(bookingSystem, "VIP", "Emma", Thread.MAX_PRIORITY);
        User u4 = new User(bookingSystem, "Regular", "Bob", Thread.NORM_PRIORITY);
        User u5 = new User(bookingSystem, "Regular", "Charlie", Thread.NORM_PRIORITY);
        User u6 = new User(bookingSystem, "VIP", "Olivia", Thread.MAX_PRIORITY);
        User u7 = new User(bookingSystem, "Regular", "David", Thread.NORM_PRIORITY);
        u1.start();
        u3.start();
        u6.start();
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        u2.start();
        u4.start();
        u5.start();
        u7.start();
        try {
            u1.join();
            u2.join();
            u3.join();
            u4.join();
            u5.join();
            u6.join();
            u7.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("\nSystem shutting down...");
    }}
```

## 4. Output:

### 4.1

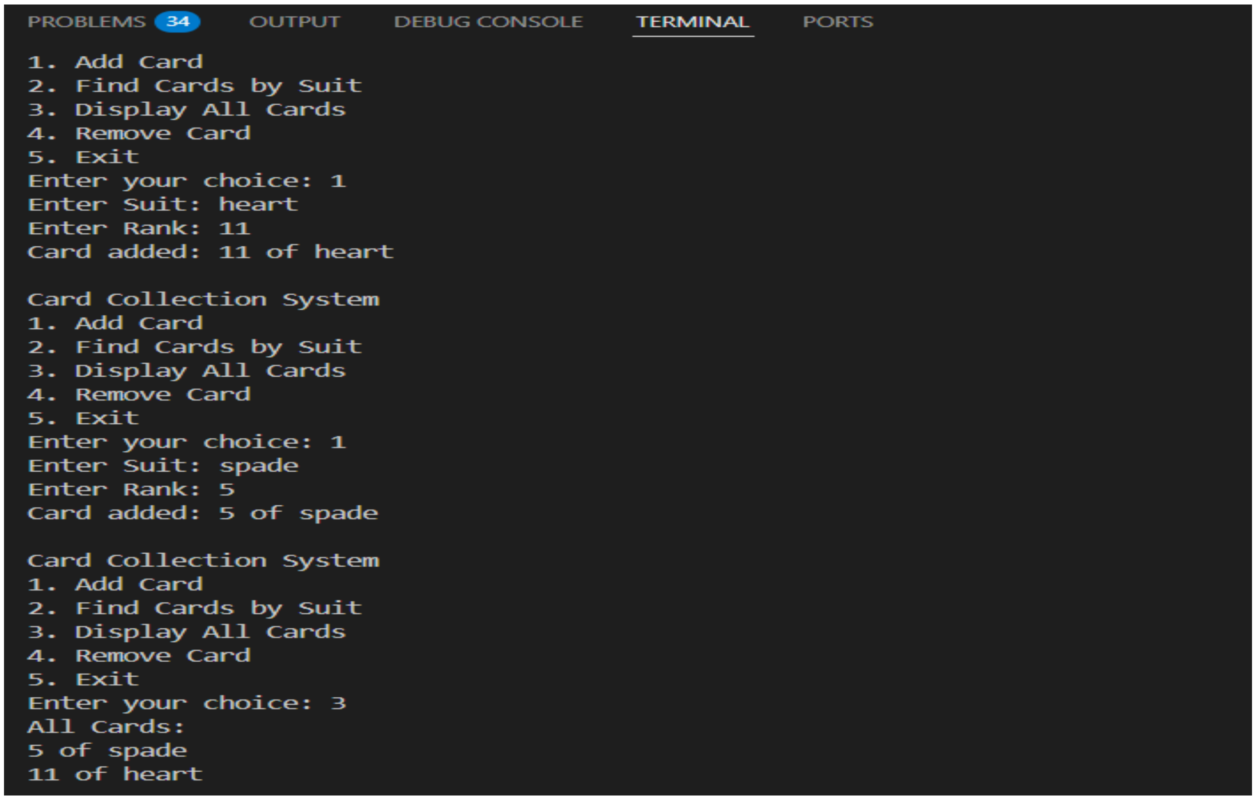


```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 1
Enter ID: 101
Enter Name: jhon
Enter Salary: 50000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 1
Enter ID: 102
Enter Name: ishi
Enter Salary: 60000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 1
Enter ID: 103
Enter Name: abhi
Enter Salary: 70000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
```



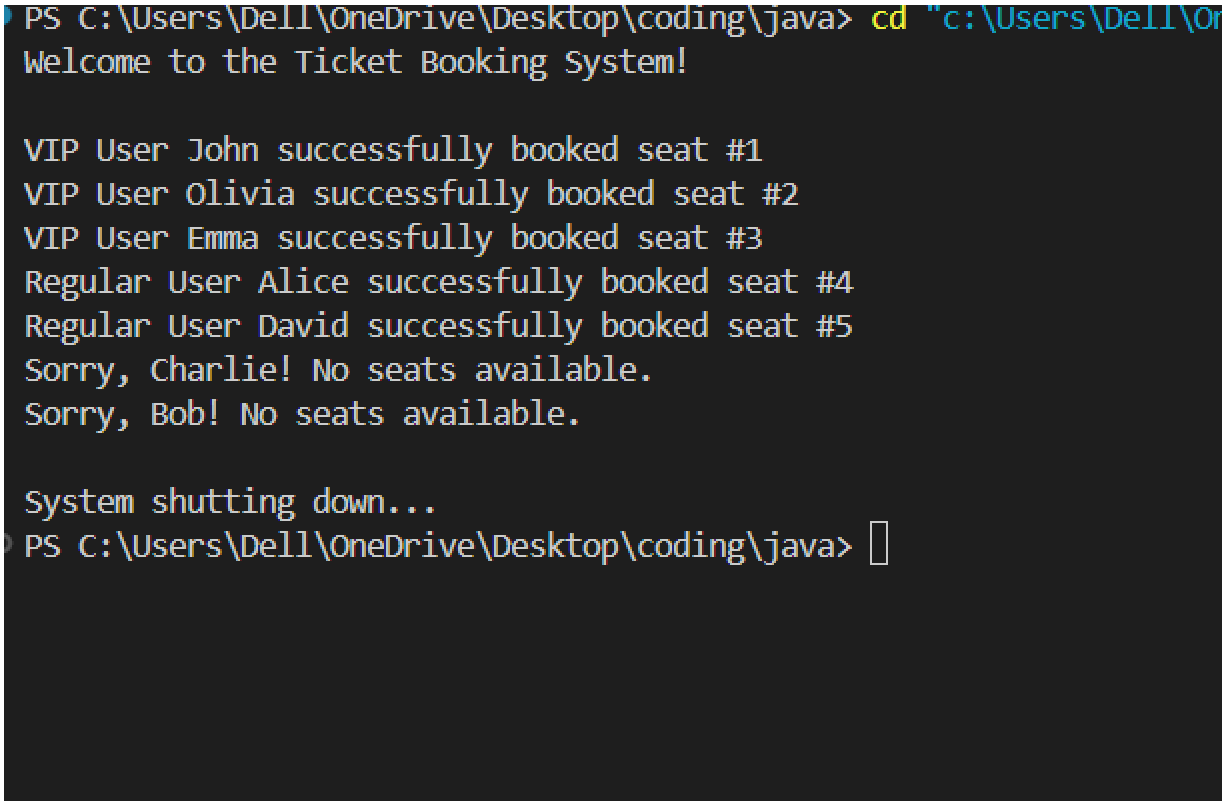
```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 4
Enter Employee ID to search: 103
ID: 103, Name: abhi, Salary: $70000.0

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 5
Exiting...
PS C:\Users\Dell\OneDrive\Desktop\coding\java>
```

**4.2**



```
1. Add Card
2. Find Cards by Suit
3. Display All Cards
4. Remove Card
5. Exit
Enter your choice: 1
Enter Suit: heart
Enter Rank: 11
Card added: 11 of heart

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 Suit: spade
Enter Rank: 5
Card added: 5 of spade

Card Collection System
1. Add Card
2. Find Cards by Suit
3. Display All Cards
4. Remove Card
5. Exit
Enter your choice: 3
All Cards:
5 of spade
11 of heart
```

**4.3**



```
PS C:\Users\Dell\OneDrive\Desktop\coding\java> cd "c:\Users\Dell\Or
Welcome to the Ticket Booking System!

VIP User John successfully booked seat #1
VIP User Olivia successfully booked seat #2
VIP User Emma successfully booked seat #3
Regular User Alice successfully booked seat #4
Regular User David successfully booked seat #5
Sorry, Charlie! No seats available.
Sorry, Bob! No seats available.

System shutting down...
PS C:\Users\Dell\OneDrive\Desktop\coding\java>
```

## 5. Learning Outcome:

- Understanding list operations such as insertion, deletion, searching, and displaying elements in Java.
- Applying OOP concepts like encapsulation and method abstraction to manage list operations efficiently.
- Handling user input using the Scanner class for interactive program execution.
- Utilizing control structures like loops and conditional statements to implement list operations dynamically.
- Enhancing problem-solving skills by organizing and manipulating string data in a structured manner.