Experiment 4.1

Student Name: Vivek Garg **UID: 22BCS16972**
Branch: CSE **Section: 22BCS_EPAM-801/B**
Semester: 6th **DOP:17/02/25**
Subject: PBLJ **Subject Code:22CSH-359**

1. Aim:

Write a Program to perform the basic operations like insert, delete, display and search in list. List contains String object items where these operations are to be performed.

2. Objective:

The objective of this program is to implement basic operations (insert, delete, display, and search) on a List containing String objects. The program will demonstrate how to manipulate a list using common list operations in Java, providing functionality to manage and interact with data stored in the list.

3. Implementation/Code:

```
import java.util.*;

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;
    }

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

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

        while (true) {
            System.out.println("\n1. Add Employee | 2. Update | 3. Remove | 4. Search | 5. Display | 6.
                    Exit");
            System.out.print("Enter your choice: ");
            int choice = sc.nextInt();
```

```
            if (choice == 6) {
                System.out.println("Exiting...");
                sc.close();
                return;
            }
            System.out.print("Enter Employee ID: ");
            int id = sc.nextInt();
            sc.nextLine(); // Consume newline

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Salary: ");
                    double salary = sc.nextDouble();
                    employees.add(new Employee(id, name, salary));
                    System.out.println("Employee added.");
                }
                case 2 -> {
                    Employee emp = findEmployee(employees, id);
                    if (emp != null) {
                        System.out.print("Enter new Name: ");
                        emp.name = sc.nextLine();
                        System.out.print("Enter new Salary: ");
                        emp.salary = sc.nextDouble();
                        System.out.println("Employee updated.");
                    } else {
                        System.out.println("Employee not found.");
                    }
                }
                case 3 -> {
                    if (employees.removeIf(e -> e.id == id)) {
                        System.out.println("Employee removed.");
                    } else {
                        System.out.println("Employee not found.");
                    }
                }
                case 4 -> {
                    Employee emp = findEmployee(employees, id);
                    System.out.println(emp != null ? "Employee Found: " + emp : "Employee not
                        found.");
                }
                case 5 -> {
                    if (employees.isEmpty()) System.out.println("No employees found.");
                    else employees.forEach(System.out::println);
                }
                default -> System.out.println("Invalid choice! Try again.");
            }
        }
    }
```

```
    private static Employee findEmployee(List<Employee> employees, int id) {
        return employees.stream().filter(e -> e.id == id).findFirst().orElse(null);
    }
}
```

4. Output

```
PS D:\PBLJ>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetail
orkspaceStorage\9821a2369c944fde6ad1ceda6f40c905\redhat.java\jdt_ws\PBLJ_72dc80

1. Add Employee | 2. Update | 3. Remove | 4. Search | 5. Display | 6. Exit
Enter your choice: 1
Enter Employee ID: 01
Enter Name: Vivek
Enter Salary: 100000
Employee added.

1. Add Employee | 2. Update | 3. Remove | 4. Search | 5. Display | 6. Exit
Enter your choice: 1
Enter Employee ID: 02
Enter Name: Kushan
Enter Salary: 10000
Employee added.

1. Add Employee | 2. Update | 3. Remove | 4. Search | 5. Display | 6. Exit
Enter your choice: 4
Enter Employee ID: 01
Employee Found: ID: 1, Name: Vivek, Salary: 100000.0

1. Add Employee | 2. Update | 3. Remove | 4. Search | 5. Display | 6. Exit
Enter your choice: 5
Enter Employee ID: 01
ID: 1, Name: Vivek, Salary: 100000.0
ID: 2, Name: Kushan, Salary: 10000.0

1. Add Employee | 2. Update | 3. Remove | 4. Search | 5. Display | 6. Exit
Enter your choice: 6
Exiting...
PS D:\PBLJ>
```

5. Learning Outcome

1. Proficiency in Java Basics
2. Understanding Data Structures
3. Improved Problem-Solving Skills
4. Application of Encapsulation and Modularity

Experiment 4.2

1. Aim:

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.

2. Objective:

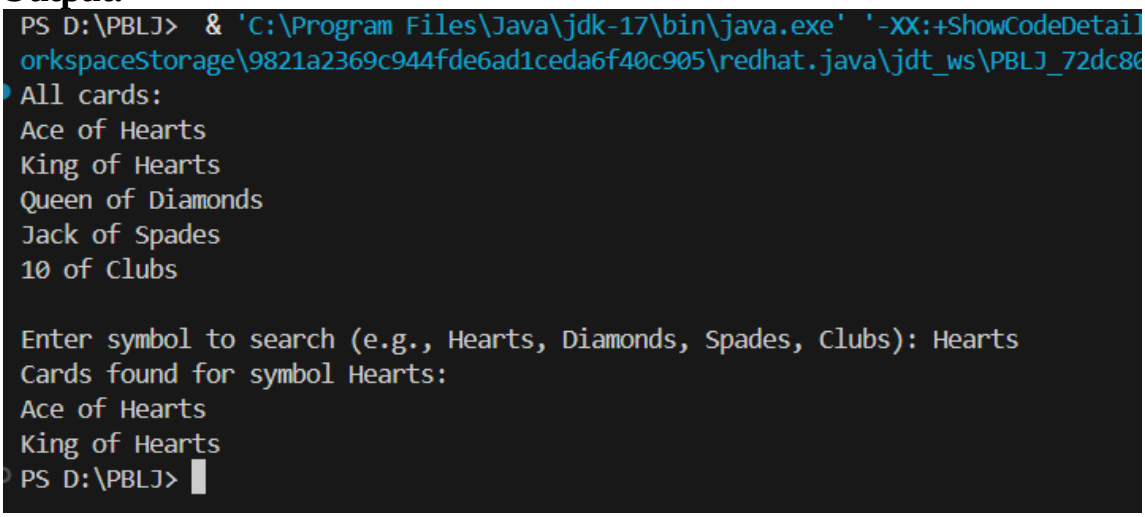
The objective of this program is to utilize the `Collection` interface in Java to store and manage a set of playing cards efficiently. It allows users to add cards, search for all cards of a specific symbol, and display the stored cards. This helps in organizing and retrieving cards based on their symbols in an easy and structured manner.

3. Implementation/Code:

```
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;
    }
    public String getValue() {
        return value;
    }
    @Override
    public String toString() {
        return value + " of " + symbol;
    }
}

class CardCollection {
    private Collection<Card> cards;

    public CardCollection() {
        this.cards = new ArrayList<>();
    }
    public void addCard(String symbol, String value) {
        cards.add(new Card(symbol, value));
    }

    public List<Card> findCardsBySymbol(String symbol) {
        List<Card> result = new ArrayList<>();
        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
```

```
                result.add(card);
            }
        }
        return result;
    }

    public void displayAllCards() {
        for (Card card : cards) {
            System.out.println(card);
        }
    }
}

public class CardManager {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CardCollection collection = new
CardCollection();

        collection.addCard("Hearts", "Ace");
        collection.addCard("Hearts", "King");
        collection.addCard("Diamonds", "Queen");
        collection.addCard("Spades", "Jack");
        collection.addCard("Clubs", "10");

        System.out.println("All cards:");
        collection.displayAllCards();

        System.out.print("\nEnter symbol to search (e.g., Hearts, Diamonds, Spades, Clubs): ");
        String searchSymbol = scanner.nextLine();
        List<Card> foundCards = collection.findCardsBySymbol(searchSymbol);

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

4. **Output:**



5. **Learning Outcomes:**
   1. Proficiency in Java Collections
   2. Enhanced Object Oriented Programming Skills
   3. Hands-on experience with Lists
   4. Efficient Data Filtering Techniques

Experiment 4.3

1. Aim:

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.

2. Objective:

The objective of this program is to utilize the `Collection` interface in Java to store and manage a set of playing cards efficiently. It allows users to add cards, search for all cards of a specific symbol, and display the stored cards. This helps in organizing and retrieving cards based on their symbols in an easy and structured manner.

3. Implementation/Code:

```
import java.util.*;

class TicketBookingSystem {
    private int availableSeats;
    private Set<Integer> bookedSeats;

    public TicketBookingSystem(int totalSeats) {
        this.availableSeats = totalSeats;
        this.bookedSeats = new HashSet<>();
    }

    public synchronized boolean bookSeat(int seatNumber, String user) {
        if (bookedSeats.contains(seatNumber)) {
            System.out.println(user + " attempted to book Seat " + seatNumber + ", but it's already
  taken!");
            return false;
        } else {
            bookedSeats.add(seatNumber);
            System.out.println(user + " successfully booked Seat " + seatNumber);
            return true;
        }
    }
}

class User extends Thread {
    private TicketBookingSystem system;
    private int seatNumber;
    private String userName;

    public User(TicketBookingSystem system, int seatNumber, String userName, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.userName = userName;
        setPriority(priority);
    }
```

```
    @Override
    public void run() {
        system.bookSeat(seatNumber, userName);
    }
}

public class TicketBookingMain {
    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem(5);

        User u1 = new User(bookingSystem, 1, "VIP_User1", Thread.MAX_PRIORITY);
        User u2 = new User(bookingSystem, 2, "Regular_User1", Thread.NORM_PRIORITY);
        User u3 = new User(bookingSystem, 3, "VIP_User2", Thread.MAX_PRIORITY);
        User u4 = new User(bookingSystem, 1, "Regular_User2", Thread.NORM_PRIORITY);
        User u5 = new User(bookingSystem, 4, "VIP_User3", Thread.MAX_PRIORITY);
        User u6 = new User(bookingSystem, 5, "Regular_User3", Thread.NORM_PRIORITY);

        u1.start();
        u2.start();
        u3.start();
        u4.start();
        u5.start();
        u6.start();
    }
     }
```

4. Output:

```
PS D:\PBLJ>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+Sho
orkspaceStorage\9821a2369c944fde6ad1ceda6f40c905\redhat.java\jdt_ws\
VIP_User1 successfully booked Seat 1
Regular_User3 successfully booked Seat 5
VIP_User3 successfully booked Seat 4
Regular_User2 attempted to book Seat 1, but it's already taken!
VIP_User2 successfully booked Seat 3
Regular_User1 successfully booked Seat 2
PS D:\PBLJ>
```

5. Learning Outcomes:

1. Proficiency in Multi-Threading
2. Understanding Thread Synchronization
3. Improved Problem-Solving Skills
4. Concurrency Management