Experiment-4

Name: Sakshi Keshri **UID: 22BCS12460**
Branch: BE-CSE **Section/Group: IOT_642/A**
Semester:6th **Date of Performance:18-02-25**
Subject Name: Project Based Learning in Java **Subject Code: 22CSH-359**

Problem1:

1. **Aim :** 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 :** To store the details of an Employee in Arraylist and allow them to use the functionalities.

3. **Code:**

```
import java.util.*;

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 {
   static ArrayList<Employee> employees = new ArrayList<>();
   static Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
      while (true) {
         System.out.println("\n1. Add  2. Update  3. Remove  4. Search  5. Display
6. Exit");
         switch (scanner.nextInt()) {
            case 1 -> add(); case 2 -> update(); case 3 -> remove(); case 4 ->
search(); case 5 -> display(); case 6 -> System.exit(0);
```

```
                default -> System.out.println("Invalid choice!");

            }
        }
    }

    static void add() {
        System.out.print("ID: "); int id = scanner.nextInt(); scanner.nextLine();
        System.out.print("Name: "); String name = scanner.nextLine();
        System.out.print("Salary: "); double salary = scanner.nextDouble();
        employees.add(new Employee(id, name, salary));
    }

    static void update() {
        System.out.print("Enter ID: "); int id = scanner.nextInt();
        for (Employee e : employees) if (e.id == id) {
            scanner.nextLine(); System.out.print("New Name: "); e.name =
scanner.nextLine();
            System.out.print("New Salary: "); e.salary = scanner.nextDouble(); return;
        }
        System.out.println("Not found!");
    }

    static void remove() {
        System.out.print("Enter ID: "); int id = scanner.nextInt();
        employees.removeIf(e -> e.id == id);
    }

    static void search() {
        System.out.print("Enter ID: "); int id = scanner.nextInt();
        employees.stream().filter(e -> e.id ==
id).findFirst().ifPresentOrElse(System.out::println, () -> System.out.println("Not
found!"));
    }

    static void display() {
        employees.forEach(System.out::println);
    }
}
```

4. Output:

```
C:\Users\DELL\.jdks\openjdk-22\bin\java.exe ...

1. Add  2. Update  3. Remove  4. Search  5. Display  6. Exit
1
ID: 121
Name: Akshay
Salary: 30000

1. Add  2. Update  3. Remove  4. Search  5. Display  6. Exit
1
ID: 122
Name: Akash
Salary: 45000

1. Add  2. Update  3. Remove  4. Search  5. Display  6. Exit
2
Enter ID: 121
New Name: Shanya
New Salary: 55000

1. Add  2. Update  3. Remove  4. Search  5. Display  6. Exit
4
Enter ID: 122
ID: 122, Name: Akash, Salary: 45000.0
```

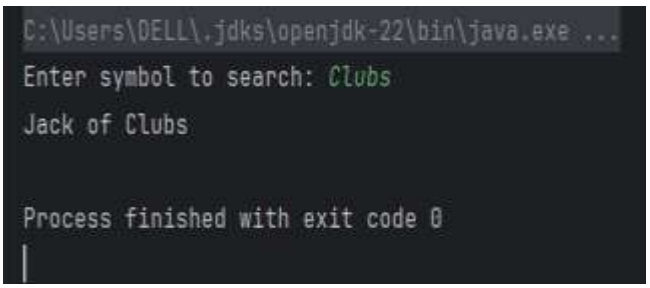
Problem2:

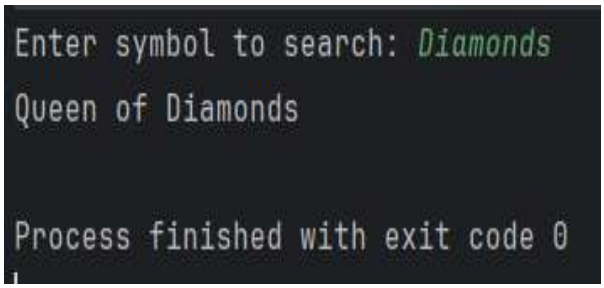
1. **Aim:** Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.
2. **Objective:** Allow users to search for all cards belonging to a specific symbol (e.g., Hearts, Spades).
3. **Code:**

```
import java.util.*;

class Card {
    String symbol, value;
    Card(String symbol, String value) { this.symbol = symbol; this.value = value; }
    public String toString() { return value + " of " + symbol; }
}

public class CardCollection {
    static Collection<Card> cards = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        addCards();
        System.out.print("Enter symbol to search: ");
        String symbol = scanner.next();
        findCards(symbol);
    }
    static void addCards() {
        cards.add(new Card("Hearts", "Ace"));
        cards.add(new Card("Spades", "King"));
```

```
        cards.add(new Card("Diamonds", "Queen"));
        cards.add(new Card("Hearts", "10"));
        cards.add(new Card("Clubs", "Jack"));
    }
    static void findCards(String symbol) {
        boolean found = false;
        for (Card c : cards) {
            if (c.symbol.equalsIgnoreCase(symbol)) {
                System.out.println(c);
                found = true;
            }
        }
        if (!found) System.out.println("No cards found for symbol: " + symbol);
    }
}
```

4. Output:





Problem3:

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:** To implement ticket booking system effectively with synchronized threads.
3. **Code:**

```
import java.util.*;

class TicketBookingSystem {
    private int availableSeats = 5;
    public synchronized void bookTicket(String name) {
        if (availableSeats > 0) {
            System.out.println(name + " booked a seat. Remaining: " + (--availableSeats));
        } else {
            System.out.println(name + " failed to book. No seats available.");
        }
    }
}

class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final String name;

    public BookingThread(TicketBookingSystem system, String name, int priority) {
        this.system = system;
        this.name = name;
        setPriority(priority);
```

```
}
    public void run() {
        system.bookTicket(name);
    }
}

public class TicketBookingApp {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        Thread vip1 = new BookingThread(system, "VIP1", Thread.MAX_PRIORITY);
        Thread vip2 = new BookingThread(system, "VIP2", Thread.MAX_PRIORITY);
        Thread user1 = new BookingThread(system, "User1", Thread.NORM_PRIORITY);
        Thread user2 = new BookingThread(system, "User2", Thread.NORM_PRIORITY);
        Thread user3 = new BookingThread(system, "User3", Thread.NORM_PRIORITY);

        vip1.start();
        vip2.start();
        user1.start();
        user2.start();
        user3.start();
    }
}
```

4. Output:

```
VIP1 booked a seat. Remaining: 6
User3 booked a seat. Remaining: 5
VIP2 booked a seat. Remaining: 4
User2 booked a seat. Remaining: 3
User1 booked a seat. Remaining: 2

Process finished with exit code 0
```

5. Learning Outcomes:

i. Understand how to use maps (dictionaries) for efficient data storage and retrieval.

ii. Learn to group and organize data based on a key attribute.

iii. Gain experience in handling user input and storing objects dynamically.
iv. Learn how to use switch cases in a program.
v. Understood the working of Ticket ooking system.