Experiment 4

Student Name: Vishal Sah **UID: 22BCS16978**
Branch: BE-CSE **Section/Group: 641-B**
Semester:6th **Date of Performance:05/02/25 Subject**
Name: PBLJ **Subject Code: 22CSH-359**

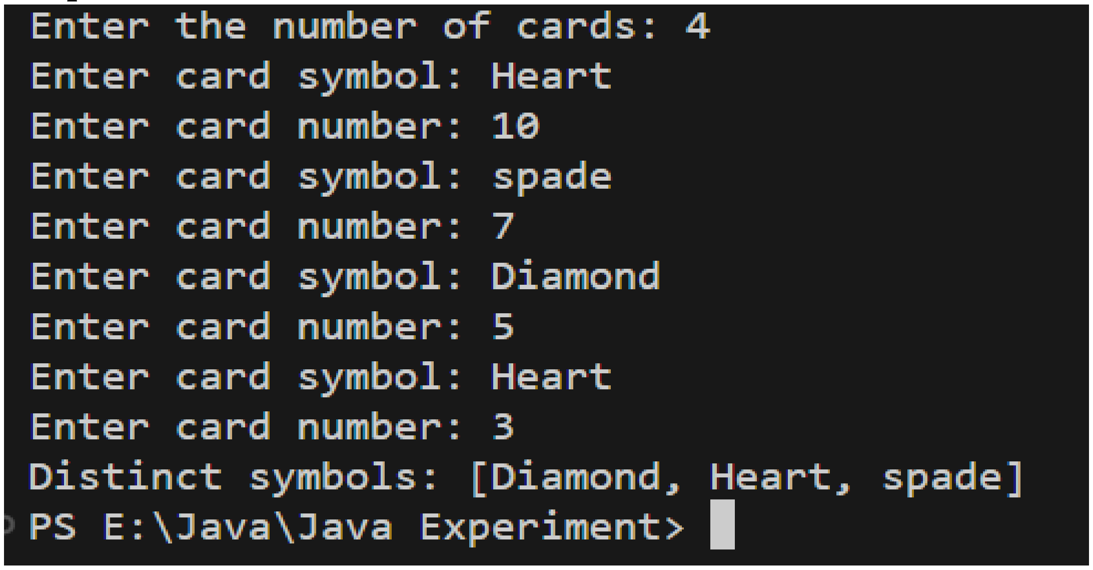
1. **Aim :** Write a program to collect and store all the cards to assist the users in finding all the cards in a given symbol.This cards game consist of N number of cards. Get N number of cards details from the user and store the values in Card object with the attributes symbol and Number. Store all the cards in a map with symbols as its key and list of cards as its value. Map is used here to easily group all the cards based on their symbol. Once all the details are captured print all the distinct symbols in alphabetical order from the Map.

2. **Objective :** This program collects and stores N cards, grouping them by symbol in a map for easy retrieval. It displays distinct symbols in alphabetical order along with their associated cards, total count, and sum of numbers, ensuring efficient organization and user-friendly output.

3. **Code**

```java
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 CardCollectionProgram4 {
    private final Map<String, List<Card>> cards = new HashMap<>();

    public void addCard(Card card) {
        cards.computeIfAbsent(card.getSymbol(), k -> new
ArrayList<>()).add(card);
    }

    public void printDistinctSymbols() {
        List<String> symbols = new ArrayList<>(cards.keySet());
        Collections.sort(symbols);
        System.out.println("Distinct symbols: " + symbols);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CardCollectionProgram4 ccp = new CardCollectionProgram4();
        System.out.print("Enter the number of cards: ");
        int n = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter card symbol: ");
            String symbol = sc.nextLine();
            System.out.print("Enter card number: ");
            int number = sc.nextInt();
            sc.nextLine();
            ccp.addCard(new Card(symbol, number));
        }

        ccp.printDistinctSymbols();
        sc.close();
    }
}
```

4. Output



```
Enter the number of cards: 4
Enter card symbol: Heart
Enter card number: 10
Enter card symbol: spade
Enter card number: 7
Enter card symbol: Diamond
Enter card number: 5
Enter card symbol: Heart
Enter card number: 3
Distinct symbols: [Diamond, Heart, spade]
PS E:\Java\Java Experiment>
```

5. Learning Outcomes

- Understand how to use maps (dictionaries) for efficient data storage and retrieval.
- Learn to group and organize data based on a key attribute.
- Gain experience in handling user input and storing objects dynamically.
- Develop skills in sorting and displaying structured data in a meaningful

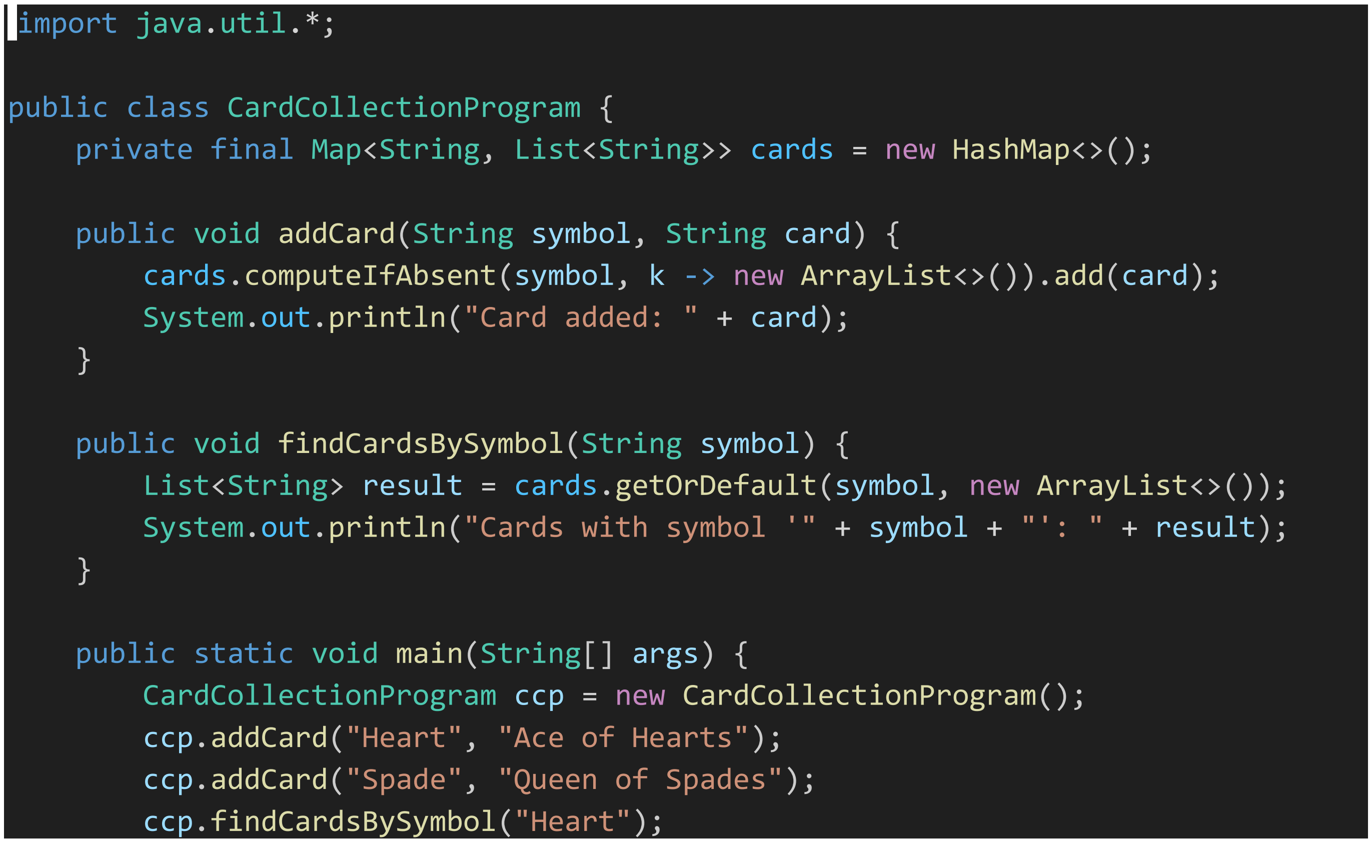
Experiment 4.2

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.

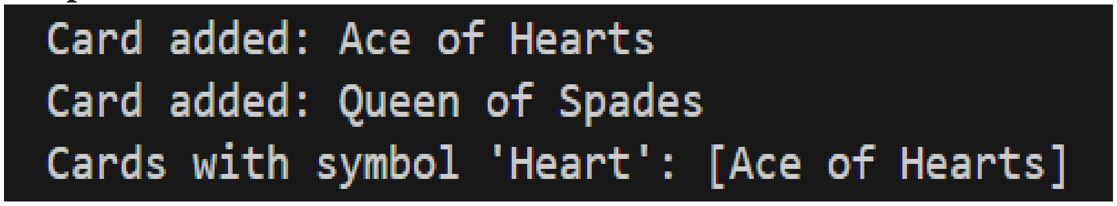
2. **Objective-**
   The objective of this program is to create a card collection system using the Collection interface in Java. It will allow users to add, store, and manage cards with associated symbols, providing an efficient search feature to find all cards related to a specific symbol. The system will use Collection types like `List`, `Set`, or `Map` to organize data, offering dynamic addition, removal, and real-time updates to the collection, along with a simple and intuitive interface.

3. **Code:**



```java
import java.util.*;

public class CardCollectionProgram {
    private final Map<String, List<String>> cards = new HashMap<>();

    public void addCard(String symbol, String card) {
        cards.computeIfAbsent(symbol, k -> new ArrayList<>()).add(card);
        System.out.println("Card added: " + card);
    }

    public void findCardsBySymbol(String symbol) {
        List<String> result = cards.getOrDefault(symbol, new ArrayList<>());
        System.out.println("Cards with symbol '" + symbol + "': " + result);
    }

    public static void main(String[] args) {
        CardCollectionProgram ccp = new CardCollectionProgram();
        ccp.addCard("Heart", "Ace of Hearts");
        ccp.addCard("Spade", "Queen of Spades");
        ccp.findCardsBySymbol("Heart");
    }
}
```

4. **Output-**



```
Card added: Ace of Hearts
Card added: Queen of Spades
Cards with symbol 'Heart': [Ace of Hearts]
```

5. **Learning Outcomes:**

- Understanding Java Collections: Learn how to use `Map`, `List`, and `ArrayList` in Java to store and manage data efficiently.

- Utilizing `computeIfAbsent` Method: Gain practical experience with the `Map` method to streamline adding elements to a collection.

- Implementing Search Functionality: Develop the ability to search and retrieve specific data from a collection using keys (`findCardsBySymbol` method).

- Code Simplification Techniques: Understand how to write clean and concise code by reducing complexity and avoiding redundancy.

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-**

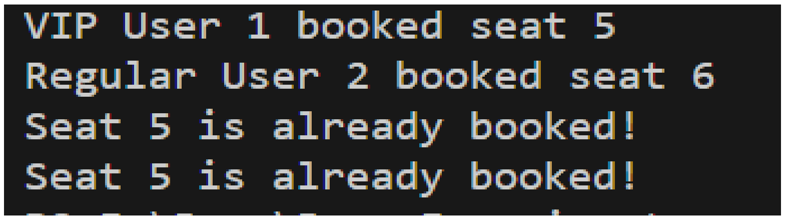
The objective of this program is to create a ticket booking system using synchronized threads to prevent double booking of seats. It uses thread priorities to ensure VIP bookings are processed first, maintaining thread safety and simulating real-world scenarios with multiple booking requests.

3- Code:

```
import java.util.*;

class SeatBookingSystem {
    private final boolean[] seats;

    public SeatBookingSystem(int totalSeats) {
        seats = new boolean[totalSeats];
    }

    public synchronized boolean bookSeat(int seatNumber, String user) {
        if (seatNumber < 0 || seatNumber >= seats.length) {
            System.out.println("Invalid seat number!");
            return false;
        }
        if (!seats[seatNumber]) {
            seats[seatNumber] = true;
            System.out.println(user + " booked seat " + seatNumber);
            return true;
        } else {
            System.out.println("Seat " + seatNumber + " is already booked!");
            return false;
```

```java
        }
    }
}

class BookingThread extends Thread {
    private final SeatBookingSystem system;
    private final int seatNumber;

    public BookingThread(SeatBookingSystem system, int seatNumber, String name,
int priority) {
        super(name);
        this.system = system;
        this.seatNumber = seatNumber;
        setPriority(priority);
    }

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

public class TicketBookingSystem {
    public static void main(String[] args) {
        SeatBookingSystem system = new SeatBookingSystem(10);

        Thread vip1 = new BookingThread(system, 5, "VIP User 1",
Thread.MAX_PRIORITY);
        Thread vip2 = new BookingThread(system, 5, "VIP User 2",
Thread.MAX_PRIORITY);
        Thread user1 = new BookingThread(system, 5, "Regular User 1",
Thread.MIN_PRIORITY);
        Thread user2 = new BookingThread(system, 6, "Regular User 2",
Thread.MIN_PRIORITY);

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

4- Output-



```
VIP User 1 booked seat 5
Regular User 2 booked seat 6
Seat 5 is already booked!
Seat 5 is already booked!
```

5- **Learning Outcomes:**

- **Thread Synchronization:** Understand how to use the synchronized keyword to prevent race conditions and ensure thread safety in multi-threaded applications.
- **Thread Priorities:** Learn how to set and use thread priorities to influence the execution order of threads, simulating scenarios like prioritizing VIP bookings.
- **Avoiding Double Booking:** Gain practical experience in implementing logic to avoid double booking of seats by checking seat availability before confirming the booking.
- **Real-World Simulation:** Develop the ability to model real-world scenarios such as a ticket booking system where multiple users (threads) compete to book the same resources (seats).