WIA/WIB1002 Data Structure

Lab Test 2 (Thursday)

Question 1 (5 marks)

Prepare yourself for a thrilling encounter as Daniel J. D'Arby, a master gambler, challenges the formidable Stardust Crusaders to a game of *Monopoly Deal*! However, to efficiently track the actions and cards played during the game, D'Arby requires a powerful stack data structure. Recognizing your exceptional coding skills, he seeks your assistance in creating a generic stack named **DArbyStack** with a type parameter *DArby* using the `Node` class. The stack should support the following crucial methods:

- `public void push(DArby e)`: allow D'Arby to insert an element onto the top of the stack.
- `public DArby pop()`: allow D'Arby to remove and retrieve the topmost element from the stack, or return `null` if the stack is empty.
- `public DArby peek()`: allow D'Arby to retrieve the element at the top of the stack, or return `null` if the stack is empty.
- `public boolean isEmpty()`: allow D'Arby to check whether the stack contains any elements
- `public int size()`: allow D'Arby to retrieve the current size of the stack.
- `public String toString()`: allow D'Arby to obtain a string representation of all the elements in the stack.
- `public DArby remove(int k)`: allow D'Arby to remove and retrieve the kth most recently inserted element from the stack, where `k` is a positive integer, or return `null` if no such element exists. This method is restricted to using only the `DArbyStack` class to achieve the operation, forbidding the use of any other data structures like `List` and `Queue`.

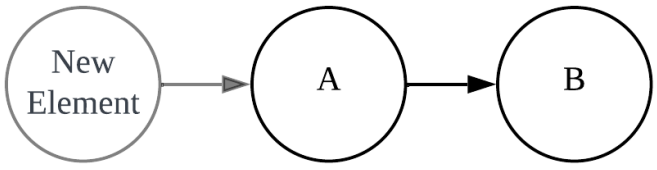

Figure 1. Insertion of an element onto the top of the stack.

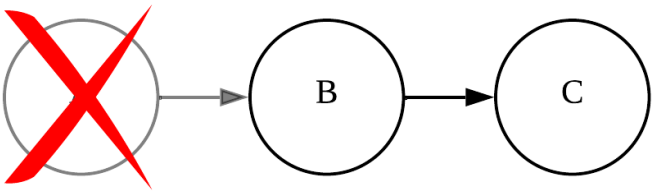

Figure 2. Removal of the topmost element from the stack.

D'Arby has kindly provided visual examples demonstrating the insertion and removal of elements in the stack, which you can find in the image above. Your goal is to complete the given code and ensure that it produces the expected output after running the test program.

Initial Code

```
public class DArbyStack<DArby> {
    private static class Node<E> {
        E item;
        Node<E> next;

        public Node(E item) {
            this.item = item;
        }
    }

    private Node<DArby> head = null;
    private int size = 0;

    public void push(DArby o) {
        // Implement your code here
    }

    public DArby pop() {
        // Implement your code here
    }

    public DArby peek() {
        // Implement your code here
    }
```

```
    public boolean isEmpty() {
        // Implement your code here
    }

    public int size() {
        // Implement your code here
    }

    @Override
    public String toString() {
        // Implement your code here
    }

    public DArby remove(int k) {
        // Implement your code here
    }
}
```

Test Program

```
public class TestDArbyStack {
    public static void main(String[] args) {
        DArbyStack<String> stack = new DArbyStack<>();
        System.out.println(stack);
        stack.push("It's My Birthday");
        stack.push("Pass Go");
        stack.push("Pass Go");
        stack.push("Pass Go");
        stack.push("Debt Collector");
        System.out.println(stack);
        System.out.println();

        for (int i = 0; i < 3; i++)
            System.out.print(stack.pop() + " > ");
        System.out.println();
        System.out.println(stack.remove(1));
        System.out.println(stack);
        System.out.println(stack.size());
        System.out.println();

        stack.push("Forced Deal");
```

```
        stack.push("Deal Breaker");
        stack.push("Just Say No");
        stack.push("Deal Breaker");
        stack.push("Sly Deal");
        stack.push("Debt Collector");
        System.out.println(stack.pop());
        System.out.println(stack.peek());
        System.out.println(stack.remove(7));
        System.out.println(stack.remove(6));
        System.out.println(stack.size());
        System.out.println(stack);
        System.out.println();

        stack.push("Pass Go");
        stack.push("Double The Rent");
        stack.push("Sly Deal");
        System.out.println(stack.peek());
        System.out.println(stack);
        System.out.println(stack.remove(5));
        System.out.println(stack.remove(5));
        System.out.println(stack.size());
        System.out.println();

        while (!stack.isEmpty())
            System.out.print(stack.pop() + " > ");
        System.out.println();
        System.out.println(stack.pop());
        System.out.println(stack.peek());
        System.out.println(stack.size());
        System.out.println(stack);
    }
}
```

Sample Output

```
[]
[It's My Birthday, Pass Go, Pass Go, Pass Go, Debt Collector]

Debt Collector > Pass Go > Pass Go >
Pass Go
[It's My Birthday]
```

```
1

Debt Collector
Sly Deal
null
It's My Birthday
5
[Forced Deal, Deal Breaker, Just Say No, Deal Breaker, Sly Deal]

Sly Deal
[Forced Deal, Deal Breaker, Just Say No, Deal Breaker, Sly Deal, Pass Go,
Double The Rent, Sly Deal]
Deal Breaker
Just Say No
6

Sly Deal > Double The Rent > Pass Go > Sly Deal > Deal Breaker > Forced Deal
>
null
null
0
[]
```

Question 2 (5 marks)

The people of JOJOLands are really excited to visit the famous ∞th residential college! The previous boss of the *Passione*, Diavolo, has been assigned the role of welcoming the guests. However, he doesn't always like the order in which the residents enter the college. Fortunately, the layout of the college allows him to instruct some residents to wait in the foyer while others enter the college normally. In this arrangement, the residents waiting in the foyer form a queue, where only the latest person to enter the foyer is free to leave and proceed into the college.

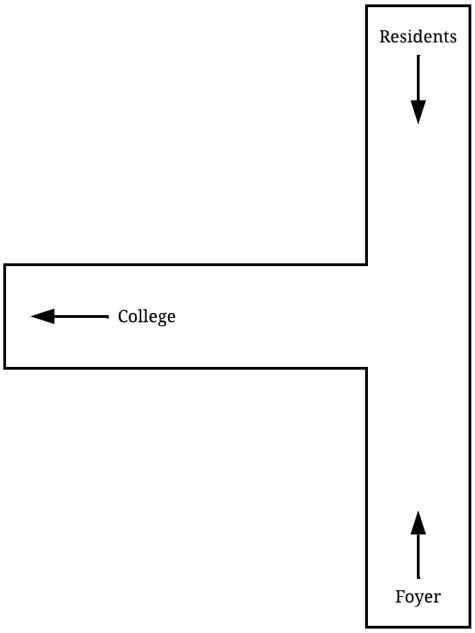

Figure 3. Layout of the ∞th residential college.

Using his Stand called *King Crimson*, Diavolo can erase time and make no one remember what happened during that skipped period. As a result, nobody knows how the final order of residents entering the college is produced! In cases where Diavolo can't achieve his desired order using the standard operation procedure (SOP) aforementioned, he resorts to extraordinary means to manipulate the order.

The principal of the ∞th residential college isn't pleased with this behavior and has asked you to write a method that uses a `Stack` to deduce the actions taken by Diavolo to create such arrangements. You will be given the initial arrival order of the residents and the final order of residents entering the college. The method should take two String arrays representing the orders and return a string describing Diavolo's actions. If Diavolo doesn't follow the SOP, simply return the string `"KING CRIMSON!!!"`. Otherwise, return a string describing Diavolo's actions, with each line structured in one of the following formats:

- `<number> Diavolo instructs <guest name> to enter the college.`
- `<number> Diavolo instructs <guest name> to stay in the foyer.`

If there are multiple possible arrangements, you can return any one of them. Please note that you can only use the methods `push`, `pop`, `peek`, and `isEmpty` provided by `java.util.Stack`, and no other additional data structures like `List`, `Queue`, `PriorityQueue`, etc.

```
import java.util.Stack;

public class KingCrimson {
    public static String getActions(String[] initialOrder, String[]
finalOrder) {
        final Stack<String> stack = new Stack<>();
        // Implement your code here
    }

    public static void main(String[] args) {
        final int N = 8;
        final String[][] initialOrderArray = {
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Jolyne", "Giorno", "Josuke", "Jotaro", "Joseph",
"Jonathan"},
                {"DIO", "Kira", "Pucci"}
        };
        final String[][] finalOrderArray = {
                {"Hermit Purple", "Hierophant Green", "Magician Red",
"Silver Chariot", "Star Platinum"},
                {"Star Platinum", "Silver Chariot", "Hermit Purple",
"Hierophant Green", "Magician Red"},
```

```
                {"Hermit Purple", "Silver Chariot", "Magician Red",
"Hierophant Green", "Star Platinum"},
                {"Magician Red", "Silver Chariot", "Hierophant Green", "Star
Platinum", "Hermit Purple"},
                {"Magician Red", "Silver Chariot", "Hierophant Green",
"Hermit Purple", "Star Platinum"},
                {"Silver Chariot", "Magician Red", "Star Platinum",
"Hierophant Green", "Hermit Purple"},
                {"Jonathan", "Joseph", "Jotaro", "Josuke", "Giorno",
"Jolyne"},
                {"Pucci", "DIO", "Kira"}
        };
        for (int i = 0; i < N; i++) {
            System.out.printf("Case %d\n", i + 1);
            System.out.println(getActions(initialOrderArray[i],
finalOrderArray[i]));
        }
    }
}
```

With correct implementation of the method, you should achieve a similar output provided below.

```
Case 1
1 Diavolo instructs Hermit Purple to enter the college.
2 Diavolo instructs Hierophant Green to enter the college.
3 Diavolo instructs Magician Red to enter the college.
4 Diavolo instructs Silver Chariot to enter the college.
5 Diavolo instructs Star Platinum to enter the college.

Case 2
KING CRIMSON!!!

Case 3
1 Diavolo instructs Hermit Purple to enter the college.
2 Diavolo instructs Hierophant Green to stay in the foyer.
3 Diavolo instructs Magician Red to stay in the foyer.
4 Diavolo instructs Silver Chariot to enter the college.
5 Diavolo instructs Magician Red to enter the college.
6 Diavolo instructs Hierophant Green to enter the college.
7 Diavolo instructs Star Platinum to enter the college.
```

```
Case 4
1 Diavolo instructs Hermit Purple to stay in the foyer.
2 Diavolo instructs Hierophant Green to stay in the foyer.
3 Diavolo instructs Magician Red to enter the college.
4 Diavolo instructs Silver Chariot to enter the college.
5 Diavolo instructs Hierophant Green to enter the college.
6 Diavolo instructs Star Platinum to enter the college.
7 Diavolo instructs Hermit Purple to enter the college.

Case 5
1 Diavolo instructs Hermit Purple to stay in the foyer.
2 Diavolo instructs Hierophant Green to stay in the foyer.
3 Diavolo instructs Magician Red to enter the college.
4 Diavolo instructs Silver Chariot to enter the college.
5 Diavolo instructs Hierophant Green to enter the college.
6 Diavolo instructs Hermit Purple to enter the college.
7 Diavolo instructs Star Platinum to enter the college.

Case 6
1 Diavolo instructs Hermit Purple to stay in the foyer.
2 Diavolo instructs Hierophant Green to stay in the foyer.
3 Diavolo instructs Magician Red to stay in the foyer.
4 Diavolo instructs Silver Chariot to enter the college.
5 Diavolo instructs Magician Red to enter the college.
6 Diavolo instructs Star Platinum to enter the college.
7 Diavolo instructs Hierophant Green to enter the college.
8 Diavolo instructs Hermit Purple to enter the college.

Case 7
1 Diavolo instructs Jolyne to stay in the foyer.
2 Diavolo instructs Giorno to stay in the foyer.
3 Diavolo instructs Josuke to stay in the foyer.
4 Diavolo instructs Jotaro to stay in the foyer.
5 Diavolo instructs Joseph to stay in the foyer.
6 Diavolo instructs Jonathan to enter the college.
7 Diavolo instructs Joseph to enter the college.
8 Diavolo instructs Jotaro to enter the college.
9 Diavolo instructs Josuke to enter the college.
10 Diavolo instructs Giorno to enter the college.
11 Diavolo instructs Jolyne to enter the college.
```

```
Case 8
KING CRIMSON!!!
```

In Case 7, all the residents except Jonathan are instructed to remain in the foyer. Only after Jonathan has entered the college, the residents in the foyer subsequently enter the college one by one, as illustrated in the following figures, resulting in the final order.

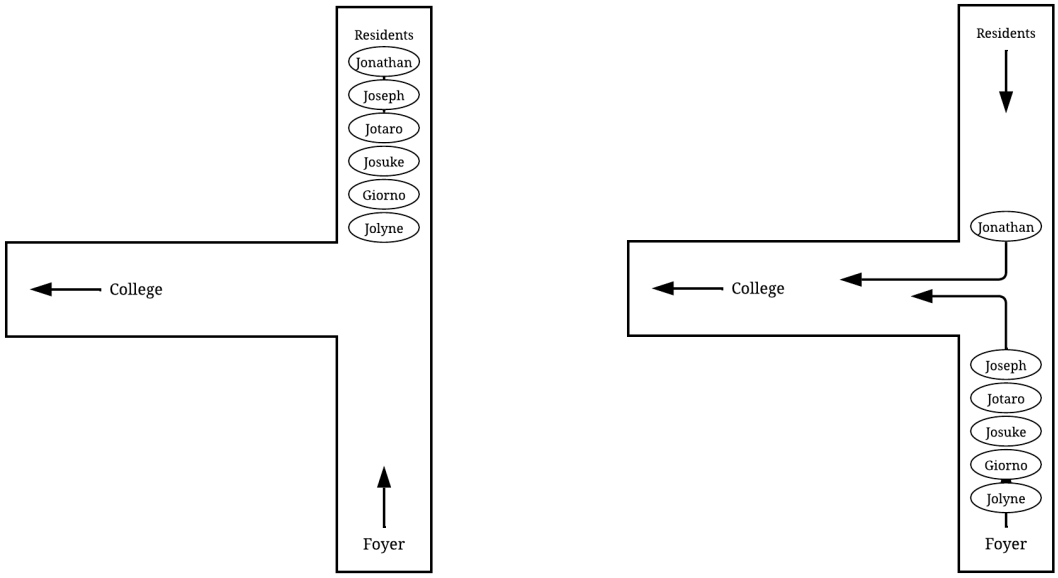

Figure 4 & 5. The order in which the residents enter the college.

Question 3 (5 marks)

Imagine you are a cashier at a busy supermarket. Your task is to process a queue of customers waiting to make their purchases. Suddenly, you spot Chris Martin with his bandmates secretly queuing at your line. As a hardcore fan and a responsible worker, you need to reverse the order of the first **k** customers in the queue while keeping the remaining customers in their original order and ensuring a smooth shopping experience for everyone involved.

In that case, assuming you are given an integer **k** and an array of integers, **arr**. Write a program to reverse the order of the first **k** elements of the array. Using a queue-based solution, you can efficiently handle this situation as such:

- enqueue(x) : Allows you to add customers to the rear of the queue.
- dequeue() : Allows you to remove customers from the front of the queue
- size() : Helps you keep track of the number of customers in the queue
- peek() : Allows you to see the next customer in line
- empty() : Determines whether the queue is empty or still has customers waiting.

Sample Output 1 (k=3):

```
Original Queue: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Reversed Queue: [3, 2, 1, 4, 5, 6, 7, 8, 9, 10]
```

Sample Output 2 (k=4):

```
Original Queue: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Reversed Queue: [4, 3, 2, 1, 5, 6, 7, 8, 9, 10]
```

Sample Output 3 (k=5):

```
Original Queue: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Reversed Queue: [5, 4, 3, 2, 1, 6, 7, 8, 9, 10]
```