WIA/WIB1002 Data Structure

Lab Test 2 (Wednesday)

Question 1 (5 marks)

DIO, the enigmatic and powerful vampire, seeks to ascend to *Heaven* by memorizing fourteen phrases. To aid him in his quest, he requires a clever mechanism to store and manage his precious flashcards. That's where you come in. DIO has enlisted your expertise to craft a generic stack data structure named **DIOStack** with a type parameter known as *DIO*, implemented using an **array**.

Your task is to design and implement **DIOStack** that must possess the following methods:

- `public void push(DIO o)`: allow DIO to insert an element onto the top of the stack, dynamically doubling the size of the stack if it is full before the insertion.
- `public DIO pop()`: allow DIO to remove and retrieve the topmost element from the stack, or return `null` if the stack is empty. If the number of elements in the stack after the removal only reaches a quarter of its array capacity, the stack size should reduce by half.
- `public DIO peek()`: allow DIO to retrieve the element at the top of the stack, or return `null` if the stack is empty.
- `public boolean isEmpty()`: allow DIO to check whether the stack contains any elements
- `public int size()`: allow DIO to retrieve the current size of the stack.
- `public String toString()`: allow DIO to obtain a string representation of all the elements in the stack.
- `private void resize(int size)`: move the stack to a new array of the specified size.

DIO provides the initial code for implementing the **DIOStack**, which you can find below. By successfully accomplishing this task, you should be able to obtain the same output after running the test program below.

Initial Code

```
public class DIOStack<DIO> {
    private DIO[] array = (DIO[]) new Object[1];
    private int size = 0;

    public void push(DIO o) {
```

```
        // Implement your code here
    }

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

    public DIO 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
    }

    private void resize(int size) {
        // Implement your code here
    }
}
```

Test Program

```
public class TestDIOStack {
    public static void main(String[] args) {
        DIOStack<String> stack = new DIOStack<>();
        System.out.println(stack); // the underlying array should have a
length of 1 at this point
        stack.push("Spiral Staircase");
        stack.push("Rhinoceros Beetle");
        stack.push("Desolation Row");
        stack.push("Fig Tart");
        stack.push("Rhinoceros Beetle");
```

```
        System.out.println(stack); // the underlying array should have a
length of 8 at this point
        System.out.println();

        for (int i = 0; i < 4; i++)
            System.out.print(stack.pop() + " > ");
        System.out.println();
        System.out.println(stack); // the underlying array should have a
length of 2 at this point
        System.out.println(stack.size());
        System.out.println();

        stack.push("Via Dolorosa");
        stack.push("Rhinoceros Beetle");
        stack.push("Singularity Point");
        stack.push("Giotto");
        stack.push("Angel");
        stack.push("Hydrangea");
        System.out.println(stack.pop());
        System.out.println(stack.peek());
        System.out.println(stack.size());
        System.out.println(stack); // the underlying array should have a
length of 8 at this point
        System.out.println();

        stack.push("Rhinoceros Beetle");
        stack.push("Singularity Point");
        stack.push("Secret Emperor");
        System.out.println(stack.peek());
        System.out.println(stack); // the underlying array should have a
length of 16 at this point
        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); // the underlying array should have a
length of 2 at this point
    }
}
```

Sample Output

```
[]
[Spiral Staircase, Rhinoceros Beetle, Desolation Row, Fig Tart, Rhinoceros
Beetle]

Rhinoceros Beetle > Fig Tart > Desolation Row > Rhinoceros Beetle >
[Spiral Staircase]
1

Hydrangea
Angel
6
[Spiral Staircase, Via Dolorosa, Rhinoceros Beetle, Singularity Point,
Giotto, Angel]

Secret Emperor
[Spiral Staircase, Via Dolorosa, Rhinoceros Beetle, Singularity Point,
Giotto, Angel, Rhinoceros Beetle, Singularity Point, Secret Emperor]
9

Secret Emperor > Singularity Point > Rhinoceros Beetle > Angel > Giotto >
Singularity Point > Rhinoceros Beetle > Via Dolorosa > Spiral Staircase >
null
null
0
[]
```

Question 2 (5 marks)

Step into the thrilling world of Giorno Giovanna, the current mafia boss of Italy, who finds himself overseeing a bustling parcel collection center. The center has gained immense popularity, attracting an overwhelming influx of parcels. Faced with a shortage of manpower, Giorno harnesses the power of his Stand, *Gold Experience*, to summon a multitude of animals, each tasked with temporarily holding the incoming parcels.

Your responsibility will be to organize the parcels and ensure a seamless retrieval process for their respective owners. As each parcel arrives, it is placed on top of the stack held by a chosen animal. Once all parcels have been accounted for, the owners will arrive to claim their parcels. However, there's a catch - the owners express their dissatisfaction with the need to remove other parcels stacked on top in order to retrieve their own. In other words, they can only access the elements on the top of the stack.

Your mission is to design a method that utilizes the `Stack` class to solve this challenge with the **minimum** number of animals required. The method will receive two lists of strings as input: one representing the arrival order of the parcels labeled with their owners' names, and the other representing the arrival order of the owners themselves. It should return a stack of stacks, where each stack represents the parcels held by a specific animal. If multiple valid arrangements are possible, **any** one of them can be returned. Most importantly, you are only allowed to use the methods provided by the `Stack` and `List` classes **without** utilizing other data structures such as `Queue` or `PriorityQueue`. It is worth mentioning that the owner names are unique identifiers.

```
import java.util.List;
import java.util.Stack;

public class GoldExperience {
    public static Stack<Stack<String>> minimumAnimals(List<String> parcels,
List<String> owners) {
        final Stack<Stack<String>> animals = new Stack<>();
        // Implement your code here
    }

    public static void main(String[] args) {
        final int N = 6;
        final List<List<String>> parcelList = List.of(
```

```
                List.of("Jotaro", "Joseph", "Jolyne", "Jotaro", "Joseph",
"Jolyne", "Jotaro", "Joseph", "Jolyne", "Jotaro", "Joseph", "Jolyne",
"Jotaro", "Joseph", "Jolyne"),
                List.of("DIO"),
                List.of("Babyface", "Beach Boy", "Metallica", "King
Crimson", "Beach Boy", "The Grateful Dead", "Beach Boy"),
                List.of("Whitesnake", "Whitesnake", "Whitesnake",
"Whitesnake", "Made in Heaven", "Made in Heaven", "Made in Heaven", "Made in
Heaven", "C-Moon", "C-Moon", "C-Moon", "C-Moon"),
                List.of("Formaggio", "Formaggio", "Prosciutto", "Illuso",
"Melone", "Pesci", "Formaggio", "Ghiaccio", "Pesci", "Sale"),
                List.of("Koichi", "Hayato", "JoJo", "Iggy", "Giorno",
"Emporio", "Foo", "DIO", "DIO", "DIO", "DIO", "DIO", "Akira", "Bucciarati",
"Akira", "Akira")
        );
        final List<List<String>> ownerList = List.of(
                List.of("Jolyne", "Joseph", "Jotaro"),
                List.of("DIO"),
                List.of("Babyface", "Beach Boy", "King Crimson",
"Metallica", "The Grateful Dead"),
                List.of("C-Moon", "Made in Heaven", "Whitesnake"),
                List.of("Formaggio", "Ghiaccio", "Illuso", "Melone",
"Pesci", "Prosciutto", "Sale"),
                List.of("Akira", "Bucciarati", "DIO", "Emporio", "Foo",
"Giorno", "Hayato", "Iggy", "JoJo", "Koichi")
        );

        for (int i = 0; i < N; i++) {
            Stack<Stack<String>> animals = minimumAnimals(parcelList.get(i),
ownerList.get(i));
            System.out.printf("Case %d\n", i + 1);
            System.out.printf("Number of animals: %d\n", animals.size());
            for (int j = 0; j < animals.size(); j++)
                System.out.printf("Animal %d: %s\n", j + 1, animals.get(j));
            System.out.println();
        }
    }
}
```

Upon successfully implementing and executing the program, you should see a sample output similar to the following:

```
Case 1
Number of animals: 3
Animal 1: [Jotaro, Jotaro, Jotaro, Jotaro, Jotaro]
Animal 2: [Joseph, Joseph, Joseph, Joseph, Joseph]
Animal 3: [Jolyne, Jolyne, Jolyne, Jolyne, Jolyne]

Case 2
Number of animals: 1
Animal 1: [DIO]

Case 3
Number of animals: 4
Animal 1: [Babyface]
Animal 2: [Beach Boy, Beach Boy, Beach Boy]
Animal 3: [Metallica, King Crimson]
Animal 4: [The Grateful Dead]

Case 4
Number of animals: 1
Animal 1: [Whitesnake, Whitesnake, Whitesnake, Whitesnake, Made in Heaven,
Made in Heaven, Made in Heaven, Made in Heaven, C-Moon, C-Moon, C-Moon,
C-Moon]

Case 5
Number of animals: 5
Animal 1: [Formaggio, Formaggio]
Animal 2: [Prosciutto, Illuso, Formaggio]
Animal 3: [Melone, Ghiaccio]
Animal 4: [Pesci, Pesci]
Animal 5: [Sale]

Case 6
Number of animals: 2
Animal 1: [Koichi, Hayato, Giorno, Emporio, DIO, DIO, DIO, DIO, DIO, Akira,
Akira, Akira]
Animal 2: [JoJo, Iggy, Foo, Bucciarati]
```

In Case 3, Metallica's parcel can be positioned beneath King Crimson's parcel within the same stack, since King Crimson will arrive first to claim his parcel. In Case 4, all the parcels can be placed into one single stack. Each owner can retrieve their respective parcels from the top of the stack once all the preceding owners have collected theirs.

Question 3 (5 marks)

Imagine you are a concert organizer responsible for managing a ticketing system for a popular band, Coldplay. As the concert approaches, ticket sales are in full swing, and you want to keep track of the popularity of different ticket categories. As such, you are given an array of ticket sales data, where each element represents the number of tickets sold for a particular category at a specific time. To gain insights into the ticket trends, you decide to use a "sliding window" approach.

Using a sliding window of a certain size, you want to determine the **maximum number** of tickets sold within each window. This information will help you identify peak periods of ticket sales and adjust your event planning and marketing strategies accordingly. By applying a **Priority Queue**-based solution, you can efficiently process the ticket sales data and extract the maximum number of tickets sold within each sliding window. This knowledge will empower you to make data-driven decisions and optimize your ticketing system for a successful and well-attended concert.

In that case, assuming you are given an integer array, **arr,** of size **n,** where a sliding window of size **k** starting from **index 0**. In each iteration, the sliding window moves to the right by one position till **n-k**. Write a program to return an array representing the **maximum number** in all sliding windows.

Note:

- The first element of the resultant array is **max(arr[0...k])**, then the second element is **max(arr[1...k+1])** and so on.
- The size of the resultant array will be **n-k+1**.
- You are expected to solve this question based on **Priority Queue**.

Sample Output 1:

```
Input arr[]: [4, 3, 8, 9, 0, 1], k = 3
Output: [8, 9, 9, 9]
Explanation: The window size is 3 and the maximum at different iterations
are as follows:
max(4, 3, 8) = 8
max(3, 8, 9) = 9
max(8, 9, 0) = 9
max(9, 0, 1) = 9
Hence, we get arr = [8, 9, 9, 9] as output.
```

Sample Output 2:

```
Input arr[]: [9, 8, 6, 4, 3, 1], k = 4
Output: [9, 8, 6]
Explanation: The window size is 4 and the maximum at different iterations
are as follows:
max(9, 8, 6, 4) = 9
max(8, 6, 4, 3) = 8
max(6, 4, 3, 1) = 6
Hence, we get arr = [9, 8, 6] as output.
```

Sample Output 3:

```
Input arr[]: [1, 2, 3, 4, 10, 6, 9, 8, 7, 5], k = 3
Output: [3, 4, 10, 10, 10, 9, 9, 8]
Explanation: The window size is 3 and the maximum at different iterations
are as follows:
max(1, 2, 3) = 3
max(2, 3, 4) = 4
max(3, 4, 10) = 10
max(4, 10, 6) = 10
max(10, 6, 9) = 10
max(6, 9, 8) = 9
max(9, 8, 7) = 9
max(8, 7, 5) = 8
Hence, we get arr = [3, 4, 10, 10, 10, 9, 9, 8] as output.
```