WIA/WIB1002 Data Structure

Lab Test 2 (Friday)

Question 1 (5 marks)

Okuyasu has ingeniously implemented a generic stack known as **TheHandStack**. However, there is a glaring flaw in his implementation: the `pop` and `peek` operations retrieve the bottommost element instead of the desired topmost element. In light of this setback, Okuyasu's loyal friend Josuke takes it upon himself to design a new and improved generic stack called **CrazyDiamondStack**. Determined to honor his friendship, Josuke formulates a brilliant plan: he will utilize two instances of **TheHandStack**, harnessing their combined power to achieve the desired functionality **without** employing any additional data structures.

Your objective is to design and implement **CrazyDiamondStack** with a type parameter of `CD`, which features the following essential methods:

- `public void push(CD element)`: insert an element onto the top of the stack.
- `public CD pop()`: remove and retrieve the topmost element from the stack, or return null if the stack is empty.
- `public CD peek()`: retrieve the element at the top of the stack, or return null if the stack is empty.
- `public boolean isEmpty()`: check whether the stack contains any elements.
- `public int size()`: retrieve the current size of the stack.
- `public String toString()`: obtain a string representation of all the elements in the stack.

Complete the following code snippet, which includes the original **TheHandStack** implementation as well as a skeletal structure for the **CrazyDiamondStack**. Your goal is to implement the program correctly and obtain the expected output provided below.

Initial Code

```
import java.util.ArrayList;

public final class TheHandStack<TH> {
    private final ArrayList<TH> stack = new ArrayList<>();
```

```
    public void push(TH element) {
        stack.add(element);
    }

    public TH pop() {
        return stack.remove(0);
    }

    public TH peek() {
        return stack.get(0);
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }

    public int size() {
        return stack.size();
    }

    @Override
    public String toString() {
        return stack.toString();
    }
}
```

```
public class CrazyDiamondStack<CD> {
    private final TheHandStack<CD> stack1 = new TheHandStack<>();
    private final TheHandStack<CD> stack2 = new TheHandStack<>();

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

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

    public CD 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
    }
}
```

Test Program

```
public class TestCrazyDiamondStack {
    public static void main(String[] args) {
        CrazyDiamondStack<String> stack = new CrazyDiamondStack<>();
        System.out.println(stack);
        stack.push("Cinderella");
        stack.push("Echoes");
        stack.push("Earth Wind and Fire");
        stack.push("The Hand");
        stack.push("Heaven's Door");
        System.out.println(stack);
        System.out.println();

        for (int i = 0; i < 4; i++)
            System.out.print(stack.pop() + " > ");
        System.out.println();
        System.out.println(stack);
        System.out.println(stack.size());
        System.out.println();

        stack.push("Harvest");
        stack.push("The Lock");
        stack.push("Pearl Jam");
        stack.push("Surface");
        stack.push("Love Deluxe");
        stack.push("Highway Star");
        System.out.println(stack.pop());
        System.out.println(stack.peek());
        System.out.println(stack.size());
        System.out.println(stack);
```

```
        System.out.println();

        stack.push("Killer Queen");
        stack.push("Sheer Heart Attack");
        stack.push("Bites the Dust");
        System.out.println(stack.peek());
        System.out.println(stack);
        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

```
[]
[Heaven's Door, The Hand, Earth Wind and Fire, Echoes, Cinderella]

Heaven's Door > The Hand > Earth Wind and Fire > Echoes >
[Cinderella]
1

Highway Star
Love Deluxe
6
[Love Deluxe, Surface, Pearl Jam, The Lock, Harvest, Cinderella]

Bites the Dust
[Bites the Dust, Sheer Heart Attack, Killer Queen, Love Deluxe, Surface,
Pearl Jam, The Lock, Harvest, Cinderella]
9

Bites the Dust > Sheer Heart Attack > Killer Queen > Love Deluxe > Surface >
Pearl Jam > The Lock > Harvest > Cinderella >
```

```
null
null
0
[]
```

Question 2 (5 marks)

Funny Valentine has a peculiar habit of traveling to various parallel dimensions using the ability of his Stand *Dirty Deeds Done Dirt Cheap*. Each dimension he visits is given a unique name by Valentine himself. However, there are times when he forgets whether he is in his original world or in one of these parallel dimensions. To complicate matters further, some dimensions face the imminent threat of destruction by supernatural forces. In such cases, Valentine must quickly retreat to a safe dimension positioned before the dangerous dimension. If Valentine happens to enter the dimension to be destroyed multiple times, he must return to the earliest dimension he entered right before entering that endangered dimension.

To tackle this issue, Funny Valentine has assigned you the task of developing a method using two `Stack`s. This method will receive a `String` array that describes the sequence of actions taken by Valentine. Each string in the array will be in one of the following formats:

- `"Travel <dimension name>"`: Valentine travels to a dimension with the specified name.
- `"Destroy <dimension name>"`: The dimension with the specified name is destined for destruction. Valentine swiftly returns to the safe dimension, as mentioned above.
- `"Ask"`: Valentine queries about the dimension he is currently in, and you need to display the name of that dimension. If he is in his original dimension, you should print `"In Original Dimension"`.

For your implementation, you are only allowed to use the methods `push`, `pop`, `peek`, `isEmpty`, and `clear` provided by the `java.util.Stack`, without employing any additional data structures such as `List`, `Queue`, `PriorityQueue`, etc.

```
import java.util.Stack;

public class DirtyDeedsDoneDirtCheap {
    public static void displayAnswers(String[] actions) {
        final Stack<String> stack1 = new Stack<>();
        final Stack<String> stack2 = new Stack<>();
        // Implement your code here
    }

    public static void main(String[] args) {
        final String[][] actionArray = {
```

```
                {"Travel Three Kingdoms", "Travel Suzume Door", "Travel
JOJOLands", "Ask", "Travel FaceBook", "Travel JOJOLands", "Travel Trading
World", "Ask", "Destroy JOJOLands", "Ask", "Travel FaceBook", "Ask", "Travel
Trading World", "Travel Three Kingdoms", "Ask", "Destroy Three Kingdoms",
"Ask"},
                {"Travel Morioh", "Travel Colosseum", "Travel Cape
Canaveral", "Ask", "Travel Morioh", "Travel Sardinia", "Destroy Morioh",
"Destroy Devil's Palm", "Ask", "Travel Cape Canaveral", "Ask", "Travel
Colosseum", "Destroy Sardinia", "Ask", "Travel Naples", "Destroy Colosseum",
"Ask"},
                {"Ask", "Travel Jade Garden", "Travel Cafe Deux Magots",
"Ask", "Travel Trattoria Trussardi", "Travel Jade Garden", "Travel Cafe Deux
Magots", "Travel Trattoria Trussardi", "Ask", "Destroy Trattoria Trussardi",
"Ask", "Travel Libeccio", "Travel Cafe Deux Magots", "Travel Jade Garden",
"Travel Savage Garden", "Destroy Libeccio", "Ask", "Destroy Savage Garden",
"Destroy Cafe Deux Magots", "Ask"}
        };
        for (int i = 0; i < actionArray.length; i++) {
            System.out.printf("Case %d\n", i + 1);
            displayAnswers(actionArray[i]);
            System.out.println();
        }
    }
}
```

After correctly implementing and executing the program, you should obtain the exact sample output shown below:

```
Case 1
JOJOLands
Trading World
Suzume Door
FaceBook
Three Kingdoms
In Original Dimension

Case 2
Cape Canaveral
In Original Dimension
Cape Canaveral
Colosseum
```

```
Cape Canaveral

Case 3
In Original Dimension
Cafe Deux Magots
Trattoria Trussardi
Cafe Deux Magots
Cafe Deux Magots
Jade Garden
```

Question 3 (5 marks)

Imagine you are embarking on an adventurous road trip with your friends, where you will be exploring various destinations along the way. You have a fixed number of days for your trip and thus giving the time constraint to your trip. As you plan to travel from day one to another, you have a "sliding window" system that helps along with your journey. This system helps you to determine the minimum distance between destinations within each sliding window. In each iteration, the sliding window moves to the next destination, symbolizing your progress along the road trip. Use this program to plan your travel itinerary, discover the shortest routes between destinations, and make the most efficient use of your time and resources during your exciting road trip adventure!

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 **minimum number** in all sliding windows.

Note:

- The first element of the resultant array is **min(arr[0...k])**, then the second element is **min(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: [3, 3, 0, 0]
Explanation: The window size is 3 and the maximum at different iterations
are as follows:
min(4, 3, 8) = 3
min(3, 8, 9) = 3
min(8, 9, 0) = 0
min(9, 0, 1) = 0
Hence, we get arr = [3, 3, 0, 0] as output.
```

Sample Output 2:

```
Input arr[]: [9, 8, 6, 4, 3, 1], k = 4
Output: [4, 3, 1]
Explanation: The window size is 4 and the maximum at different iterations
are as follows:
```

```
min(9, 8, 6, 4) = 4
min(8, 6, 4, 3) = 3
min(6, 4, 3, 1) = 1
Hence, we get arr = [4, 3, 1] as output.
```

Sample Output 3:

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