Day 12

Assignment 6 : Searching for a Sequence in a Stack
Given a stack and a smaller array representing a sequence, write a function that determines if the sequence is present in the stack. Consider the sequence present if, upon popping the elements, all elements of the array appear consecutively in the stack.
A)
To determine if a sequence is present in a stack, you can iterate through the stack and the sequence array simultaneously. Here's a simple explanation of the logic:

Initialization*: Start with two pointers, one for the stack and one for the sequence array, both pointing to the first elements.*

Comparison: *While there are elements in both the stack and the sequence array:*

If the current element of the stack matches the current element of the sequence array, move both pointers to the next elements.

If they don't match, move only the stack pointer to the next element.

Checking for Completion*: If the sequence pointer reaches the end of the array, it means the entire sequence is found consecutively in the stack.*

Return Result: *Return true if the sequence is found, otherwise return false.*

Here's the code implementing this logic in Java:

```
package com.example.stacksorter;
import java.util.Stack;
public class StackSequenceChecker {
            public static boolean isSequenceInStack(Stack<Integer> stack, int[] sequence) {
              int[] reversedSequence = new int[sequence.length];
              for (int i = 0; i < sequence.length; i++) {
                reversedSequence[i] = sequence[sequence.length - 1 - i];
              }
Stack<Integer> stackCopy = (Stack<Integer>) stack.clone();

               int seqIndex = 0;
 while (!stackCopy.isEmpty() && seqIndex < reversedSequence.length) {
                 int top = stackCopy.pop();

                if (top == reversedSequence[seqIndex]) {

                  seqIndex++;
                }
              }
```

```
        return seqIndex == reversedSequence.length;
    }

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);

        int[] sequence1 = {3, 4, 5};
        int[] sequence2 = {2, 3, 4};
        int[] sequence3 = {5, 4, 3};

        System.out.println(isSequenceInStack(stack, sequence1));
        System.out.println(isSequenceInStack(stack, sequence2));
        System.out.println(isSequenceInStack(stack, sequence3));
    }
}
```

Explantion:

1. Reversing the Sequence: The sequence array is reversed because we need to check the elements in the order they would appear if we pop them from the stack.

2. Copying the Stack: We create a copy of the stack using the clone method to avoid modifying the original stack.

3. Iterating Through the Stack: We use a loop to pop elements from the stack copy and compare them with the elements of the reversed sequence.

4. Matching Check: If the popped element matches the current element in the reversed sequence, we move to the next element in the sequence.

5. Final Check: The function returns true if all elements of the sequence have been matched (seqIndex == reversedSequence.length).

This Java implementation correctly checks if a given sequence appears consecutively in the stack upon popping the elements.

output:

```
Sequence 1 Present: true

Sequence 2 Present: false
```