Name : Hardik
Std Id : 202103032

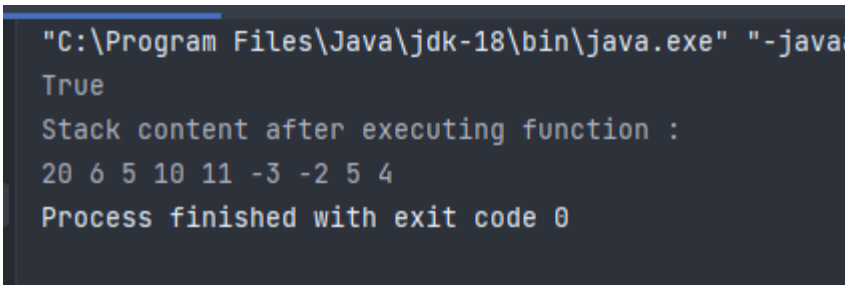
1. Given a stack of integers, how would you check whether each successive pair of numbers in the stack is consecutive or not? The pairs can be increasing or decreasing, and if the stack has an odd number of elements, the elements at the top is left out of a pair. For example, if the stack of elements are [4, 5, -2, -3, 11, 10, 5, 6, 20], then the output should be true because each of the pairs (4, 5), (-2, -3), (11, 10) and (5, 6) consists of consecutive numbers.

Code :

```
package com.DSA.LAB9;

import java.util.*;

class Consecutive {
//check pairwise consecutive in stack

    static boolean pairWiseConsecutive(Stack<Integer> first)
    {
        // Transfer elements of first to second stack.
        Stack<Integer> second = new Stack<Integer> ();

        while (!first.isEmpty()) {
            second.push(first.peek());
            first.pop();
        }

        // Traverse second and check consecutive
        boolean ans = true;

        while (second.size() > 1) {
            //check first two are consecutive in second
            int x = second.peek();
            second.pop();
            int y = second.peek();
            second.pop();

            if (Math.abs(x-y) != 1){
                ans = false;
            }

            //again copy back in first stack
            first.push(x);
            first.push(y);
        }

        if (second.size() == 1){
            first.push(second.peek());
```

```
        }

        return ans;
    }

    public static void main(String[] args)
    {
        Stack<Integer> stack = new Stack<Integer>();
        stack.push(4);
        stack.push(5);
        stack.push(-2);
        stack.push(-3);
        stack.push(11);
        stack.push(10);
        stack.push(5);
        stack.push(6);
        stack.push(20);

        if (pairWiseConsecutive(stack)){
            System.out.println("True");
        }
        else{
            System.out.println("False");
        }

        System.out.println("Stack content after executing function :
");

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

Output :



2. Given an array, print the Next Smaller Element (NSE) for every element of the array. The Next smaller Element for an element x is the first smaller element on the right side of x in the array. Elements for which no smaller element exist, consider the next smaller element as -1. For example, if the array elements are [4, 3, 22, 43, 12, 21, 32, 5], then the Next Smaller elements are : 3, -1, 12, 12, 5, 5, 5, -1.

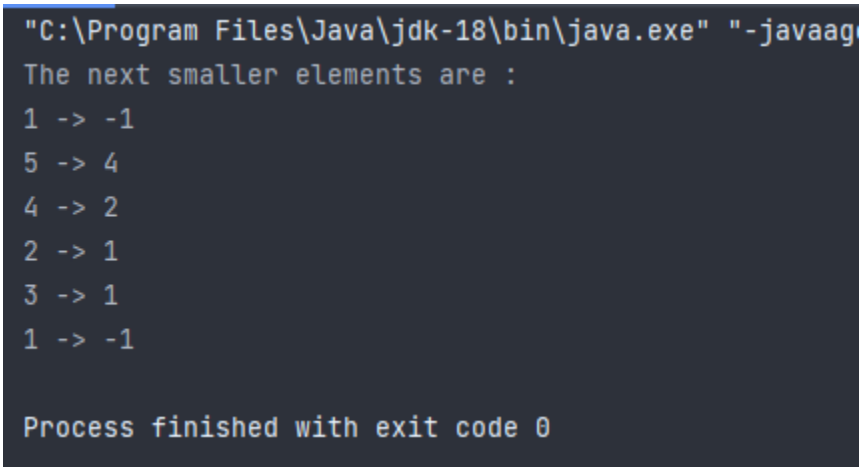
Code :

```
package com.DSA.LAB9;
```

```
import java.util.HashMap;
import java.util.Stack;

public class NSE {

    public static void displayNSE(int[] arr, int n)
    {

        Stack<Integer> stack = new Stack<Integer>();
        HashMap<Integer, Integer> map = new HashMap<Integer,
Integer>();

        // push the first element to stack
        stack.push(arr[0]);

        // iterate for rest of the elements
        for (int i = 1; i < n; i++) {

            if (stack.empty()) {
                stack.push(arr[i]);
                continue;
            }

            // if stack is not empty, then pop and popped element is
greater than next
            while (!stack.empty() && stack.peek() > arr[i]) {
                map.put(stack.peek(), arr[i]); // print the pair
                stack.pop();
            }

         // push next element to stack
            stack.push(arr[i]);
        }

      // for remaining one in stack print -1
        while (!stack.empty()) {
            map.put(stack.peek(), -1);
            stack.pop();
        }
```

```
        for (int i = 0; i < n; i++){
            System.out.println(arr[i] + " -> " + map.get(arr[i]));
        }
    }

    public static void main(String[] args)
    {
        int[] arr = {1, 5, 4, 2, 3, 1};
        System.out.println("The next smaller elements are : ");
        displayNSE(arr, arr.length);
    }
}
```

Output :



3. Given an integer k and a queue of integers, how do you reverse the order of the first k elements of the queue, leaving the other elements in the same relative order? For example, if k=4 and queue has the elements [10, 20, 30, 40, 50, 60, 70, 80, 90]; the output should be [40, 30, 20, 10, 50, 60, 70, 80, 90].

Code :

```
package com.DSA.LAB9;
```

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class Reverse {
    static Queue<Integer> q;

    static void reverseQtillN(int n)
    {
        if (q.isEmpty() || n > q.size() || n <= 0){
```

```
            return;
        }

        Stack<Integer> stack = new Stack<Integer>();

        // Push the first n elements into a Stack
        for (int i = 0; i < n; i++) {
            stack.push(q.peek());
            q.remove();
        }

        // put back stack element into queue
        while (!stack.empty()) {
            q.add(stack.peek());
            stack.pop();
        }

        // Remove the remaining elements and enqueue
        // them at the end of the Queue
        for (int i = 0; i < q.size() - n; i++) {
            q.add(q.peek());
            q.remove();
        }
    }

    // print Queue
    static void display()
    {
        // Queue is not empty
        while (!q.isEmpty()) {
            System.out.print(q.peek() + " ");
            q.remove();
        }
    }

    public static void main(String args[])
    {
        q = new LinkedList<Integer>();
        q.add(10);
        q.add(20);
        q.add(30);
        q.add(40);
        q.add(50);
        q.add(60);
        q.add(70);
        q.add(80);
        q.add(90);
        q.add(100);
        int n = 4;
```

```
        reverseQtillN(n);
        System.out.println("After reverse till "+ n + " element queue
is : ");
        display();
    }
}
```

Output :

```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-jav
After reverse till 4 element queue is :
40 30 20 10 50 60 70 80 90 100
Process finished with exit code 0
```

4. Maximum in sliding window: Given array A[] with sliding window of size w, which is moving from the very left of the array to the very right. Assume that we can only see the w numbers in the window. Each time the sliding window moves rightwards by one position. For example: The array is [1 3 -1 -3 5 3 6 7], and w is 3.

Window Position	Max
[1 3 -1] -3 5 3 6 7	3
1 [3 -1 -3] 5 3 6 7	3
1 3 [-1 -3 5] 3 6 7	5
1 3 -1 [-3 5 3] 6 7	5
1 3 -1 -3 [5 3 6] 7	6
1 3 -1 -3 5 [3 6 7]	7

Code ;

```
package com.DSA.LAB9;
```

```
import java.util.Arrays;
import java.util.Deque;
import java.util.LinkedList;

public class SlidingWindow
{
    static void displayMax(int[] arr, int n, int x)
    {
        // Double ended queue in decreasing order value from front to
```

```
rear
        Deque<Integer> Q = new LinkedList<Integer>();
        int i;
        for (i = 0; i < x; i++)
        {
            // For every element, the previous
            // smaller elements are useless so
            // remove them from q
            while (!Q.isEmpty() && arr[i] >= arr[Q.peekLast()]){
                // Remove from rear
                Q.removeLast();
            }

            // Add new element at rear of queue
            Q.addLast(i);
        }

        // remaining elements from arr[x] to arr[n-1]
        for (; i < n; i++)
        {
            // Front element is the largest element of previous
window
            System.out.print(arr[Q.peek()] + " ");

            // Remove the elements which are out of this window
            while ((!Q.isEmpty()) && Q.peek() <= i - x){
                Q.removeFirst();
            }

            //remove useless elements which are smaller than current
being added element
            while ((!Q.isEmpty()) && arr[i] >= arr[Q.peekLast()])
                Q.removeLast();

            // Add current element at the rear of Q
            Q.addLast(i);
        }

        // Print the maximum element of last window
        System.out.print(arr[Q.peek()]);
    }

    public static void main(String[] args)
    {
        int[] arr = { 1, 3, -1, -3, 5, 3, 6, 7 };

        System.out.println("Array elements are : " +
Arrays.toString(arr));
```

```
        System.out.println("For sliding window max elements are :");
        displayMax(arr, arr.length, 3);
    }
}
```

Output :

```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaag
Array elements are : [1, 3, -1, -3, 5, 3, 6, 7]
For sliding window max elements are :
3 3 5 5 6 7
Process finished with exit code 0
```

5. Perform enqueue and dequeue operations for a double ended queue

Code :

```
package com.DSA.LAB9;
```

```
import java.util.Deque;
import java.util.LinkedList;

public class Dequeue<size> {
    public static void main(String[] args)
    {
        Deque<String> deque = new LinkedList<String>();

        // Add at the last
        // enqueue in Double ended queue
        System.out.println("Enqueue :");
        deque.add("Element 3 (rear)");
        deque.add("Element 4 (rear)");
        deque.addLast("Element 5 (rear)");
        deque.offer("Element 6 (rear)");
        System.out.println(deque + "\n");

        // remove the first element
        // dequeue in Double ended queue
        deque.removeFirst();
        System.out.println("Deque after removing first(dequeue) : "
);
        System.out.println(deque);
    }
}
```

Output :

```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program Files\JetBrai
Enqueue :
[Element 3 (rear), Element 4 (rear), Element 5 (rear), Element 6 (rear)]

Deque after removing first(dequeue) :
[Element 4 (rear), Element 5 (rear), Element 6 (rear)]

Process finished with exit code 0
```

6. Check whether a given string is a palindrome or not, using a queue.

Code :

```
package com.DSA.LAB9;

import java.util.Queue;
import java.util.LinkedList;

public class CheckPalindromeString {

     static void check(String str,Queue queue){

         for (int i = str.length()-1; i >=0; i--) {
             queue.add(str.charAt(i));
         }

         String reverseString = "";
         while (!queue.isEmpty()) {
             reverseString = reverseString + queue.remove();
         }

         if (str.equals(reverseString)){
             System.out.println("palindrome");
         }
         else{
             System.out.println("not a palindrome");
         }
     }

     public static void main(String[] args) {
         String str1 = "nayan";
         Queue queue1 = new LinkedList();
```

```
        System.out.print(str1 + " is ");
         check(str1,queue1);

         String str2 = "forever";
        Queue queue2 = new LinkedList();
        System.out.print(str2 + " is ");
        check(str2,queue2);

    }
}
```

Output :