*“Hackers rarely have full knowledge of the technology* ***stack*** *of a target.”*

* *John McAfee*

Dear readers, hope you are not bored yet. Are you? But this boredom will not last long, because we will now learn a new data structure from today, known as ***Stacks***.

Please refer to this [introductory video](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/stack-usage/video) explaining Stacks data structure to have a better understanding. After watching the lecture, you can go through the article quickly.

Until now, you have heard about the recursive call stack and the function call stack, and know about some of the principles on which stack works. But now, we will discuss in detail how we can create our own stack objects using Java Collections, what are the operations allowed in stack, and find some applications of Stack data structure in real-world use-cases.

* ***Definition***:

Stack is a ***linear data structure*** which follows a ***FILO (First In Last Out) order*** (or **Last In First Out (LIFO)**) for doing various operations.

**Q)** What are linear data structures?

**R)** Data structure where data elements are arranged sequentially or linearly, is called a linear data structure. In linear data structure, we can traverse all the data items in a single pass. Until now, all the data structures you have learnt are linear data structures only, whether it is Array, Arraylist or String. Stack is also a linear data structure.

**Q)** What is *FILO (First In Last Out)* or *LIFO (Last In First Out)* order?

**R)** Let us try to get the meaning of FILO or LIFO using an example. Consider a pile of 3 books on a table.



* If you want to take out the top blue book, then you need not do anything to the pile. Just pick it from the top.
* But if you want to take out the bottom pink book, then first you need to remove the books on top (blue and yellow), and then only you can get access to the third (bottom-most) book.
* Also, if you want to add a book, let's say, green in color, then you can add it above the blue book only. Otherwise, you will have to disturb the remaining 3 books’ ordering.

This type of nature is known as LIFO (Last In First Out), as you can first get the last item added to the stack. Also, you will get the first item inserted in the stack at last (as it becomes the bottom-most element), and this is known as the FILO (First In Last Out) order.

* ***Declaration (In Java)***

Java provides us a stack built with all the operations associated with stack as class member functions.

If we want to declare a stack of Integers in Java, then we need to write the following statement: (Here *st* is the variable name)

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

Please note that the above statement not only declares a stack but also ***initializes*** it with an empty size stack on heap memory.

*Note*: If we had not initialized the stack with the new keyword, then only a reference of the stack would have been created in RAM’s stack and it would have pointed to ***NULL***.

Stack<Integer> st;

* ***Operations***

These are some of the important operations which are associated with stack data structure:

1. ***push(ele)***: Adds an item ele in the stack (to the top of stack).

If the stack is full, then it is said to be a ***Stack Overflow*** condition. Stack overflow can occur when there is no more heap memory, which can be allocated to our stack.

1. ***pop()***: Removes an item from the stack and returns the top element which is removed.

The items are popped in the reverse order in which they are pushed, i.e. the item which is pushed first is removed at last.

If the stack is empty, then it is said to be a ***Stack Underflow*** condition. Hence, trying to remove an element from an empty stack (size = 0), will give a run-time error.

1. ***peek()***: Returns top element of stack.

Note: It will give a run-time error, if there is no element present in the stack, i.e. the size of stack is zero.

1. ***empty()***: Returns true if stack is empty, else false.
2. ***size()***: Returns the number of elements present in the stack.

All these operations: *push, pop, peek, empty* and *size*are ***constant operations***, i.e. the time complexity is ***O(1) per call***.

Hence, if we add n items to the stack (i.e. call push for n times), overall time complexity will be n \* O(1) = O(n), and similarly for any other operation like pop n times from the stack.

Let us take a ***coding exercise*** to learn about the working of these operations. You have to ***predict the output*** of the following code.

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Main

{

public static void main (String[] args) throws java.lang.Exception

{

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

st.push(10);

System.out.println(st + "->" + st.peek() + " " + st.size());

st.push(20);

System.out.println(st + "->" + st.peek() + " " + st.size());

st.push(30);

System.out.println(st + "->" + st.peek() + " " + st.size());

st.push(40);

System.out.println(st + "->" + st.peek() + " " + st.size());

System.out.println("Is Stack Empty: " + st.empty());

st.pop();

System.out.println(st + "->" + st.peek() + " " + st.size());

st.pop();

System.out.println(st + "->" + st.peek() + " " + st.size());

st.pop();

System.out.println(st + "->" + st.peek() + " " + st.size());

st.pop();

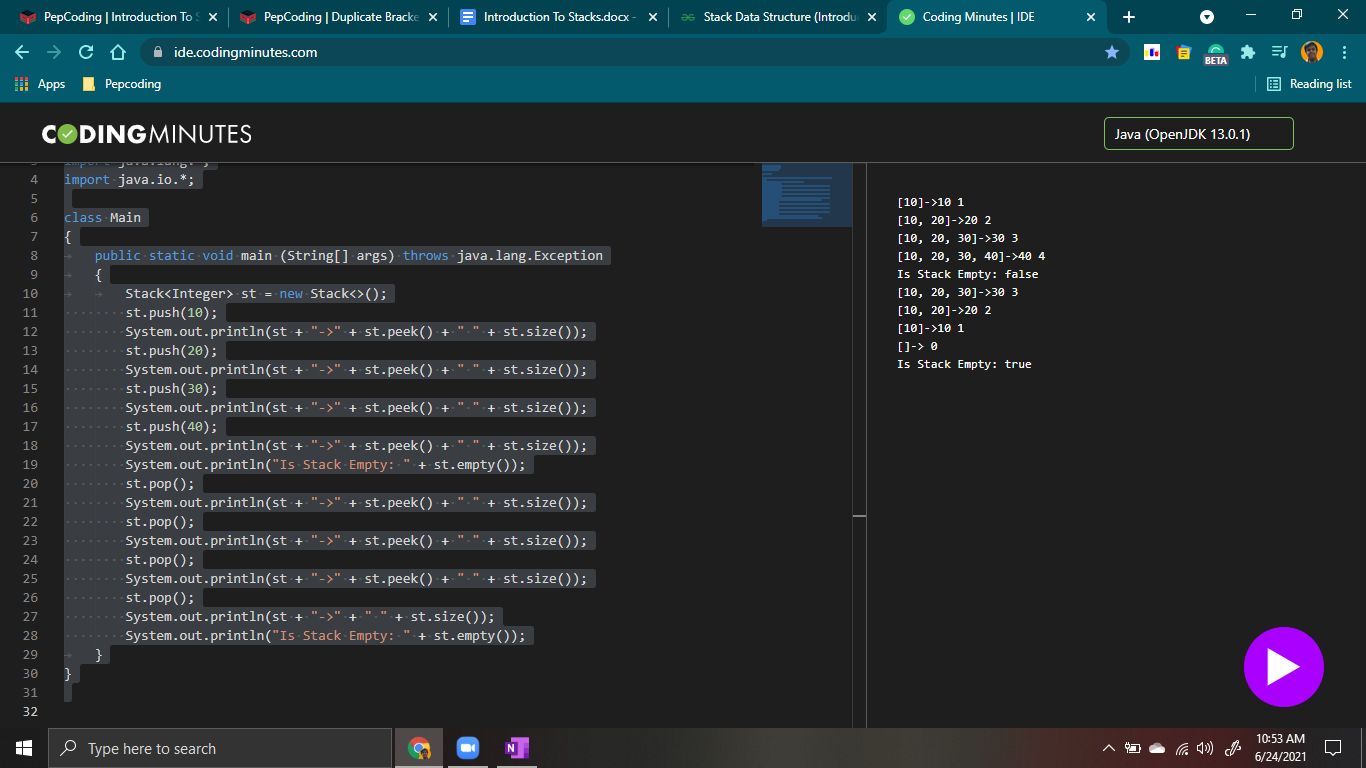
System.out.println(st + "->" + " " + st.size());

System.out.println("Is Stack Empty: " + st.empty());

}

}

Output:



* ***Applications***
  + ***Recursive Call Stack*** & Function Call Stack
  + Check whether an arithmetic expression containing ***parentheses*** is valid or not.
  + ***Infix to Postfix /Prefix*** conversion
  + ***Redo-undo*** features in microsoft word and text editors.
  + ***Tree - Traversals*** (Preorder, Inorder & Postorder)
  + Graph Algorithms like **DFS (Depth-First Search), topological sorting, connected components,** etc.
  + Memory allocation mechanisms in Operating Systems.

In this section, we will apply our knowledge of stacks to solve various problems including some of the applications stated above.

I will see you in the next problem: ‘[Duplicate Brackets](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/duplicate-brackets-official/ojquestion)’ based on Stacks. Before moving on to this problem, try to play with stacks in our *IDE*, explore the basic operations discussed, and have fun!

Contributor : [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)