Implementing Stacks in C++ and java using build-in-class

Stack in C++ STL

The C++ STL offers a built-in class named **stack** for implementing the stack data structure easily and efficiently. This class provides almost all functions needed to perform the standard stack operations like push(), pop(), peek(), remove() etc..

Syntax:

```
stack< data_type > stack_name;

Here,
data_type: This defines the type of data to be
           stored in the stack.
stack_name: This specifies the name of the stack.
```

Some Basic functions of Stack class in C++:

- **empty()** – Returns whether the stack is empty.
- **size()** – Returns the size of the stack.
- **top()** – Returns a reference to the topmost element of the stack.
- **push(g)** – Adds the element 'g' at the top of the stack.
- **pop()** – Deletes the topmost element of the stack.

All of the above functions work in O(1) time complexity.

Implementation:

```cpp
// CPP program to demonstrate working of STL stack
#include <iostream>
#include <stack>
using namespace std;
void showstack(stack <int> s)
{
    while (!s.empty())
    {
        cout << '\t' << s.top();
        s.pop();
    }
    cout << '\n';
}
int main ()
{
    stack <int> s;
    s.push(10);
    s.push(30);
    s.push(20);
    s.push(5);
    s.push(1);
    cout << "The stack is : ";
    showstack(s);
    cout << "\ns.size() : " << s.size();
    cout << "\ns.top() : " << s.top();
    cout << "\ns.pop() : ";
    s.pop();
    showstack(s);
    return 0;
}
```

Output:

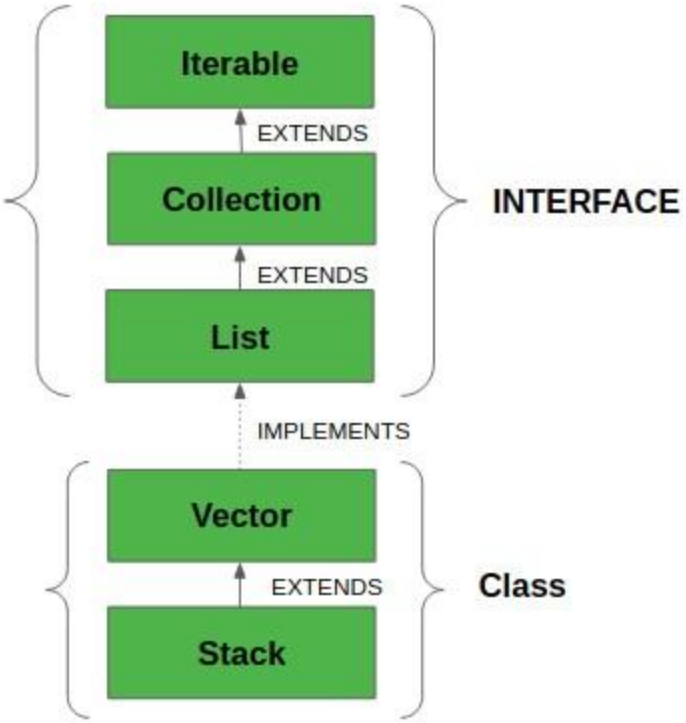
```
The stack is :     1    5    20    30    10

s.size() : 5
s.top() : 1
s.pop() :     5    20    30    10
```

Stack class in Java

Java Collection framework provides a Stack class which models and implements the Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek. The class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector.

This diagram shows the hierarchy of Stack class:



The class supports one *default constructor* **Stack()** which is used to *create an empty stack.*

Below program shows a few basic operations provided by the Stack class:

```
// Java code for stack implementation
import java.io.*;
import java.util.*;
class Test
{
    // Pushing element on the top of the stack
    static void stack_push(Stack<Integer> stack)
    {
        for(int i = 0; i < 5; i++)
        {
            stack.push(i);
        }
    }
    // Popping element from the top of the stack
    static void stack_pop(Stack<Integer> stack)
    {
        System.out.println("Pop :");

        for(int i = 0; i < 5; i++)
        {
            Integer y = (Integer) stack.pop();
            System.out.println(y);
        }
    }
    // Displaying element on the top of the stack
    static void stack_peek(Stack<Integer> stack)
    {
        Integer element = (Integer) stack.peek();
        System.out.println("Element on stack top : " + element);
    }
    // Searching element in the stack
    static void stack_search(Stack<Integer> stack, int element)
    {
        Integer pos = (Integer) stack.search(element);

        if(pos == -1)
            System.out.println("Element not found");
        else
            System.out.println("Element is found at position " + pos);
    }
    public static void main (String[] args)
    {
        Stack<Integer> stack = new Stack<Integer>();
        stack_push(stack);
        stack_pop(stack);
        stack_push(stack);
        stack_peek(stack);
        stack_search(stack, 2);
        stack_search(stack, 6);
    }
}

```

Output:

```
Pop :
4
3
2
1
0
Element on stack top : 4
Element is found at position 3
Element not found
```

Methods in Stack class:

1. **Object push(*Object element*)** : Pushes an element on the top of the stack.
2. **Object pop()** : Removes and returns the top element of the stack. An 'EmptyStackException' exception is thrown if we call pop() when the invoking stack is empty.
3. **Object peek()** : Returns the element on the top of the stack, but does not remove it.
4. **boolean empty()** : It returns true if nothing is on the top of the stack. Else, returns false.
5. **int search(*Object element*)** : It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1.