HackerRank PRACTICE COMPETE JOBS LEADERBOARD Search yves_guduszeit

Stacks

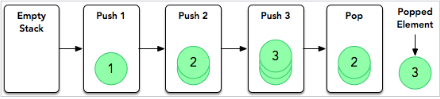
Authored by AllisonP

A stack is a data structure that uses a principle called *Last-In-First-Out* (*LIFO*), meaning that the last object added to the stack must be the first object removed from it.

At minimum, any stack should be able to perform the following three operations:

- *Peek*: Return the object at the top of the stack (without removing it).
- *Push*: Add an object passed as an argument to the top of the stack.
- *Pop*: Remove the object at the top of the stack and return it.

The diagram below demonstrates these simple operations on a stack:



In addition, it's often helpful to implement a method to check whether or not a stack is empty to ensure you are not attempting to perform *peek* or *pop* operations on an empty stack.

Sample Java Implementation

- EXAMPLE

The code below demonstrates a simple Java Stack implementation.

```
import java.util.*;

class Stack<E> {

    private class Element<E> {
        // The data value of the element
        private E data;
        // The next element in the stack
        private Element<E> next;

        Element(E data) {
            this.data = data;
            this.next = null;
        }
    }

    // The element at the top of the stack
    private Element<E> top;

    /** Create an empty stack **/
    public Stack() {
        this.top = null;
    }

    /** @return true if the stack is empty, false if it is not.
    **/
    public boolean isEmpty() {
        return top == null;
    }

    /**
        Pushes a value onto the top of the stack.
        @param value The data for the stack's new top element.
    **/
    public void push(E value) {
        // Create new top element
        Element<E> newTop = new Element<E>(value);

        // If the stack is not empty
        if(!isEmpty()) {
            // Set old top's next variable to point to new top
            newTop.next = top;
        }

        // Set new top regardless of whether or not stack is empty
        this.top = newTop;
    }

    /**
        Remove the element at the top of the stack.
        @return the data associated with the stack's topmost element being remov
        @throws EmptyStackException if the stack contains no elements.
    **/
    public E pop() {
        if(isEmpty()) {
            throw new EmptyStackException();
        }

        Element<E> oldTop = top;
        this.top = top.next;

        return oldTop.data;
    }

    /**
        'View' the element at the top of the stack.
        @return the data associated with the stack's topmost element.
        @throws EmptyStackException if the stack contains no elements.
    **/
    public E peek() {
        if(isEmpty()) {
            throw new EmptyStackException();
        }

        return top.data;
    }
}

class StackDemo {
    public static void main(String[] args) {
        Stack<Integer> intStack = new Stack<Integer>();
```

```java

        try {
            intStack.pop();
        }
        catch(EmptyStackException e) {
            System.out.println("We cannot pop off an empty stack.");
        }

        for(int i = 0; i < 4; i++) {
            intStack.push(i);
            System.out.println("New Top: " + intStack.peek());
        }
        for(int i = 0; i < 4; i++) {
            System.out.println("Popped: " + intStack.pop());
        }

        try {
            intStack.peek();
        }
        catch(EmptyStackException e) {
            System.out.println("We cannot peek at an empty stack.");
        }
    }
}
```

Output

Run