**My Stack**

A ***stack*** is an abstract data type (ADT) that is relatively easy to understand. Think about stacking your lunch trays in the cafeteria – the first person to place their tray on the stack will be at the bottom, and will therefore be the last one "out" of the stack (cleaned by the cafeteria staff).

In this way, a stack of items is a "first in, last out" (FILO) data structure. It is important to note that there is no stack data structure inside your computer (hence the ***abstract*** part of ADT) – a stack is a "wrapper", built around an already implemented list object (e.g. an array).

1. Create a **Runner.java** class with a main method and a class called **StackProbs.java** that will define the methods below. Make a StackProbs object in the Runner class to test the methods. A helper method that will return a new java.util.Stack given an array of integers might be useful for testing.
2. Complete the method void doubleUp(Stack<Integer> nums) that will replace every value in the stack with two of the same value (in the same order).

(1, 3, 5, 0, -1) >>> (1, 1, 3, 3, 5, 5, 0, 0, -1, -1)

1. Complete the method void posAndNeg(Stack<Integer> nums) that will split nums, such that all the negative numbers will be at the bottom of the stack, and all non-negative numbers will be at the top of the stack. The actual order of the elements is irrelevant if the negative and non-negative numbers are properly split.

(2, 9, -4, 3, -1, 0, -6) >>> (-4, -1, -6, 2, 9, 3, 0)

1. Complete the method void swapPairs(Stack<Integer> nums) that will "swap" every element in nums, starting at the bottom of the stack. The last element will not be moved for stacks with an odd number of elements.

(1, 2, 3, 4, 5) >>> (2, 1, 4, 3, 5)

1. Complete the method void shiftByN(Stack<Integer> nums, int n) that "shifts" n values from the bottom of the stack to the top of the stack. You can assume n will be a valid value (greater than 0, and not larger than the number of elements in the stack).

(7, 23, -7, 0, 22, -8, 4, 5), 3 >>> (0, 22, -8, 4, 5, 7, 23, -7)

1. Complete the method String reverseVowels(String str) that will reverse all the vowels (a, e, i, o, u) in a String. You must use a Stack in your solution. Hint: there is an implicit cast when using compound operators (e.g. +=); you can concatenate a char or a Character to a String.

reverseVowels("hello how are you") >>> " hullo hew aro yoe"

1. (Riddle) 2 = L in the HB
2. Complete the method boolean bracketBalance(String s) that will return true if the brackets in the String parameter are "balanced". A String's brackets are balanced if there are always opening and closing brackets, in the right order, of each type of bracket. For this problem, we will only consider round () and square [] type brackets.

bracketBalance("(([()])))") >>> false

bracketBalance("([()[]()])()") >>> true

//A better version of this method would work for *any* symbols pairs, maybe using a HashMap

**My Stack**

Write a class **MyStack.java** that implements a stack data structure, using an array to store (only) Integer objects. MyStack should have the following:

|  |  |
| --- | --- |
| Integer[] stack | array "backing" the stack abstract data structure (stores actual objects) |
| int size | stores the current size of the stack and is used as an index variable for remembering the index of the top of the stack |
| MyStack() and MyStack(int initCap) | default and parameterized constructor (default and user-specified initial array capacity), chained with a call to this() |
| boolean isEmpty() | returns true if this stack is currently empty |
| Integer peek() | returns the object at the top of this stack without removing it from the stack. This method should throw a new EmptyStackException if the user tries to peek at an element from an empty stack |
| Integer pop() | removes and returns the object at the top of this stack. This method should throw a new EmptyStackException if the user tries to pop an element from an empty stack |
| void push(Integer item) | pushes an item onto the top of this stack. Should invoke (call) the private doubleCapacity() method if necessary |
| void doubleCapacity() | *<private>* this helper method should double the size of the backing array |
| String toString() | *<overridden>* shows the state of the stack (in a stack-like way) |

A runner class with a main method has been provided. Your output should match the output in the file **"output.txt"**. Test this class thoroughly; you will be using it in future projects.

**(Advanced) Constant time minimum**

Add a method Integer getMin() that returns the minimum value in the stack in constant (i.e. O(1)) time. You may use additional storage, but returning the minimum value in the stack should require no iteration (or calling methods that perform iterative searches for you, e.g. contains or indexOf).

All methods (push, pop, and getMin) should operate in constant time (i.e. you can't do iterative searches when pushing and popping).