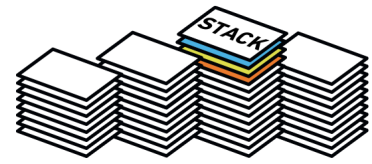# My Stack

Write a class **MyStack.java** that implements a stack data structure, using an array to store (only) <u>Integer</u> objects. <u>MyStack</u> 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 <u>EmptyStackException</u> 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 <u>EmptyStackException</u> 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).