

## PA2: Stack and ArrayStack

Assignment	Write a <code>Stack</code> interface and <code>ArrayStack</code> class in the package <code>ku.util</code> .
What to Submit	<ol style="list-style-type: none"><li>1. Create an empty repository on Github Classroom using this link: <a href="https://classroom.github.com/a/16iWXFAP">https://classroom.github.com/a/16iWXFAP</a></li><li>2. Clone the repository. Add your own README.md and .gitignore.</li><li>3. Submit your source code in the "src/ku/util" directory. Push to Github.</li></ol>

The Java API has a `Stack` interface, but some of the method names are *inconsistent* with other collections, and it has a `search` method which a `Stack` should not have. So, we will define our own `Stack` type.

### 1. Stack Interface

1.1 Define a `Stack` interface in the package `ku.util`, with the methods shown below. The `Stack` interface has a type parameter (`T`) so that it can be used to hold any kind of data we want.

The `Stack` methods are

<code>int capacity( )</code>	the maximum number of elements that this <code>Stack</code> can hold. Return -1 if unknown or infinite.
<code>boolean isEmpty( )</code>	true if stack is empty.
<code>boolean isFull( )</code>	true if stack is full.
<code>T peek( )</code>	return the item on the top of the stack, without removing it. If the stack is empty, return <code>null</code> .
<code>T pop( )</code>	return the item on the top of the stack, and remove it from the stack. Throws: <code>java.util.EmptyStackException</code> if stack is empty.
<code>void push( T obj )</code>	push a new item onto the top of the stack. If the stack is already full, this method does nothing. It is the programmer's responsibility to check <code>isFull()</code> before trying to push something onto the stack.  The parameter ( <code>obj</code> ) must not be null.  Throws: <code>IllegalArgumentException</code> if parameter is null.
<code>int size( )</code>	return the number of items in the stack. Returns 0 if the stack is empty.

We want the stack to be able to hold elements of any kind, so define the class with a type parameter (`T`), like this:

```
package ku.util;

//TODO Interface must have very good Javadoc for everything

public interface Stack<T> {

    public T pop();

    ...
}
```

1.2 Write good Javadoc comments for the interface and every method!

The first sentence of each Javadoc comment should be a complete sentence. Write sentences that describe what the interface or method does. Look at the Javadoc for JDK's `Stack` class for examples. *Don't* begin sentences with "This method does..." -- just write what it does.

## 2. ArrayStack Class

Write an `ArrayStack` class in package `ku.util` that *implements* the `Stack` interface and uses an array to hold the stack elements.

2.2 `ArrayStack` has a type parameter, just like the `Stack` interface:

```
public class ArrayStack<T> implements Stack<T> {
```

2.2 Write a public constructor that specifies the capacity of the stack:

<b><code>ArrayStack(int capacity)</code></b>	create a new stack with the given capacity, which is the maximum number of elements that the stack can hold. Capacity must be zero or positive. A capacity of zero is legal, even though its useless
--	--

2.3 `ArrayStack` uses an array to store elements on the stack.

To create an array of references using a type parameter (T), use code like this:

```
items = (T[]) new Object[capacity];
```

Java doesn't allow creating instances using a type parameter, so we create an array of `Object` references and *cast* them to an array of T.

2.4 Thoroughly test your `ArrayStack`. Test cases which should fail as well as cases that should succeed.

### Example using BlueJ Interactive Mode

```
> import ku.util.*;
// a stack with capacity 2
> Stack<String> stack = new ArrayStack<String>(2);
> stack.isEmpty()
true
> stack.size()
0
> stack.push("cake");
> stack.push("ice cream");
> stack.size( )
2
> stack.isFull( )
true
> stack.push("yogurt");      // discarded - stack is already full
> stack.pop( )
"ice cream"
> stack.size( )
1
> stack.peek( )
"cake"
> stack.size( )
1                          // still 1, peek() doesn't remove anything
> stack.pop( )
"cake"
> stack.pop( )              // error - stack is empty
java.util.EmptyStackException thrown
> stack.peek( )
null
```