# Stack Testing

| | |
|---|---|
| Assignment | Create JUnit tests of the Stack interface. |
| Files | Add these files to your testing project:<br>`ku/util/Stack.java` - interface for Stack. Write tests for this.<br>`StackFactory.jar` - creates stack instances. See #3 below. |

## 1. Stack Interface

The **`Stack`** interface in the package **`ku.util`**, with the methods shown below. A Stack has a type parameter (**T**) which determines the type of objects you can put on the stack.

The Stack methods are

| | |
|---|---|
| **`int capacity( )`** | return the maximum number of objects the stack can hold |
| **`boolean isEmpty( )`** | true if the stack is empty, false otherwise. |
| **`boolean isFull( )`** | true if stack is full, false otherwise. |
| **`T peek( )`** | return the item on the top of the stack, without removing it. If the stack is empty, return **`null`**. |
| **`T pop( )`** | Return the item on the top of the stack, and remove it from the stack.<br>Throws: `java.util.EmptyStackException` if stack is empty. |
| **`void push( T obj )`** | push a new item onto the top of the stack. If the stack is already full, this method is ... its the programmer's responsibility to check isFull() before trying to push something onto stack.<br>The parameter (obj) must not be null.<br>Throws: `InvalidArgumentException` if parameter is null. |
| **`int size( )`** | return the number of items in the stack. Returns 0 if the stack is empty. |

## 2. Create a Dummy Stack

Since you don't have any real stack implementation yet, create a Dummy stack class that implements the Stack interface. Eclipse will auto-generate "do nothing" method code.

In your JUnit tests, use the dummy stack whenever you need to create a Stack.

Your tests will fail. This is normal and expected.

## 3. Test a Real Stack

After you've written some tests, download StackFactory.jar and add it to your project.

StackFactory.jar contains a single "factory" class ku.util.StackFactory. It has a makeStack(capacity) method that creates an untyped Stack. Use a cast if you want a type-safe Stack. Examples:

```
Stack stack = StackFactory.makeStack( 5 );  // stack with capacity 5
Stack<String> stack2 = (Stack<String>) StackFactory.makeStack( 3 );
```

4. Multiple Stack Types

StackFactory can product different types of Stack.

## Writing Good Tests

Your objective is to discover as many errors as possible. There are several kinds of cases or "zones" you should try to test:

- valid case with no ambiguity
- borderline case that should be valid,: Stack of size 1 or pushing 4 items on a stack of cap. 4
- borderline case that should be invalid, e.g. pushing 5 items onto a stack of capacity 4.
- extreme cases. Cases so weird or unusual the programmer didn't think of them.

## Example using BlueJ Interactive Mode

```
> import ku.util.*;
> Stack<String> stack = new Stack<String>(2); // pretty small
> 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.pop( )
"cake"
> stack.pop( )
java.util.EmptyStackException thrown
> stack.peek( )
null
```