**StackSolution**

Basic observations

Nuget packages

NUnit + Adapter (required by VS2019)

Moq (legacy use Rhino, but moving towards Moq)

FluentAssertions (Generally use Assert.IsTrue etc but I like FluentAssertions. Will include both in test examples)

NCrunch… you may or may not get a license

<https://www.freecodecamp.org/news/tdd-explanation-hands-on-practice-with-c-a0124338be44/>

We will use C# to write a Stack implementation.

**So our requirements are pretty simple: we want to implement a Stack class, so the requirements are:**

1. **Limit the size of the stack.**
2. **Add element. (push)**
3. **Remove element. (pop)**
4. **Check what was the last element. (peek)**
5. **Get the current size of the stack.**
6. **Have a class that can accept any data type.**
7. **When the customer exceeds the size of the Stack, we need to throw an appropriate exception.**

After we know what the requirements of the system are, we can start to define how we will solve this. We’ll implement it using an array.

We will start writing out test units in the StackTests project under StackTest class.

Before we can start writing the code we need to learn 3 important things: TestFixture, Test, and Assertions.

**Pattern Arrange, Act, Assert**

TestFixture Attribute on the class

Test Attribute on a specific test

Setup/Teardown – can be per test or per test run

**Test 1. Creation test**. We will write a creation test, which will create a new object of our Stack, and it will check that the size of the Stack is 0 at the beginning.

Fails. Write enough code to make the test pass.

We defined this class (Stack) to be implemented as an array, therefore we will define the member field as an array of type T.

We require to pass the maximum length of the stack at the constructor, therefore we will create a constructor that takes a size argument.

And since we require that we receive the current size of the stack at any point, we’ll define a property of “Size”. Of course, none will be able to change the size, therefore it will be a private set.

Run tests

Now we should refactor our code — but at this point, we don’t really have anything to refactor, so we will move forward.

**Push & Pop test**

Push will take an argument and add it to the top of the stack.

Pop will remove the element from the stack and return it.

We’ll add 3 elements to the stack, then we will take the last element out. At this point we will check that the last element is the exact one we expect to get and that the stack size decreased.

Push and pop functions

Let’s run our tests again, and boom, everything works perfectly! All the tests passed successfully.

**Error exceeding the allowed size**

We want to throw custom exceptions when we:

Push new element when the stack is full.

Pop element when there are no elements in the stack.

As you can see, we need to create two new custom exceptions.

ExpenditureProhibitedException — This exception will occur when the stack is empty and the client attempts to pop a new element out.

ExceededSizeException — This exception will occur when the stack is full and the client attempts to add a new element to the stack.

Go to Stack Project and create a new class called CustomExceptions. In this class we will define our new exceptions and they will inherit from the Exception class.

~~Custom exceptions~~

~~Modify our current push and pop functionality to throw an exception when needed.~~

~~Custom exceptions~~

~~So now, as part of the TDD lifecycle, we’re running the tests… and Hooray! All tests have passed successfully.~~

Peek the last element

We’re about to finish with the last tests. We want to peek the last element in the stack. If the stack is empty, we will throw an ExpenditureProhibitedException exception, otherwise, we will return the last element.

**Let’s create our test cases.**

Attempt to peek the element when the stack is empty. In this test we will throw a custom exception.

Insert some elements in the stack, then peek an element, make sure this is the correct element, and check that the array size hasn’t changed.

**Injection**

StringReverser – implement without injection first, then with. Will have to add a interface for the Stack class