CST236 – Assignment 1

Wilsonville, Spring 2020

# Goal

Create a simple collections library with full unit tests, written in MSTest for C# .NET Framework.

Collections:

1. StackUp (stack using linked nodes)
2. QueueUp (queue using linked nodes)

Steps 1 through 6 walk you through implemeting and testing StackUp

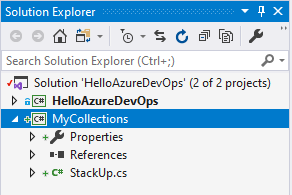
Step 7 challenges you to implement QueueUp

Step 8 describes what to turn in to submit this assignment

# Steps

## Step 1: Create a simple stack implementation to test

1. Create a new class library MyCollections
2. Add a new class called StackUp



1. Paste the following code into your StackUp.cs file to implement the stack…

## StackUp class before debugging

using System;

namespace MyCollections

{

public class StackEmptyException : Exception

{

public StackEmptyException() : base("Stack empty!") { }

}

public class StackUp<T>

{

private class Node

{

public T item;

public Node next;

}

private Node head;

private int count;

public StackUp()

{

head = null;

count = 0;

}

public int Count { get { return count; } }

public void Push(T item)

{

head = new Node() { item = item, next = null };

count++;

}

public T Pop()

{

if (head == null)

throw new StackEmptyException();

T item = head.item;

head = head.next;

return item;

}

public T Peek()

{

return head.item;

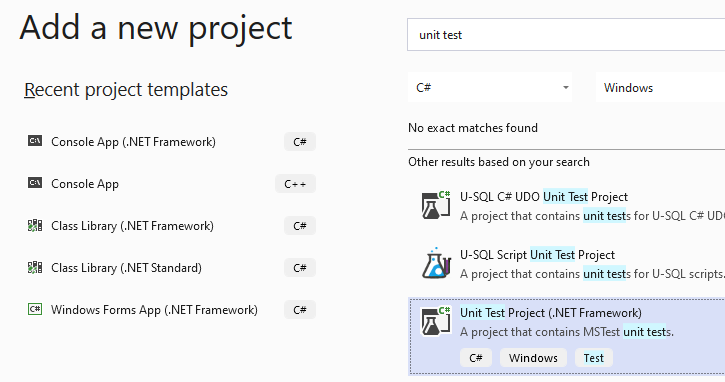
}

}

}

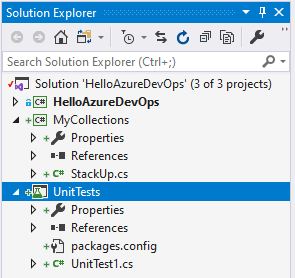
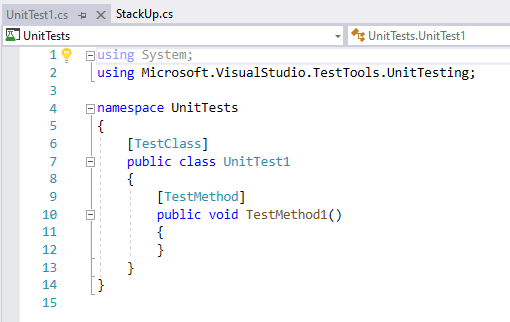
## Step 2: Add a MSTest Project for C# to your solution

1. Right-click on your solution and choose Add 🡪 New Project…
2. Search for “unit test” template
3. Select “Unit Test Project (.NET Framework)” for C#



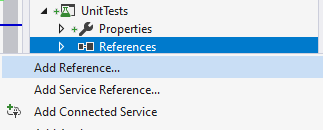
1. Click Next
2. Name your new project UnitTests and click Create

The new project is added to your solution with a default UnitTest1 class

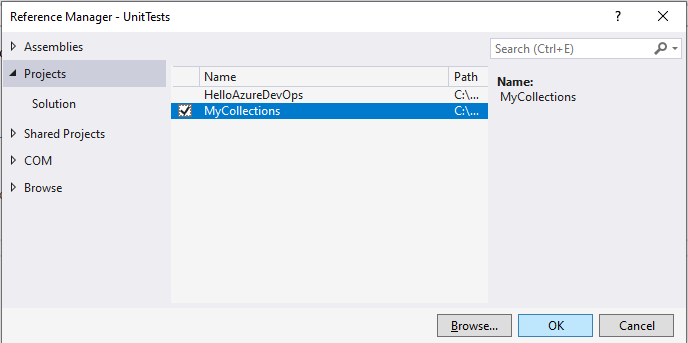
 

## Step 3: Connect UnitTests to MyCollections library

1. Right-click on References in the UnitTests project and choose Add Reference…



1. Select Projects in the tree on the left
2. Check MyCollections and click OK



## Step 4: Write and run a first unit test

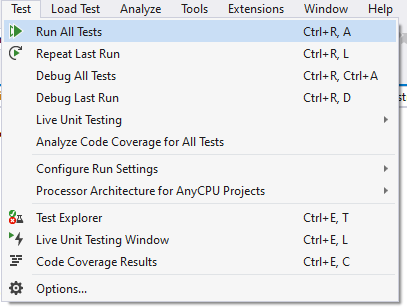
1. Rename UnitTest1.cs class to StackUpUT.cs
2. Add a using statement for MyCollections to the top of StackUpUT.cs
3. Rename TestMethod1() to TestConstructor()
4. Add some code to instantiate a StackUp and verify its Count property is 0 (zero)

[TestMethod]

public void TestConstructor()

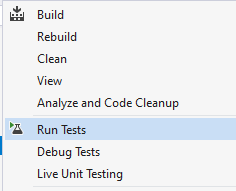
{

StackUp<string> stack = new StackUp<string>();

 Assert.AreEqual(stack.Count, 0);

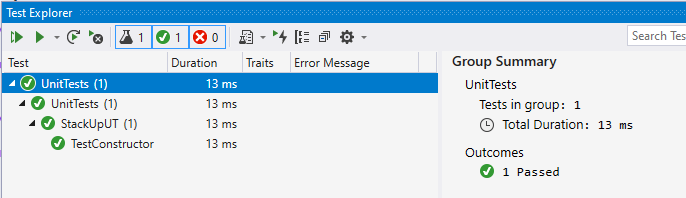
}

1. Select Run All Tests… from the Test menu



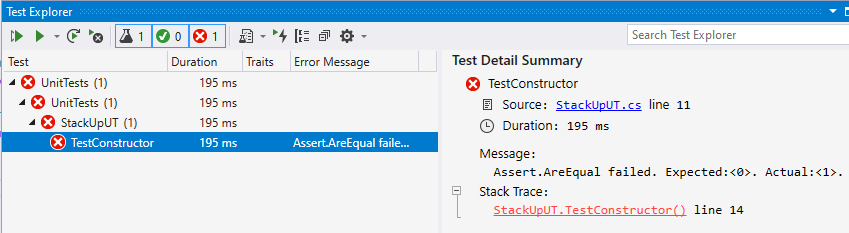
1. Or, right-click on the UnitTests project and select Run Tests

The test should run and pass…



1. To see an example failure, change the expected value to ‘1’ and re-run the tests

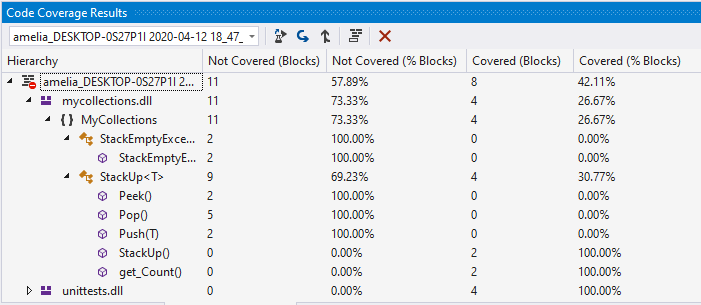
The test should run and fail…



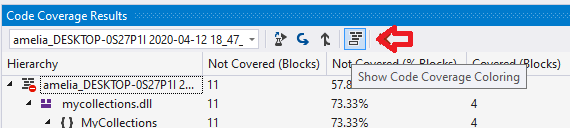
1. Click the red error under Stack Trace: and it will select the test line that failed in your unit test
2. Make sure to fix the test so it expects 0 (zero) again!

## Step 5: Use code coverage to focus further testing

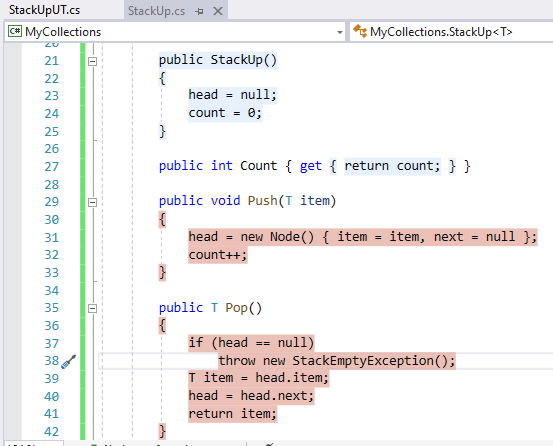
1. Select Analyze Code Coverage for All Tests from the Test menu
2. Review the report in the Code Coverage Results pane



1. Click the Show Code Coverage Coloring icon



The StackUp.cs file now shows code highlighted as green (covered) or red (not-covered)…



## Step 6: Add more tests and make them pass!

1. Add a new test to StackUpUT.cs
2. Run all tests and fix any issues until all tests pass
3. Analyze code coverage
4. Rinse and repeat 1-3, until coverage is “good enough”

Did you find and fix all the bugs? There are 3 of them. Don’t give up!

When you think you’re done, compare your code and tests with the following…

## StackUp class after debugging

using System;

namespace MyCollections

{

public class StackEmptyException : Exception

{

public StackEmptyException() : base("Stack empty!") { }

}

public class StackUp<T>

{

private class Node

{

public T item;

public Node next;

}

private Node head;

private int count;

public StackUp()

{

head = null;

count = 0;

}

public int Count { get { return count; } }

public void Push(T item)

{

head = new Node() { item = item, next = head };

count++;

}

public T Pop()

{

if (head == null) throw new StackEmptyException();

T item = head.item;

head = head.next;

count--;

return item;

}

public T Peek()

{

if (head == null) throw new StackEmptyException();

return head.item;

}

}

}

## Full Unit Tests for StackUp

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using MyCollections;

namespace UnitTests

{

[TestClass]

public class StackUpUT

{

[TestMethod]

public void TestConstructor()

{

StackUp<string> stack = new StackUp<string>();

Assert.AreEqual(stack.Count, 0);

}

[TestMethod]

public void TestPush()

{

StackUp<string> stack = new StackUp<string>();

stack.Push("one");

Assert.AreEqual(stack.Count, 1);

stack.Push("two");

Assert.AreEqual(stack.Count, 2);

stack.Push("three");

Assert.AreEqual(stack.Count, 3);

}

[TestMethod]

public void TestPop()

{

StackUp<string> stack = new StackUp<string>();

stack.Push("one");

stack.Push("two");

stack.Push("three");

Assert.AreEqual(stack.Pop(), "three");

Assert.AreEqual(stack.Count, 2);

Assert.AreEqual(stack.Pop(), "two");

Assert.AreEqual(stack.Count, 1);

Assert.AreEqual(stack.Pop(), "one");

Assert.AreEqual(stack.Count, 0);

}

[TestMethod]

public void TestPeek()

{

StackUp<string> stack = new StackUp<string>();

stack.Push("one");

Assert.AreEqual(stack.Peek(), "one");

stack.Push("two");

Assert.AreEqual(stack.Peek(), "two");

stack.Push("three");

Assert.AreEqual(stack.Peek(), "three");

stack.Pop();

Assert.AreEqual(stack.Peek(), "two");

stack.Pop();

Assert.AreEqual(stack.Peek(), "one");

}

[TestMethod]

[ExpectedException(typeof(StackEmptyException))]

public void TestPopEmpty()

{

StackUp<string> stack = new StackUp<string>();

stack.Pop();

}

[TestMethod]

[ExpectedException(typeof(StackEmptyException))]

public void TestPeekEmpty()

{

StackUp<string> stack = new StackUp<string>();

stack.Peek();

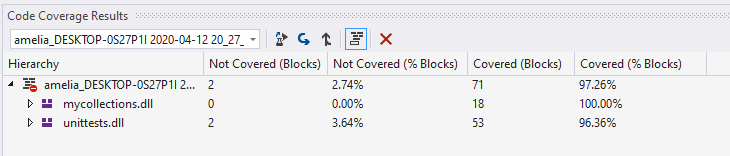
}

}

}

## Full Test Coverage for StackUp

After fixing the defects and running all tests, 100% of the StackUp code blocks are covered. However not all of the unit test code is reached.



This reflects the fact that two of the unit tests throw exceptions when Pop() or Peek() are called on an empty stack. The exception is expected, and there is no way to test these conditions without the exception. So, this is fine! The test results and coverage are great!

## Step 7: Create and fully test a queue collection using a TDD approach

1. Add a new class to your library project called QueueUp, matching the following interface

namespace MyCollections

{

public class QueueEmptyException : Exception

{

public QueueEmptyException() : base("Queue empty!") { }

}

public class QueueUp<T>

{

// instantiates an empty queue

public QueueUp();

// returns the number of items in the queue

public int Count { get; }

// inserts an item at the tail

public void Insert(T item);

// removes an item from the head

// throws QueueEmptyException

public T Remove();

// returns the item at the head, without removing it

// throws QueueEmptyException

public T Peek();

}

}

1. Stub out your class so it can be instantiated and it compiles
2. Add a new test class to the unit test project
3. Write complete unit tests for your queue collection
4. Run the tests and verify that they all fail
5. Implement your queue using linked Node classes, simpliar to StackUp
6. As you implement, keep running your unit tests, until they pass
7. Use the technique from Step 6 to ensure good coverage

## Step 8: Turn in your work

1. Create a zip file of your code, including solution, projects and source
2. Name the zip file CST236\_Assign1\_<your name>.zip, where <your name> is really your name
3. Upload your zip file on Canvas to submit your assignment

Note: The instructor should be able to unzip, build the solution, run the unit tests and verify code coverage.