.net core TDD

Test-Driven Development

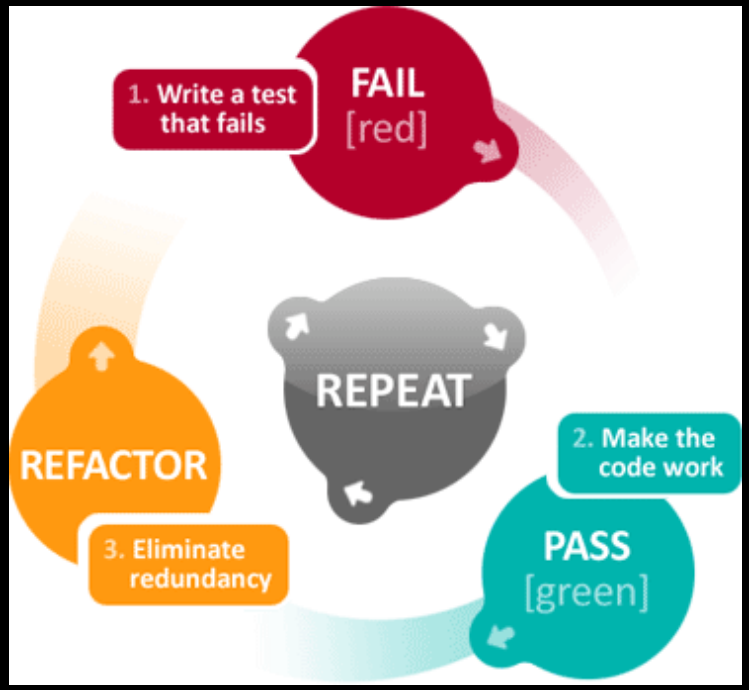
- Test-driven development (TDD) is a software development process that relies on the **repetition** of a **very short development cycle**: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements.
- American software engineer **Kent Beck**, who is credited with having developed or "rediscovered" the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.
- Test-driven development is related to the **test-first** programming concepts of extreme programming, begun in 1999, but more recently has created more general interest in its own right.
- Programmers also apply the concept to improving and debugging legacy code developed with older techniques.

The Three Laws of TDD

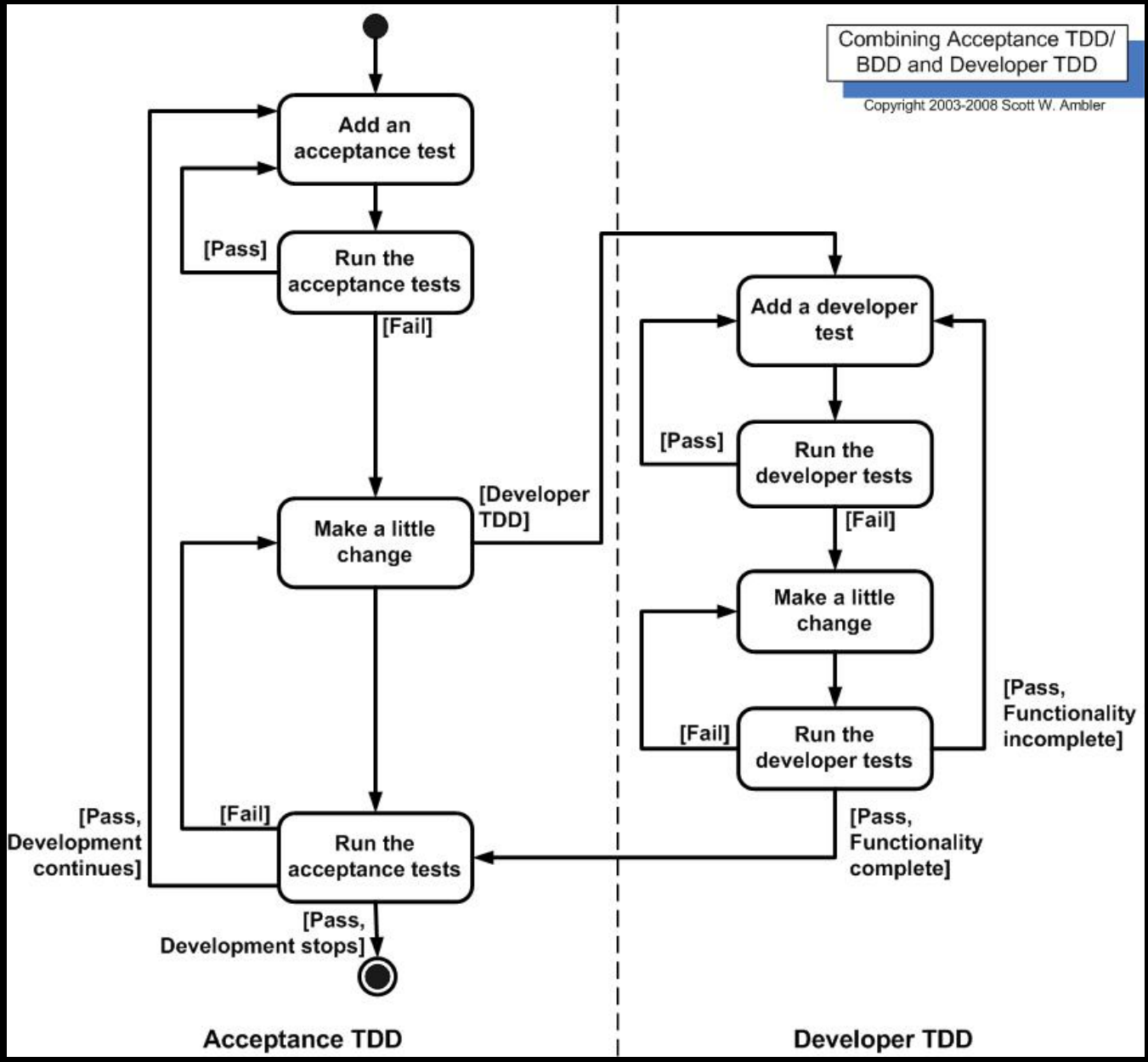
1. No writing production code until you have written failing unit test.

2. No writing more of a unit test than is sufficient to fail.

3. No writing more production code than is sufficient to pass the currently failing test.



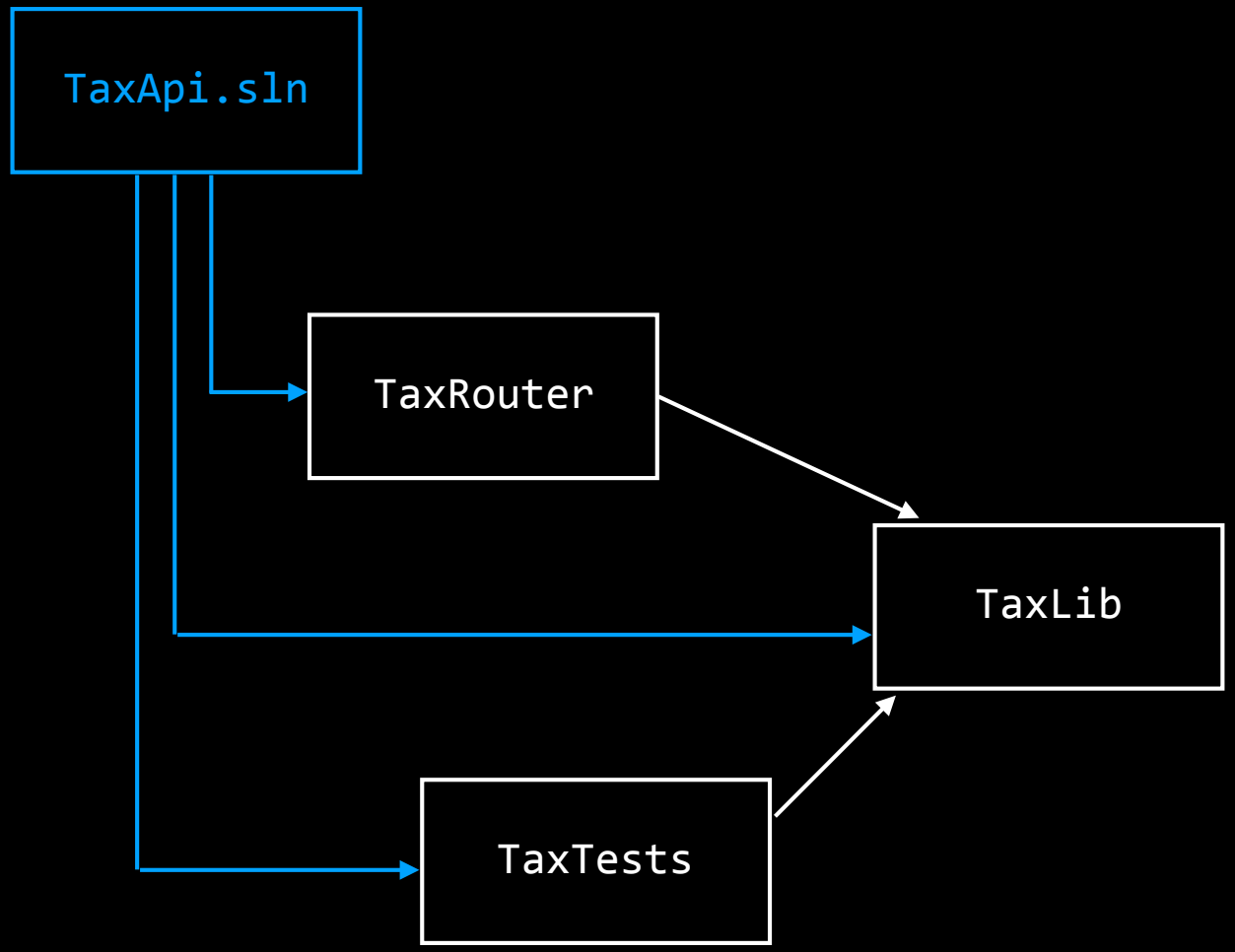
TDD Cycle


Combining Acceptance TDD/ BDD and Developer TDD
Copyright 2003-2008 Scott W. Ambler
Add an acceptance test
[Pass]
Run the acceptance tests
[Fail]
Make a little change
[Developer TDD]
[Pass, Development continues]
[Fail]
Run the acceptance tests
[Pass, Development stops]
Add a developer test
[Pass]
Run the developer tests
[Fail]
Make a little change
[Fail]
Run the developer tests
[Pass, Functionality incomplete]
[Pass, Functionality complete]
Acceptance TDD
Developer TDD

Unit Test Should be...

- Small.
- Run very fast, from millisecond to seconds.
- Independent from other unit tests.
- Stand alone, no boundary systems dependency.
- Repeatable, Regress-able.

Project Structure



Classes

- **VatController**
- **VatManager**

Pre-requisite

- .net core 2.1 or above
- VS Code
- .net core Test Explorer for VSCode https://goo.gl/gfofVf

Initial Project

Initial ASP.net Web API project

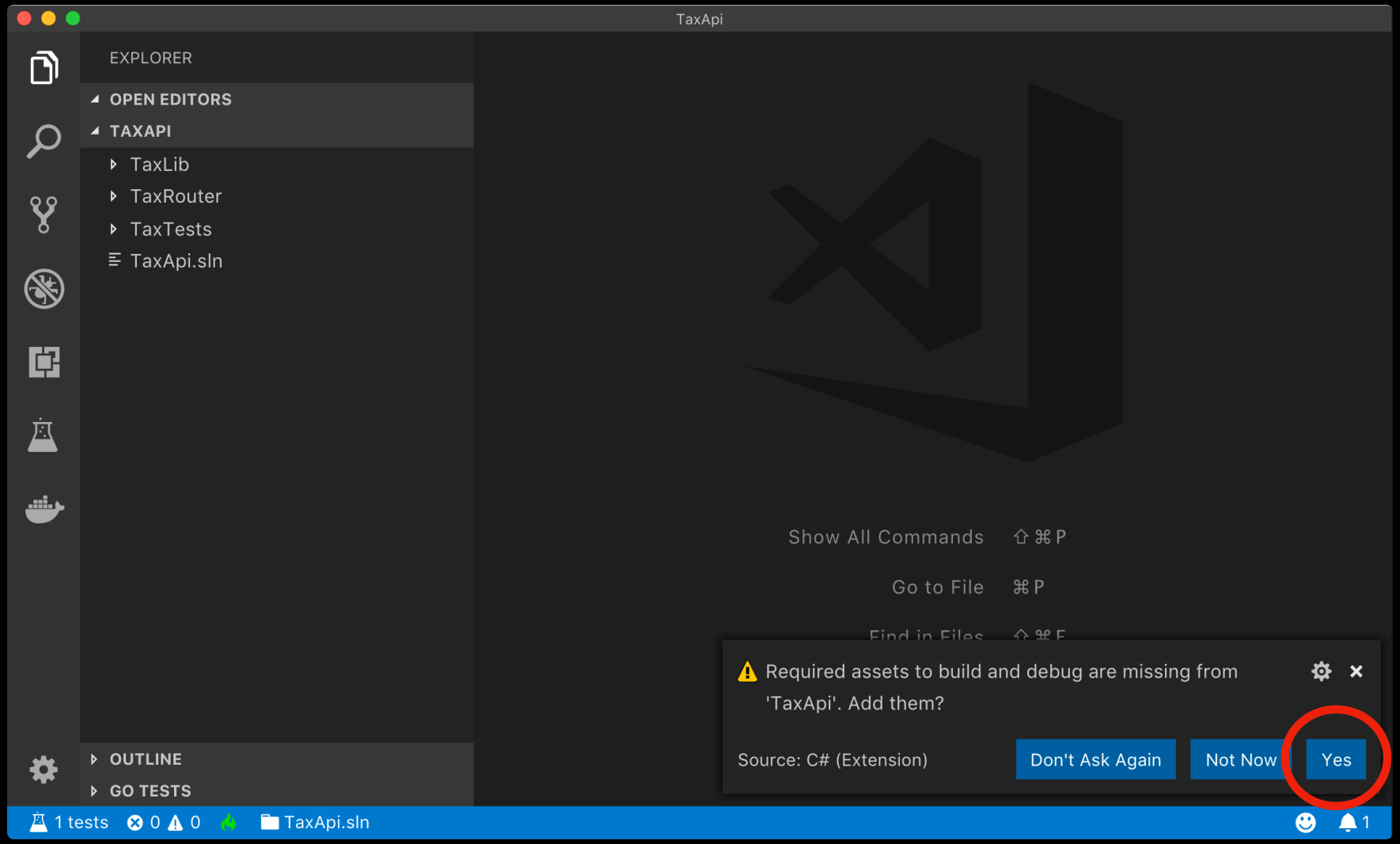
Initial Class Lib

Initial XUnit Project

Initial Solution

```
Initial solution
  $ dotnet new sln -o TaxApi
  $ cd TaxApi
Initial ASP.net web API project
  $ dotnet new webapi -o TaxRouter
Initial class library
  $ dotnet new classlib -o TaxLib
Initial xUnit project
  $ dotnet new xunit -o TaxTests
Add projects to solution
  $ dotnet sln add ./TaxRouter/TaxRouter.csproj
  $ dotnet sln add ./TaxLib/TaxLib.csproj
  $ dotnet sln add ./TaxTests/TaxTests.csproj
Add TodoLib to TodoRouter’s class references
  $ cd TaxRouter
  $ dotnet add reference ../TaxLib/TaxLib.csproj
add TodoLib to TodoTests’s class references
  $ cd ../TaxTests
  $ dotnet add reference ../TaxLib/TaxLib.csproj
Build project, Test Project
  $ cd ..
  $ dotnet build
  $ dotnet test ./TaxTests/
Open VS code
  $ code .
```

Install Assets for project



Test Cases

1. Calculate 7% vat rate for 100 baht should return 107.
2. Change vat rate from 7% to 10%, input 100 should return 110.
3. Calculate exclude vat 7% rate from 107 should return 100.
4. Change vat rate from 7% to 10%, exclude from 110 should return 100.

- Rename UnitTest1.cs to VatCalculatorTests.cs
- Rename UnitTest class to VatCalculatorTests
- Rename Test1() to Calculate_7_percents_vat_rate_for_100_baht_should_return_107()

VatCalculatorTests.cs

```
using System;
using Xunit;

namespace TaxTests
{
    0 references | Run All Tests | Debug All Tests
    public class VatCalculatorTests
    {
        [Fact]
        0 references | Run Test | Debug Test
        public void Calculate_7_percents_vat_rate_for_100_baht_should_return_107()
        {
            // Arrange

            // Act

            // Assert
        }
    }
}
```

```
public void Calculate_7_percents_vat_rate_for_100_baht_should_return_107()
{
    // Arrange
    decimal amount = 100;
    decimal vatRate = 7;
    decimal expectedTotalAmount = 107;
    decimal actualTotalAmount = 0;

    // Act

    // Assert
    Assert.Equal(expectedTotalAmount, actualTotalAmount);
}
```

- TaxLib
  - Rename Class1.cs to VatCalculator.cs
  - Rename Class1 to VatCalculator

```
using System;
using Xunit;

using TaxLib;

namespace TaxTests
{
    public class VatCalculatorTests
    {
        [Fact]
        public void Calculate_7_percents_vat_rate_for_100_baht_should_return_107()
        {
            // Arrange
            decimal amount = 100;
            decimal vatRate = 7;
            decimal expectedTotalAmount = 107;
            decimal actualTotalAmount = 0;

            // Act
            var vatCalculator = new VatCalculator();
            actualTotalAmount = vatCalculator.calculateVat(amount, vatRate);  // <- ERROR

            // Assert
            Assert.Equal(expectedTotalAmount, actualTotalAmount);
        }
    }
}
```

```
using System;

namespace TaxLib
{
    public class VatCalculator
    {
        public decimal calculateVat(decimal amount, decimal vatRate)
        {
            return amount + (amount * vatRate / 100);
        }
    }
}
```

- Run Test.
- Calculate 7% vat rate for 100 baht should return 107 passed!
- Next, implement change vat rate from 7% to 10%, input 100 should return 110.

Refactor with Theory & InlineData

```
// Arrange
[Theory]
[InlineData(100, 7, 107)]
[InlineData(100, 10, 110)]
public void CalculateVatWithVatRateShouldReturnVatIncluded(
        decimal amount,
        decimal vatRate,
        decimal expectedTotalAmount)
{
    // Act
    var vatCalculator = new VatCalculator();
    decimal actualTotalAmount = vatCalculator.calculateVat(amount, vatRate);

    // Assert
    Assert.Equal(expectedTotalAmount, actualTotalAmount);
}
```

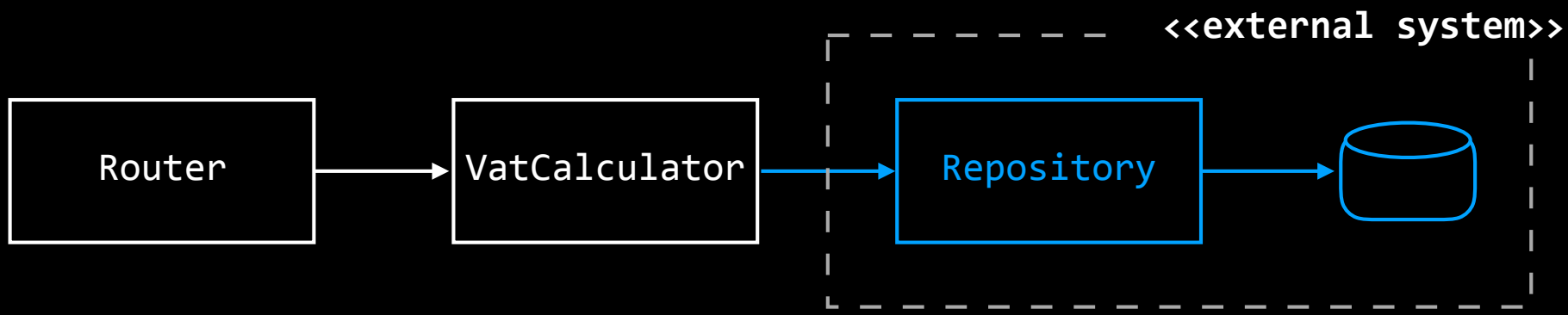
- Run Test.
- Remove old tests

Test exclude vat.

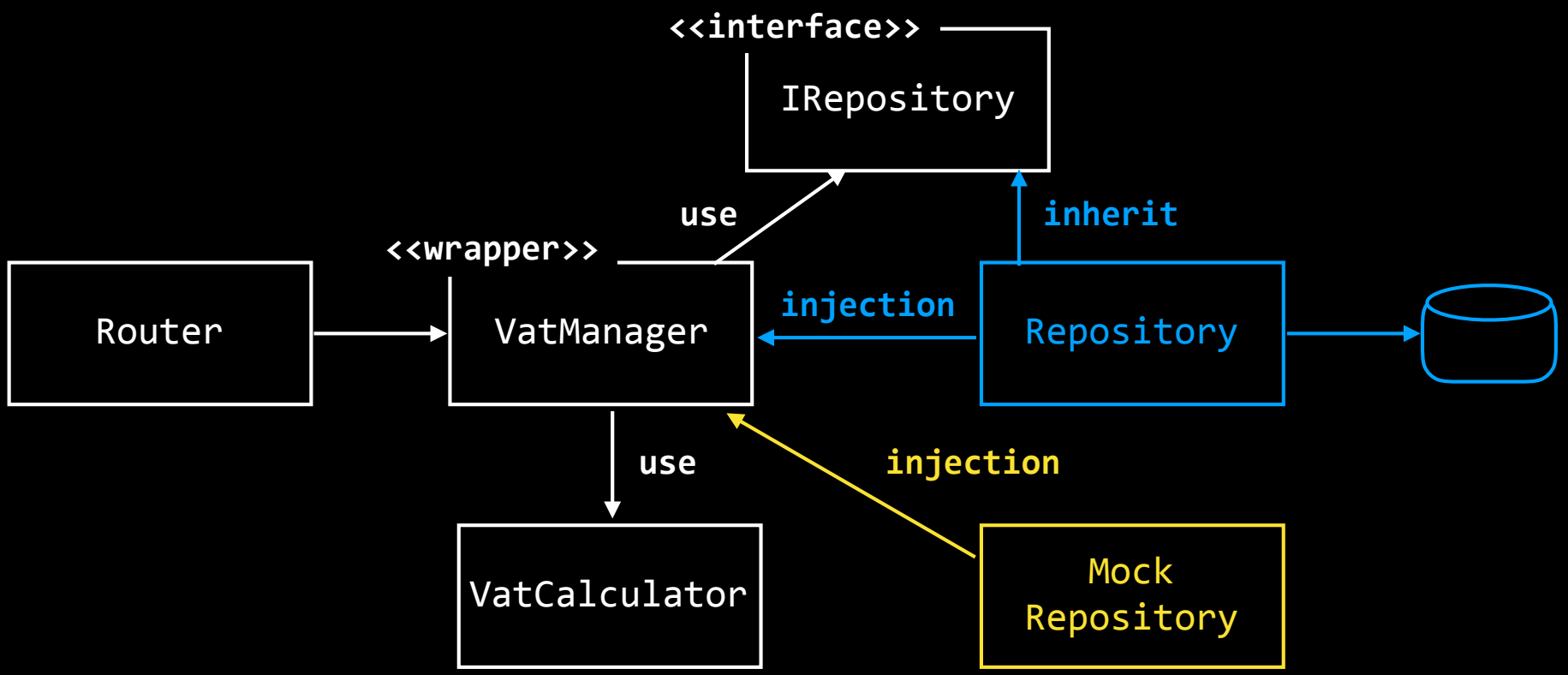
```
[Theory]
[InlineData(107, 7, 100)]
[InlineData(110, 10, 100)]
public void CalculateExcludeVatShouldReturnAmount(
        decimal totalAmount,
        decimal vatRate,
        decimal expectedAmount)
{
    // Act
    var vatCalculator = new VatCalculator();
    decimal actualAmount = vatCalculator.excludeVat(totalAmount, vatRate);  // <- ERROR

    // Assert
    Assert.Equal(expectedAmount, actualAmount);
}


public decimal excludeVat(decimal totalAmount, decimal vatRate)
{
    return totalAmount / (1 + (vatRate / 100));
}
```

- Run Test.

Dependency Injection and Mocking Framework



Dependency Injection and Mocking Framework



Moq

- Add Moq

```
$ cd TaxTests/
$ dotnet add package moq
```

- Example :-

```
using System;
using Xunit;
using TaxLib;
using Moq;

namespace TaxTests
{
    public interface IRepo
    {
        int get();
    }

    public class VatCalculatorWithRepoTests
    {
        [Fact]
        public void testMoq() {
            var moq = new Mock<IRepo>();
            moq.Setup(r => r.get()).Returns(100);

            var obj = moq.Object;
            var result = obj.get();

            Assert.Equal(100, result);
        }

    }
}
```

- New Test file named VatCalculatorWithRepoTests.cs

```
using System;
using Xunit;
using Moq;
using TaxLib;

namespace TaxTests
{
    public class VatCalculatorWithRepoTests {

        [Theory]
        [InlineData(100, 107)]
        [InlineData(100, 110)]
        public void TexController_should_return_Total_amount_with_tax_rate_from_repo(
            decimal amount, decimal vatRate, decimal expectedTotalAmount)
        {
            // Arrange
            var vatCalculator = new VatCalculator();
            var vatManager = new VatManager(repo, vatCalculator);  // <- Error

            // Act
            var actualTotalAmount = vatManager.calculateVat(amount); // <- Error

            // Assert
            Assert.Equal(expectedTotalAmount, actualTotalAmount);
        }
    }
}
```

- Create VatManager class

```
using TaxLib;

namespace TaxLib
{
    public class VatManager
    {
        private object repo;  // <- Change object to IRepository

        private VatCalculator vatCalculator;

        public VatManager(object repo, VatCalculator vatCalculator) // <- Change object to IRepository
        {
            this.repo = repo;
            this.vatCalculator = vatCalculator;
        }
    }
}
```

- Create IRepository interface class (IRepository.cs)

```
namespace TaxLib
{
    public interface IRepository
    {
        decimal getVatRate();
    }
}
```

- Update VatManager.cs

```
using System;
using TaxLib;

namespace TaxLib
{
    public class VatManager
    {
        private IRepository repo;
        private VatCalculator vatCalculator;
        public VatManager(IRepository repo, VatCalculator vatCalculator)
        {
            this.repo = repo;
            this.vatCalculator = vatCalculator;
        }

        public decimal calculateVat(decimal amount)
        {
            var vatRate = repo.getVatRate();
            return vatCalculator.calculateVat(amount, vatRate);
        }

    }
}
```

- Update VatCalculatorWithRepoTests.cs

```
using System;
using Xunit;
using Moq;
using TaxLib;

namespace TaxTests
{
    public class VatCalculatorWithRepoTests {

        [Theory]
        [InlineData(100, 107)]
        [InlineData(100, 110)]
        public void TexController_should_return_Total_amount_with_tax_rate_from_repo(
            decimal amount, decimal vatRate, decimal expectedTotalAmount)
        {
            // Arrange
            var vatCalculator = new VatCalculator();

            var mock = new Mock<IRepository>();
            mock.Setup(r => r.getVatRate()).Returns(vatRate);
            var repo = mock.Object;

            var vatManager = new VatManager(repo, vatCalculator);

            // Act
            var actualTotalAmount = vatManager.calculateVat(amount); // <- Error

            // Assert
            Assert.Equal(expectedTotalAmount, actualTotalAmount);
        }
    }
}
```

Run Test!!!

- Lab
  - Add Exclude Vat with Repository Test.

- Verify

```
namespace TaxTests
{
    public class VatCalculatorWithRepoTests {

        [Theory]
        [InlineData(100, 107)]
        [InlineData(100, 110)]
        public void TexController_should_return_Total_amount_with_tax_rate_from_repo(
            decimal amount, decimal vatRate, decimal expectedTotalAmount)
        {
            // Arrange
            var vatCalculator = new VatCalculator();

            var mock = new Mock<IRepository>();
            mock.Setup(r => r.getVatRate()).Returns(vatRate);
            var repo = mock.Object;

            var vatManager = new VatManager(repo, vatCalculator);

            // Act
            var actualTotalAmount = vatManager.calculateVat(amount); // <- Error

            // Assert
            Assert.Equal(expectedTotalAmount, actualTotalAmount);

            mock.Verify(r => r.getVatRate(), Times.Once);
        }
    }
}
```

Run Test!!!

References :-

- Getting started with xUnit.net :- https://xunit.github.io/docs/getting-started-dotnet-core
- Create a Web API with ASP.NET :- https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-vsc?view=aspnetcore-2.1
- Moq :- https://github.com/moq/moq4
- Moq Quickstart :- https://github.com/Moq/moq4/wiki/Quickstart
- Moq - Unit Test In .NET Core App Using Mock Object :- https://www.c-sharpcorner.com/article/moq-unit-test-net-core-app-using-mock-object/