
xUnit.net

xUnit.net

Jest to open-source'owe narzędzie do testowania jednostkowego dla platformy .NET. Zaprojektowany tak, aby jak najbardziej uprościć testowanie.

Należy do rodziny xUnit, czyli platform testowania jednostkowego bazowanych na SUnit wymyślonym w 1998.

Opiera się na modelu klasycznym – czyli każda assercja to oddzielna metoda, a nie jak w NUnit, gdzie każda assercja to przeciążenie Assert.That()

Wspiera też .NET Core, UWP, Xamarin MonoAndroid i Xamarin iOS Unified.

Dlaczego powstał xUnit.net?

Twórca NUnit 2.0 James Newkirk wraz ze swoim znajomym Brademem Wilsonem podczas kilku lat doświadczeń z różnymi frameworkami do testowania jednostkowego dla .NET zauważyli ich różne plusy i minusy. Uznali, że zamiast trzymać się sztywnych ram każdego z narzędzi mogą stworzyć coś swojego – lepszego.

Oprócz tego, powodem była też ewolucja .Net framework i jego nowe funkcje, które mogły znaleźć zastosowanie w testowaniu. (jak np. anonimowe delegaty, czy typy generyczne)

Chcieli też osiągnąć architekturę dla testowania TDD (test-driven development) z możliwością wsparcia innych rodzajów testowania.

Te wszystkie czynniki złożyły się na powstanie xUnit .net

Cechy xUnit.net

- **Jedna instancja obiektu na każdy test** – wzmacnia to izolację testów
- **Brak atrybutów [TestInitialize] i [TestCleanup]–** twórcy xUnit.net uważają owe za złe, ponieważ utrudniają czytelność kodu (zamiast skupić się na jednej metodzie testowej, trzeba przeglądać 2 lub 3); ponadto naruszają one izolację testów, ponieważ w metodach z tymi atrybutami inicjalizuje i niszczy się zmienne dla wszystkich metod, co oznacza, że byłyby one członkami klasy testowej (zmienne deklarowane w klasie) co mogłoby powodować konflikty w testach. Brak tych atrybutów w jakiś sposób chroni użytkowników przed tymi problemami. Można wymusić podobne działanie do tych atrybutów korzystając z konstruktorów i IDisposable
- **Brak [ExpectedExpection]** – Problem z tym atrybutem jest taki, że nie określa on konkretnej linii, która powinna zwrócić wyjątek, co może powodować problemy z analizą niektórych niepowodzeń; ponadto nie oferuje możliwości pełnego wglądu do wyjątku, ponieważ atrybut jest poza samym testowanym kodem. Ponadto atrybut ten wyłamywał się poza wzorzec 3A (Arrange,Act,Assert)
- **Brak [TestClass] –** testy mogą być w każdej publicznej klasie
- **Brak [Ignore] -** zamiast tego ustawia się parametr ‚Skip' na atrybucie [Fact]
- Starano się zredukować liczbę atrybutów, aby polegać bardziej na funkcjach języka.

Korzystanie z xUnit.net

xUnit.net można używać za pomocą:

- Konsoli
- Narzędzia MSBuild
- Narzędzia TestDriven.net (w VisualStudio)
- Narzędzia ReSharper
- Autorskiego GUI dostarczanego przez twórców xUnit.net
- Test Explorer w VisualStudio

Porównanie Assercji

NUnit 3.x (Constraint)	MSTest 15.x	xUnit.net 2.x	Comments
`Is.EqualTo`	`AreEqual`	`Equal`	MSTest and xUnit.net support generic versions of this method
`Is.Not.EqualTo`	`AreNotEqual`	`NotEqual`	MSTest and xUnit.net support generic versions of this method
`Is.Not.SameAs`	`AreNotSame`	`NotSame`	
`Is.SameAs`	`AreSame`	`Same`	
`Does.Contain`	`Contains`	`Contains`	
`Does.Not.Contain`	`DoesNotContain`	`DoesNotContain`	
`Throws.Nothing`	*n/a*	*n/a*	Ensures that the code does not throw any exceptions. See Note 5
n/a	`Fail`	*n/a*	xUnit.net alternative: `Assert.True(false, "message")`
`Is.GreaterThan`	*n/a*	*n/a*	xUnit.net alternative: `Assert.True(x > y)`
`Is.InRange`	*n/a*	`InRange`	Ensures that a value is in a given inclusive range
`Is.AssignableFrom`	*n/a*	`IsAssignableFrom`	
`Is.Empty`	*n/a*	`Empty`	
`Is.False`	`IsFalse`	`False`	
`Is.InstanceOf<T>`	`IsInstanceOfType`	`IsType<T>`	
`Is.NaN`	*n/a*	*n/a*	xUnit.net alternative: `Assert.True(double.IsNaN(x))`
`Is.Not.AssignableFrom<T>`	*n/a*	*n/a*	xUnit.net alternative: `Assert.False(obj is Type)`
`Is.Not.Empty`	*n/a*	`NotEmpty`	
`Is.Not.InstanceOf<T>`	`IsNotInstanceOfType`	`IsNotType<T>`	

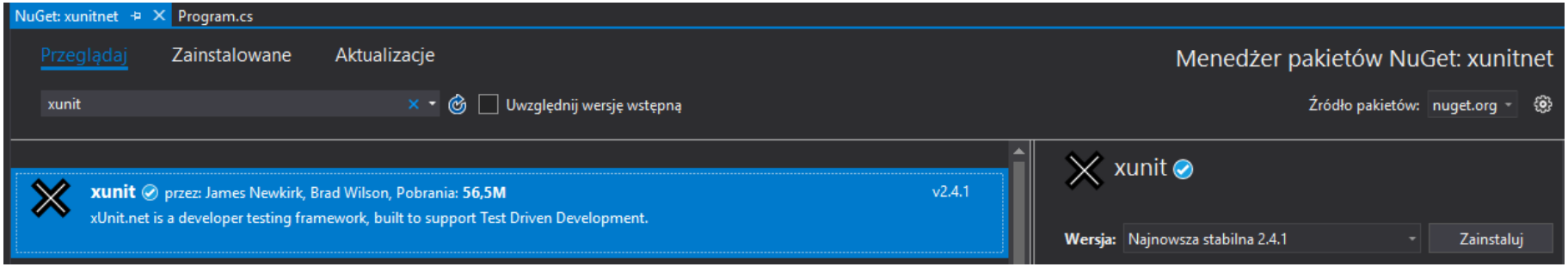
Porównanie Assercji c.d.

`Is.Not.Null`	`IsNotNull`	`NotNull`	
`Is.Null`	`IsNull`	`Null`	
`Is.True`	`IsTrue`	`True`	
`Is.LessThan`	*n/a*	*n/a*	xUnit.net alternative: `Assert.True(x < y)`
`Is.Not.InRange`	*n/a*	`NotInRange`	Ensures that a value is not in a given inclusive range
`Throws.TypeOf<T>`	*n/a*	`Throws<T>`	Ensures that the code throws an exact exception

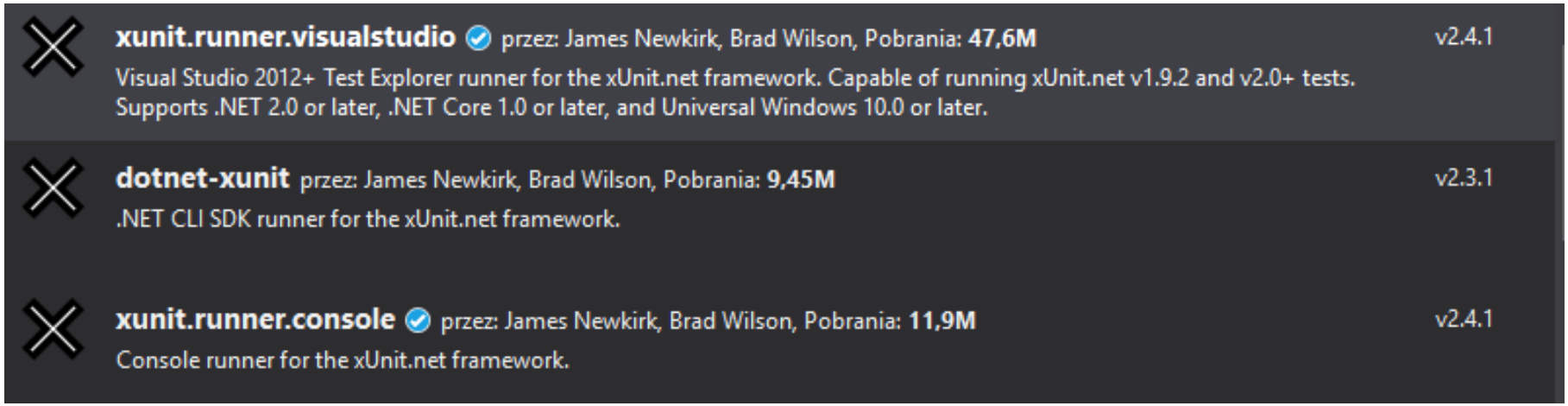
Porównanie Atrybutów

NUnit 3.x	MSTest 15.x	xUnit.net 2.x	Comments
`[Test]`	`[TestMethod]`	`[Fact]`	Marks a test method.
`[TestFixture]`	`[TestClass]`	*n/a*	xUnit.net does not require an attribute for a test class; it looks for all test methods in all public (exported) classes in the assembly.
`Assert.That` `Record.Exception`	`[ExpectedException]`	`Assert.Throws` `Record.Exception`	xUnit.net has done away with the ExpectedException attribute in favor of `Assert.Throws`. See Note 1
`[SetUp]`	`[TestInitialize]`	Constructor	We believe that use of `[SetUp]` is generally bad. However, you can implement a parameterless constructor as a direct replacement. See Note 2
`[TearDown]`	`[TestCleanup]`	`IDisposable.Dispose`	We believe that use of `[TearDown]` is generally bad. However, you can implement `IDisposable.Dispose` as a direct replacement. See Note 2
`[OneTimeSetUp]`	`[ClassInitialize]`	`IClassFixture<T>`	To get per-class fixture setup, implement `IClassFixture<T>` on your test class. See Note 3
`[OneTimeTearDown]`	`[ClassCleanup]`	`IClassFixture<T>`	To get per-class fixture teardown, implement `IClassFixture<T>` on your test class. See Note 3
n/a	*n/a*	`ICollectionFixture<T>`	To get per-collection fixture setup and teardown, implement `ICollectionFixture<T>` on your test collection. See Note 3
`[Ignore("reason")]`	`[Ignore]`	`[Fact(Skip="reason")]`	Set the Skip parameter on the `[Fact]` attribute to temporarily skip a test.
`[Property]`	`[TestProperty]`	`[Trait]`	Set arbitrary metadata on a test
`[Theory]`	`[DataSource]`	`[Theory]` `[XxxData]`	Theory (data-driven test). See Note 4

Instalacja xUnit.net



Dodajemy do rozwiązania pakiet xunit



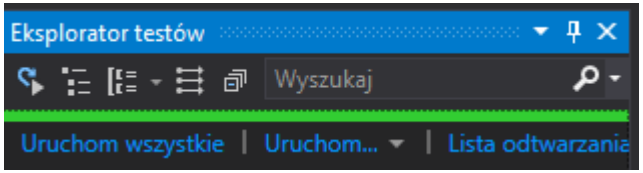
Ponadto musimy pobrać runner – może to być konsolowy lub VisualStudio (aby korzystać z Test Explorer)

Klasa testowa

```
public class Account
{
    private decimal balance=0;

    public void Deposit(decimal amount)
    {
        balance += amount;
    }

    public void Withdraw(decimal amount)
    {
        if (balance - amount >= 0)
            balance -= amount;
        else
            throw new InsufficientFundsException();
    }

    public void TransferFunds(Account destination, decimal amount){}

    public decimal Balance
    {
        get { return balance; }
    }
}
```

Uruchamianie testów

Aby skorzystać z TestExplorer używamy skrótu **Ctrl+E,T**



Aby skorzystać z konsolowego trybu uruchamiamy CMD

Wchodzimy do folderu z projektem, a następnie do:

packages\xunit.runner.console.2.4.1\tools\net472

Aby uruchomić test używamy:

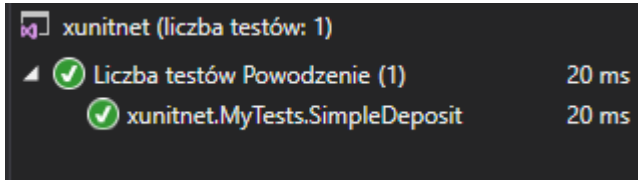
.\xunit.console.exe „Ścieżka do .exe naszego projektu”

(w folderze Debug/bin/nazwa.exe)

Atrybut [Fact]

```
[Fact]
public void SimpleDeposit()
{
    Account source = new Account();
    source.Deposit(150m);
    Assert.Equal(150m, source.Balance);
}
```

Wynik w TestExplorer



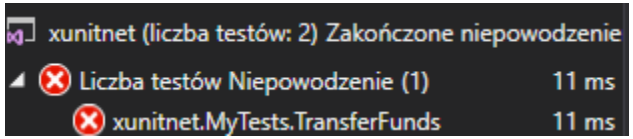
Wynik w konsoli

```
xUnit.net Console Runner v2.4.1 (64-bit Desktop .NET 4.7.2, runtime: 4.0.30319.42000)
  Discovering: xunitnet
  Discovered:  xunitnet
  Starting:    xunitnet
  Finished:    xunitnet
=== TEST EXECUTION SUMMARY ===
   xunitnet  Total: 1, Errors: 0, Failed: 0, Skipped: 0, Time: 0,119s
```

Atrybut [Fact]

```
[Fact]
public void TransferFunds()
{
    Account source = new Account();
    source.Deposit(200m);

    Account destination = new Account();
    destination.Deposit(150m);

    source.TransferFunds(destination, 100m);

    Assert.Equal(250m, destination.Balance);
    Assert.Equal(100m, source.Balance);
}
```

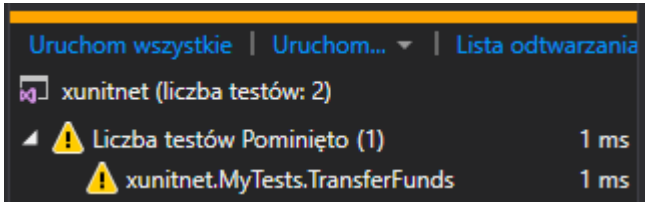
Brak implementacji metody TransferFunds




```
xUnit.net Console Runner v2.4.1 (64-bit Desktop .NET 4.7.2, runtime: 4.0.30319.42000)
  Discovering: xunitnet
  Discovered:  xunitnet
  Starting:    xunitnet
    xunitnet.MyTests.TransferFunds [FAIL]
      Assert.Equal() Failure
      Expected: 250
      Actual:   150
      Stack Trace:
           w Xunit.Assert.Equal[T](T expected, T actual, IEqualityComparer`1 comparer) w C:\Dev\xunit\xunit\src\xunit.as
sert\Asserts\EqualityAsserts.cs:wiersz 40
           w Xunit.Assert.Equal[T](T expected, T actual) w C:\Dev\xunit\xunit\src\xunit.assert\Asserts\EqualityAsserts.c
s:wiersz 25
           w xunitnet.MyTests.TransferFunds() w C:\Users\barto\Desktop\xUnit net\xunitnet\xunitnet\Program.cs:wiersz 147

  Finished:    xunitnet
=== TEST EXECUTION SUMMARY ===
   xunitnet  Total: 2, Errors: 0, Failed: 1, Skipped: 0, Time: 0,209s
```

Parametr [Fact(Skip = „Something”]

```
[Fact(Skip = "Will implement later")]
public void TransferFunds()
{
    Account source = new Account();
    source.Deposit(200m);

    Account destination = new Account();
    destination.Deposit(150m);

    source.TransferFunds(destination, 100m);

    Assert.Equal(250m, destination.Balance);
    Assert.Equal(100m, source.Balance);
}
```




```
xUnit.net Console Runner v2.4.1 (64-bit Desktop .NET 4.7.2, runtime: 4.0.30319.42000)
  Discovering: xunitnet
  Discovered:  xunitnet
  Starting:    xunitnet
    xunitnet.MyTests.TransferFunds [SKIP]
      Will implement later
  Finished:    xunitnet
=== TEST EXECUTION SUMMARY ===
   xunitnet  Total: 2, Errors: 0, Failed: 0, Skipped: 1, Time: 0,135s
```

Korzystanie z konstruktora

Żeby uniknąć powtarzania tego samego kodu, można skorzystać z konstruktora

```
Account source;
public MyTests()
{
    source = new Account();
}
```

Wtedy w metodach testowych już nie tworzymy *source*, tylko z niego korzystamy

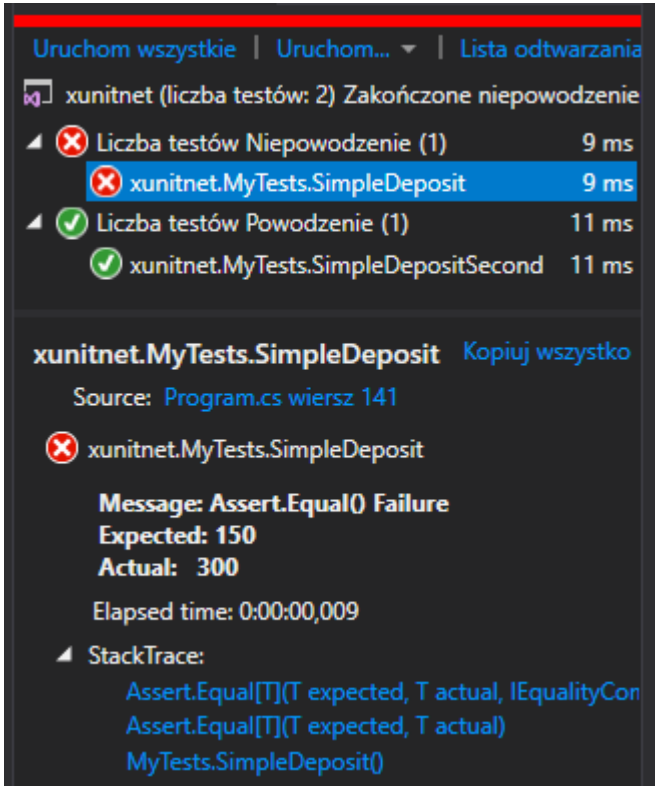
```
[Fact]
public void SimpleDeposit()
{
    source.Deposit(150m);
    Assert.Equal(150m, source.Balance);
}
[Fact(Skip = "Will implement later")]
public void TransferFunds()
{
    source.Deposit(200m);

    Account destination = new Account();
    destination.Deposit(150m);

    source.TransferFunds(destination, 100m);

    Assert.Equal(250m, destination.Balance);
    Assert.Equal(100m, source.Balance);
}
```

xUnit.net dla każdej metody testowej tworzy nową instancję!

Interfejs IClassFixture<>

Pozwala na korzystanie w wielu metodach testowych z tego samego obiektu. (nie zostanie stworzony nowy obiekt – czyli to co zmienimy w jednej metodzie testowej będzie widoczne też w następnej).

```
public class MyTests : IClassFixture<Account>
{
        Account source;
        public MyTests(Account fixture)
        {
            source = fixture;
        }

        [Fact]
        public void SimpleDepositSecond()
        {
            source.Deposit(150m);
            Assert.Equal(150m, source.Balance);
        }

        [Fact]
        public void SimpleDeposit()
        {
            source.Deposit(150m);
            Assert.Equal(150m, source.Balance);
        }
}
```



Dodaliśmy kolejne 150 do istniejących 150.

Wyjątki – Assert.Throws

Do testowania wyjątków służy Assert.Throws<Wyjątek>
Tworzę swój wyjątek, który będzie pasował do mojej metody:

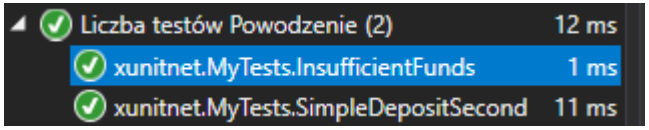
```
public class InsufficientFundsException : ApplicationException { }
```

Zmieniam metodę *Withdraw*, tak aby rzucała wyjątek w przypadku za małej ilości pieniędzy na koncie.

```
public void Withdraw(decimal amount)
{
    if (balance - amount >= 0)
        balance -= amount;
    else
        throw new InsufficientFundsException();
}
```

Metoda testowa:

```
[Fact]
public void InsufficientFunds()
{
    Assert.Throws<InsufficientFundsException>(() => source.Withdraw(600m));
}
```

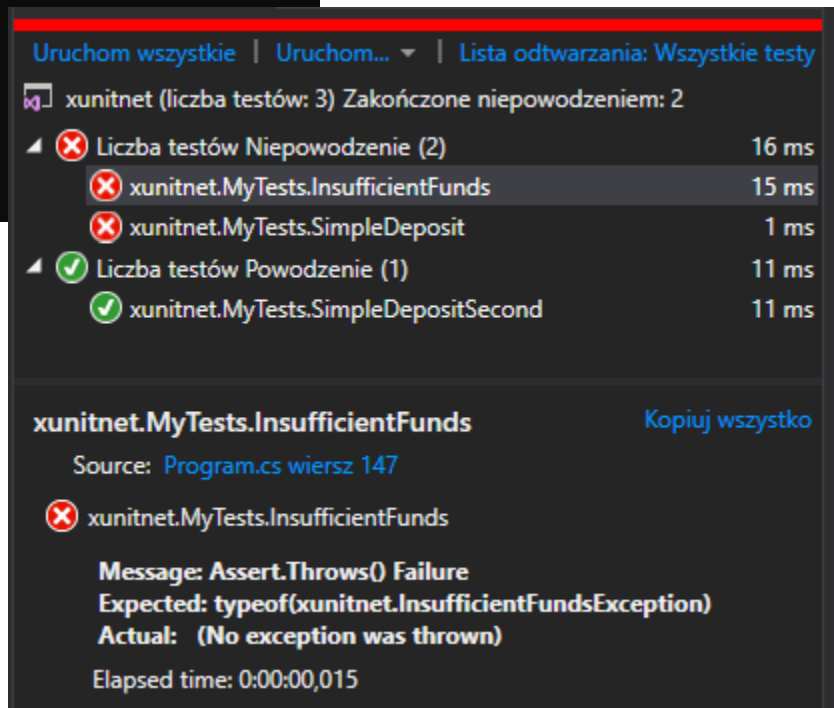


Wyjątki – Assert.Throws

W przypadku, jeżeli metoda testowa się nie powiedzie otrzymamy informację o tym czy wyjątek został zwrócony i jeżeli tak to jaki

```
[Fact]
public void InsufficientFunds()
{
    Assert.Throws<InsufficientFundsException>(() => source.Withdraw(100m));
}
```

```
xUnit.net Console Runner v2.4.1 (64-bit Desktop .NET 4.7.2, runtime: 4.0.30319.42000)
  Discovering: xunitnet
  Discovered:  xunitnet
  Starting:    xunitnet
    xunitnet.MyTests.InsufficientFunds [FAIL]
      Assert.Throws() Failure
      Expected: typeof(xunitnet.InsufficientFundsException)
      Actual:   (No exception was thrown)
```



Atrybut [Theory]

Korzystamy z niego, gdy chcemy coś przetestować dla kilku przypadków (unikamy kopiowania kodu). Gdy chcemy pokazać działanie danej funkcji programu. Wymaga sparametryzowania metody testowej.

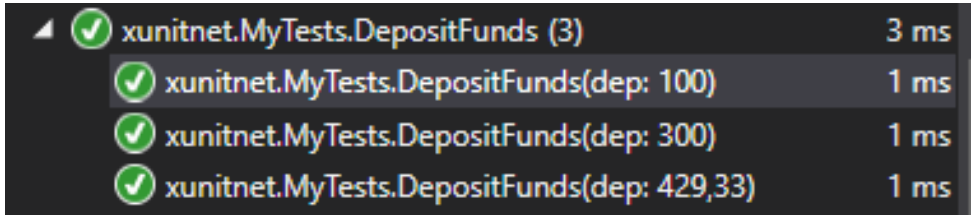
Posiada kilka podatrybutów:

- **[InlineData]** – podajemy tutaj stałe
- **[ClassData]** – jeżeli dane nie są stałymi, tworzymy nową klasę z danymi
- **[MemberData]** – podobnie jak w ClassData, ale nie trzeba tworzyć nowej klasy, może to być dowolny obiekt zwracający Ienumerable
- Możemy też stworzyć własny atrybut – musi dziedziczyć po *DataAttribute*

Atrybut [Theory] - [InlineData]

```
[Theory]
[InlineData(100)]
[InlineData(300f)]
[InlineData(429.33)]
public void DepositFunds(decimal dep)
{
        Account acc = new Account();
        acc.Deposit(dep);
        Assert.Equal(dep, acc.Balance);
}
```

Wykonuje to 3 kolejne testy, dla wartości podanych w nawiasach. Dla każdego takiego testu jest w tym przypadku tworzona nowa instancja.



Jeżeli korzystalibyśmy tu z *fixtury* to w każdym kolejnym teście w depozycie byłaby suma z poprzednich testów.

Atrybut [Theory] – [ClassData]

Aby korzystać z ClassData musimy stworzyć klasę dziedziczącą po IEnumerable<>

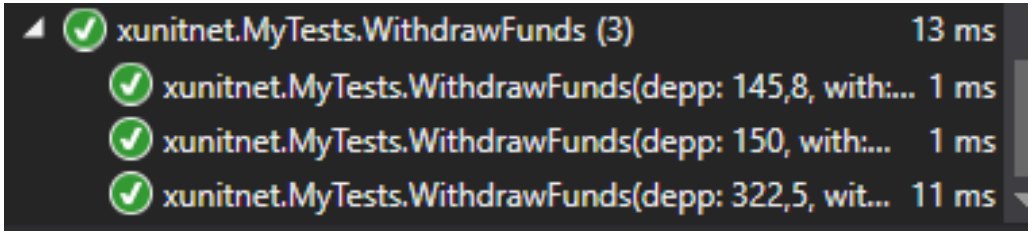
```
public class AccountTestData : IEnumerable<object[]>
    {
        public IEnumerator<object[]> GetEnumerator()
        {
            yield return new object[] { 150m,50m,100m };
            yield return new object[] { 322.5, 100,222.5 };
            yield return new object[] { 145.8m,40.3,105.5f };
        }

        IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
    }
```

Słówko *yield* oznacza, że zwracamy każdy element po kolei. Enumerator jest typu *object[],* aby móc sprawdzić zgodność dla różnych typów. Używamy tablicy ponieważ metoda testowa będzie posiadała więcej niż jeden argument.

Atrybut [Theory] – [ClassData]

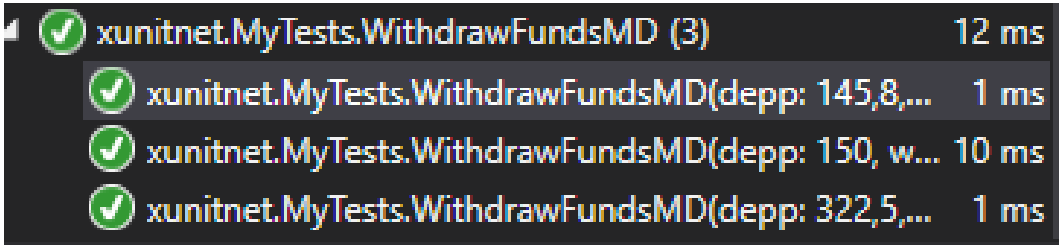
```
[Theory]
[ClassData(typeof(AccountTestData))]
public void WithdrawFunds(decimal depp, decimal with, decimal expected)
{
        Account acc = new Account();
        acc.Deposit(depp);
        acc.Withdraw(with);
        Assert.Equal(expected, acc.Balance);
}
```

Liczba argumentów musi się zgadzać z liczbą elementów w tablicy zwracanej przez Enumerator.



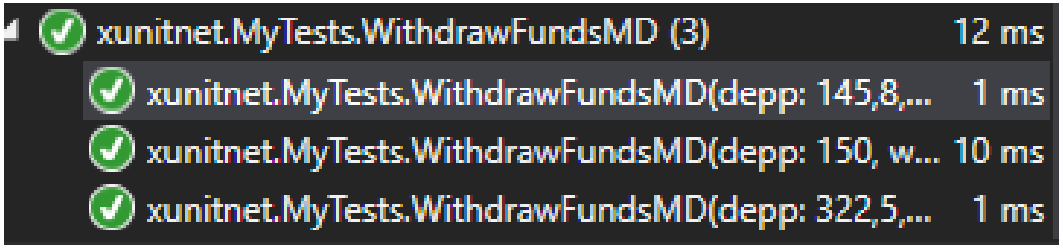
Atrybut [Theory] – [MemberData]

```
[Theory]
[MemberData(nameof(Data))]
public void WithdrawFundsMD(decimal depp, decimal with, decimal expected)
{
        Account acc = new Account();
        acc.Deposit(depp);
        acc.Withdraw(with);
        Assert.Equal(expected, acc.Balance);
}
public static IEnumerable<object[]> Data =>
            new List<object[]>
            {
                new object[] { 150m,50m,100m },
                new object[] { 322.5, 100,222.5 },
                new object[] { 145.8m,40.3,105.5f }
            };
```



Atrybut [Theory] – [MemberData]

```
[Theory]
[MemberData(nameof(Data))]
public void WithdrawFundsMD(decimal depp, decimal with, decimal expected)
{
        Account acc = new Account();
        acc.Deposit(depp);
        acc.Withdraw(with);
        Assert.Equal(expected, acc.Balance);
}
public static IEnumerable<object[]> Data =>
            new List<object[]>
            {
                new object[] { 150m,50m,100m },
                new object[] { 322.5, 100,222.5 },
                new object[] { 145.8m,40.3,105.5f }
            };
```



Atrybut [Theory] – Custom Attribute

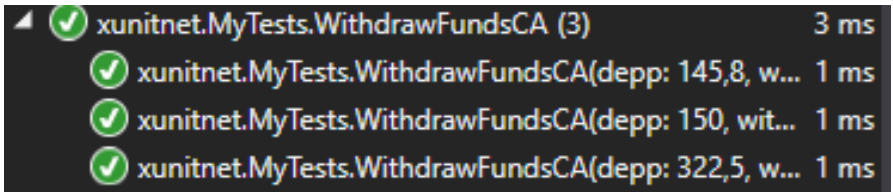
Należy stworzyć nową klasę dziedziczącą po DataAttribute

```
public class AccountDataAttribute : DataAttribute
{
    public override IEnumerable<object[]> GetData(MethodInfo testMethod)
    {

        yield return new object[] { 322.5, 100 };
        yield return new object[] { 195.8m, 40.3 };
        yield return new object[] { 160f, 50 };

    }
}
```

Następnie używamy tylko nazwy klasy bez końcówki *Attribute*

```
[Theory]
[AccountData]
public void WithdrawFundsCA(decimal depp, decimal with)
{
     source.Deposit(depp);
     source.Withdraw(with);
     Assert.InRange(source.Balance, 300, 800);
}
```

Przy argumentach Theory kolejność wymieniania danych najwyraźniej nie wpływa na kolejność ich odczytywania



Atrybut [Trait]

Służy do grupowania metod testowych

```
[Trait("Category", "Name")]
```

Używamy go przy metodach testowych

```
[Theory]
[AccountData]
[Trait("Category","Theories")]
public void WithdrawFundsCA(decimal depp, decimal with)
{
    source.Deposit(depp);
    source.Withdraw(with);
    Assert.InRange(source.Balance, 300, 800);
}
[Fact]
[Trait("Category","Facts")]
public void SimpleDepositSecond()
{
    source.Deposit(150m);
    Assert.Equal(150m, source.Balance);
}
[Trait("Category", "Facts")]
[Fact]
public void SimpleDeposit()
{
    source.Deposit(150m);
    Assert.Equal(150m, source.Balance);
}
```