# [Test] + <T> = ♡

Александр Катин
Энтузиаст C#

DOTNEXT

# Дарим радость через смех

```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

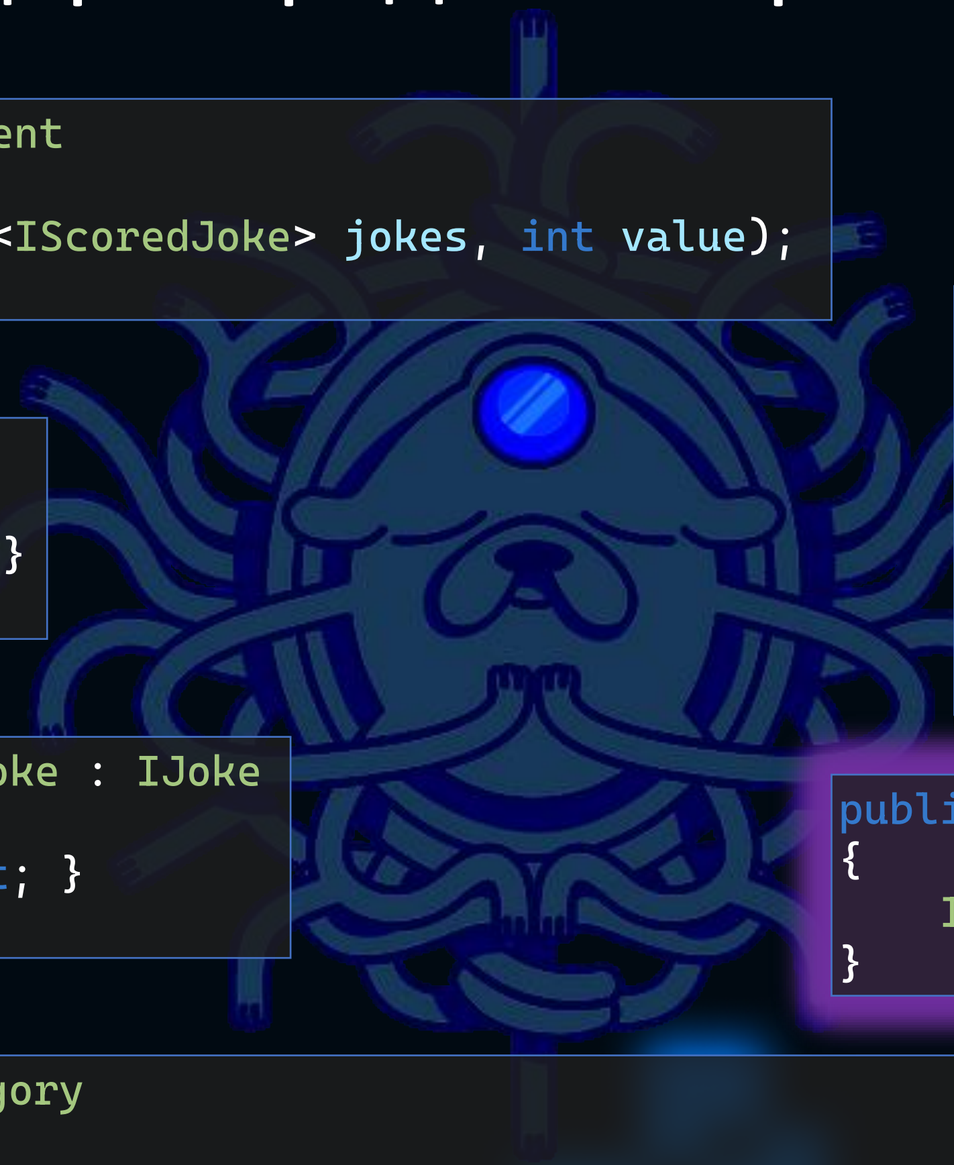
```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

# Дарим радость через смех



```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

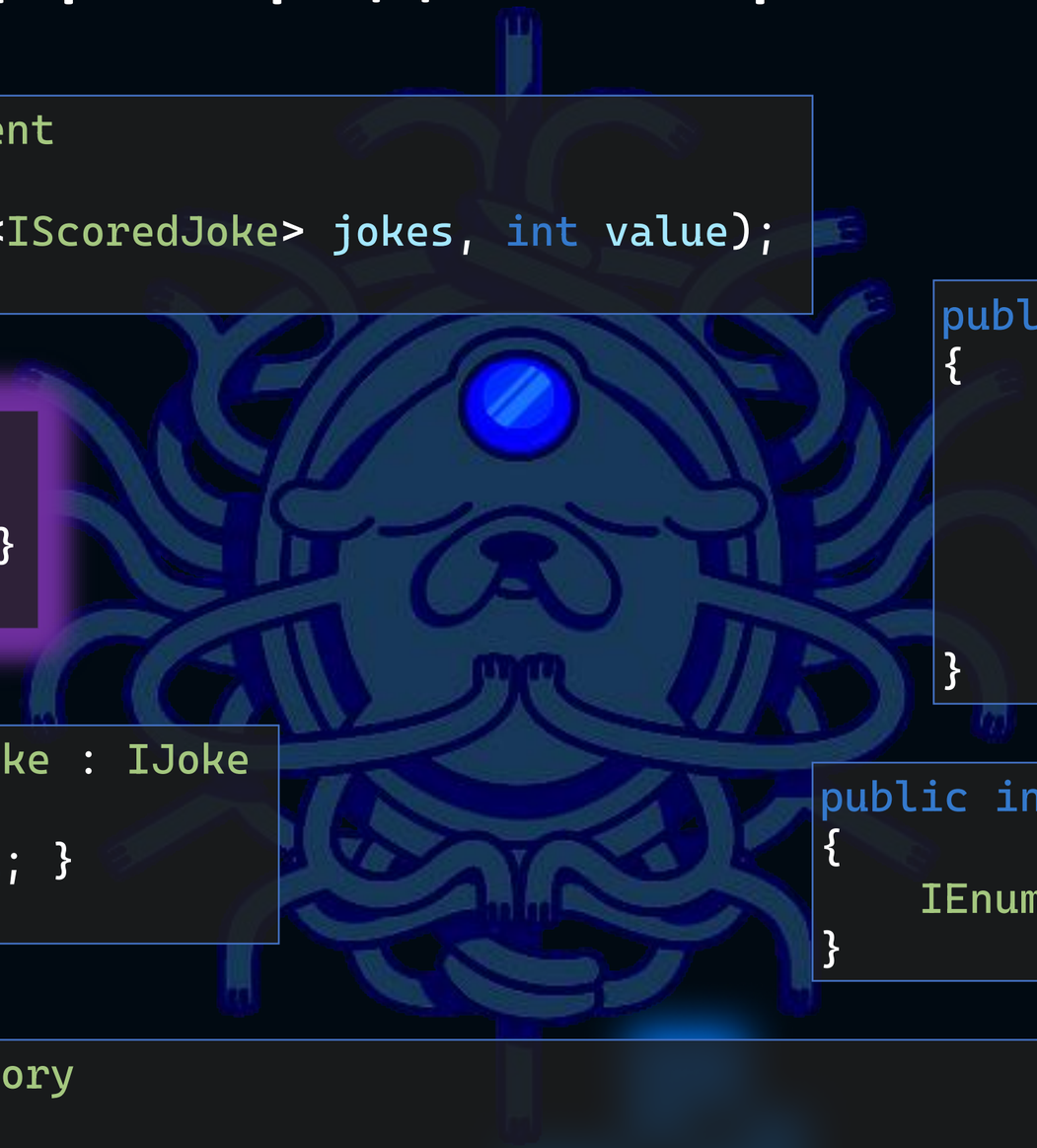
```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

# Дарим радость через смех



```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

# Дарим радость через смех

```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

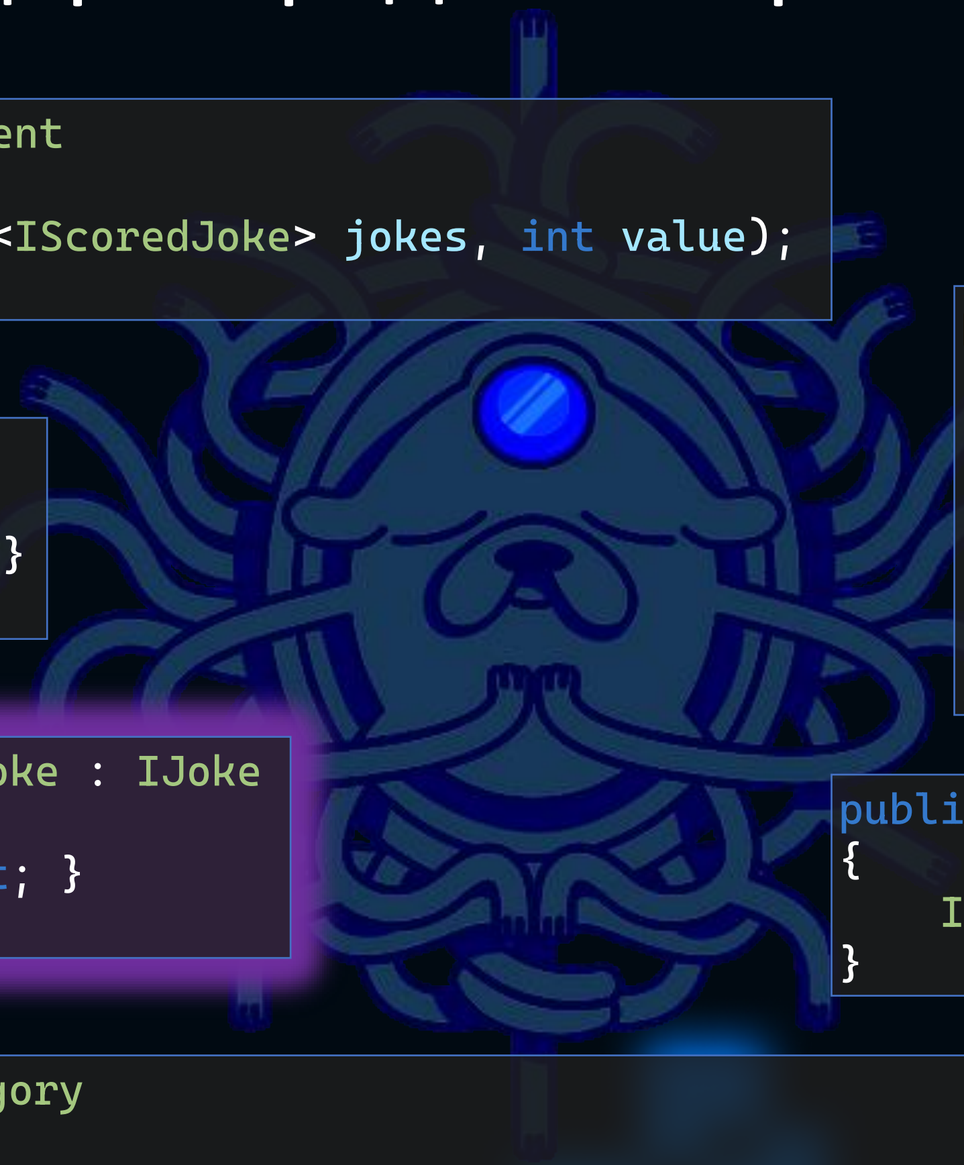
```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

# Дарим радость через смех



```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

# Дарим радость через смех

```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

# Дарим радость через смех

```
public interface IRequirement
{
    bool IsMet(IEnumerable<IScoredJoke> jokes, int value);
}
```

```
public interface IJoke
{
    string Joke { get; }
}
```

```
public interface IScoredJoke : IJoke
{
    IJokeScore Score { get; }
}
```

```
public interface ICategory
{
    IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes);
}
```

```
public interface IJokeScore
{
    int Famous { get; }
    int Funny { get; }
    int Offensive { get; }
    int Old { get; }
    int Sophisticated { get; }
}
```

```
public interface IJokeProvider
{
    IEnumerable<IJoke> GetJokes();
}
```

# Дарим радость через смех

```
public class DummyProvider : IJokeProvider
{
    public IEnumerable<IJoke> GetJokes() =>
        default;
}
```

```
public class RussianJokes : DummyProvider { }

public class GibraltarJokes : DummyProvider { }
```

```
public class DummyCategory : ICategory
{
    public IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes) =>
        default;
}
```

```
public class ForParty : DummyCategory { }

public class AboutCoders : DummyCategory { }
```

```
public class AreOfMinimalFun : IRequirement
{
    public bool IsMet(IEnumerable<IScoredJoke> jokes, int value) =>
        75 >= value;
}
```

# Дарим радость через смех

```
public class DummyProvider : IJokeProvider
{
    public IEnumerable<IJoke> GetJokes() =>
        default;
}
```

```
public class RussianJokes : DummyProvider { }

public class GibraltarJokes : DummyProvider { }
```

```
public class DummyCategory : ICategory
{
    public IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes) =>
        default;
}
```

```
public class ForParty : DummyCategory { }

public class AboutCoders : DummyCategory { }
```

```
public class AreOfMinimalFun : IRequirement
{
    public bool IsMet(IEnumerable<IScoredJoke> jokes, int value) =>
        75 >= value;
}
```

# Дарим радость через смех

```
public class DummyProvider : IJokeProvider
{
    public IEnumerable<IJoke> GetJokes() =>
        default;
}
```

```
public class RussianJokes : DummyProvider { }

public class GibraltarJokes : DummyProvider { }
```

```
public class DummyCategory : ICategory
{
    public IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes) =>
        default;
}
```

```
public class ForParty : DummyCategory { }

public class AboutCoders : DummyCategory { }
```

```
public class AreOfMinimalFun : IRequirement
{
    public bool IsMet(IEnumerable<IScoredJoke> jokes, int value) =>
        75 >= value;
}
```

# Дарим радость через смех

```
public class DummyProvider : IJokeProvider
{
    public IEnumerable<IJoke> GetJokes() =>
        default;
}
```

```
public class RussianJokes : DummyProvider { }

public class GibraltarJokes : DummyProvider { }
```

```
public class DummyCategory : ICategory
{
    public IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes) =>
        default;
}
```

```
public class ForParty : DummyCategory { }

public class AboutCoders : DummyCategory { }
```

```
public class AreOfMinimalFun : IRequirement
{
    public bool IsMet(IEnumerable<IScoredJoke> jokes, int value) =>
        75 >= value;
}
```

# Дарим радость через смех

```
public class DummyProvider : IJokeProvider
{
    public IEnumerable<IJoke> GetJokes() =>
        default;
}
```

```
public class RussianJokes : DummyProvider { }

public class GibraltarJokes : DummyProvider { }
```

```
public class DummyCategory : ICategory
{
    public IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes) =>
        default;
}
```

```
public class ForParty : DummyCategory { }

public class AboutCoders : DummyCategory { }
```

```
public class AreOfMinimalFun : IRequirement
{
    public bool IsMet(IEnumerable<IScoredJoke> jokes, int value) =>
        75 >= value;
}
```

# Дарим радость через смех

```
public class DummyProvider : IJokeProvider
{
    public IEnumerable<IJoke> GetJokes() =>
        default;
}
```

```
public class RussianJokes : DummyProvider { }

public class GibraltarJokes : DummyProvider { }
```

```
public class DummyCategory : ICategory
{
    public IEnumerable<IScoredJoke> GetTopJokes(IEnumerable<IJoke> jokes) =>
        default;
}
```

```
public class ForParty : DummyCategory { }

public class AboutCoders : DummyCategory { }
```

```
public class AreOfMinimalFun : IRequirement
{
    public bool IsMet(IEnumerable<IScoredJoke> jokes, int value) =>
        75 >= value;
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Тестируем так

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Что больше всего смущает?

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Что больше всего смущает?

```
[TestCase(typeof(RussianJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(RussianJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(AboutCoders), typeof(AreOfMinimalFun), 75)]
[TestCase(typeof(GibraltarJokes), typeof(ForParty), typeof(AreOfMinimalFun), 75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test(Type providerType, Type categoryType, Type requirementType, int value)
{
    // Arrange
    IJokeProvider provider = (IJokeProvider)Activator.CreateInstance(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = (ICategory)Activator.CreateInstance(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = (IRequirement)Activator.CreateInstance(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Предположим, добавили такой тест и запустили тестирование

```
[TestCase(typeof(object), typeof(object), typeof(object), 75)]
```

Что случится?

- Проект не соберется
- Test Explorer сломается
- Тест упадет
- Тест пройдет

# Предположим, добавили такой тест и запустили тестирование

Test Detail Summary

Test(System.Object,System.Object,System.Object,75)

Source: IntegrationTests.cs line 18

Duration: 115 ms

Message:

System.InvalidCastException : Unable to cast object of type 'System.Object' to type 'Jokes.Abstractions.IJokeProvider'.

st(Type providerType, Type categoryType, Type requirementType, Int32 value) line 21

#ЭТО ЕЩЁ КОНСТРУКТОР ПОДХОДЯЩИЙ НАШЁЛСЯ

Что случится?

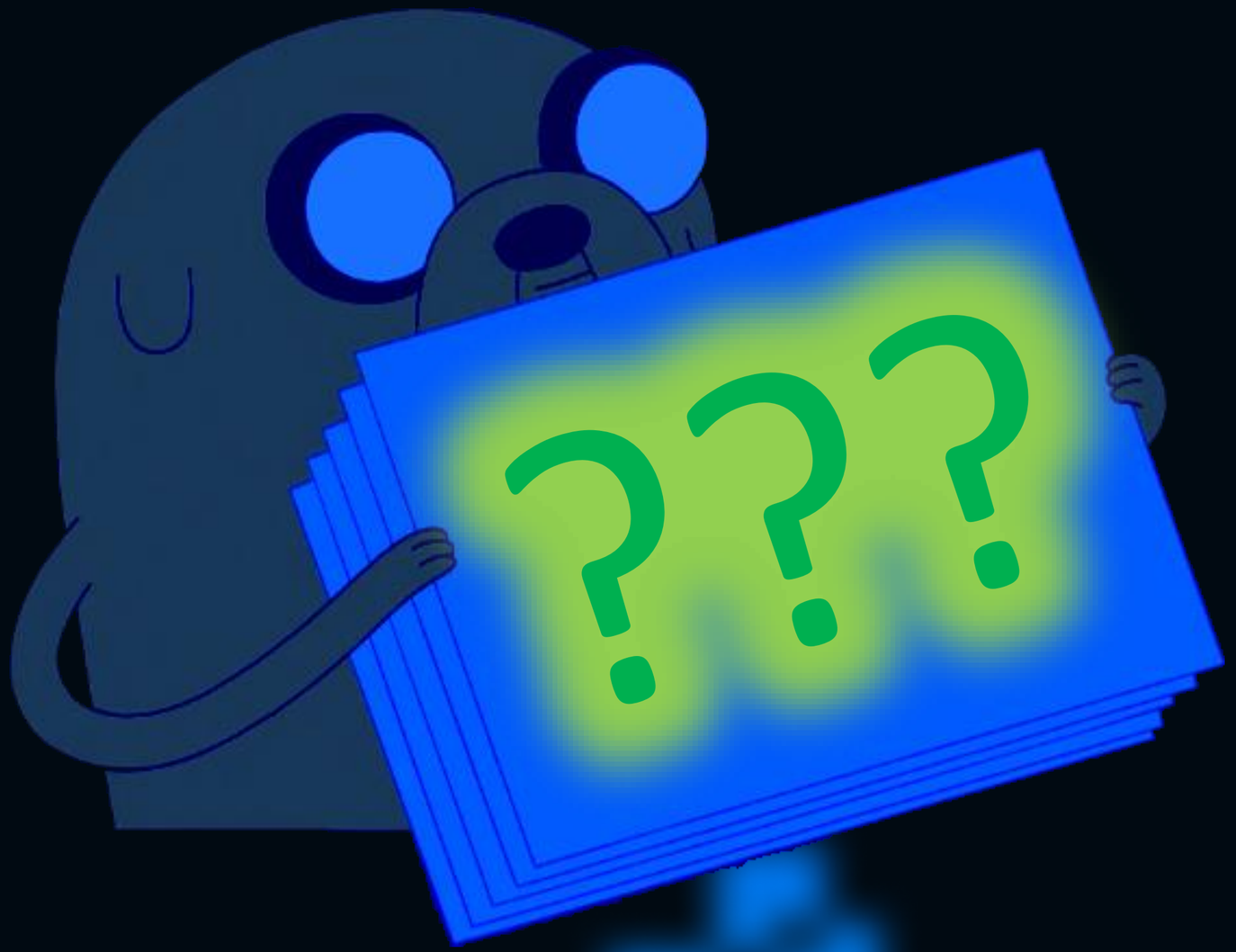
Проект не соберется

Test Explorer сломается

Тест упадет

Тест пройдет

# Как добиться контроля типов?



# Экспресс-тест: Кто Вы в мире юнит-тестирования

# Наивный юнец

```
[Test]
public void Test_RussianJokes_AboutCoders_AreOfMinimalFun()
{
    // Arrange
    IJokeProvider provider = new RussianJokes();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new AboutCoders();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new AreOfMinimalFun();
    bool actual = requirement.IsMet(jokesOnCategory, 75);
    Assert.IsTrue(actual);
}
```

# Наивный юнец

```
[Test]
public void Test_RussianJokes_AboutCoders_AreOfMinimalFun()
{
    // Arrange
    IJokeProvider provider = new RussianJokes();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new AboutCoders();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new AreOfMinimalFun();
    bool actual = requirement.IsMet(jokesOnCategory, 75);
    Assert.IsTrue(actual);
}
```

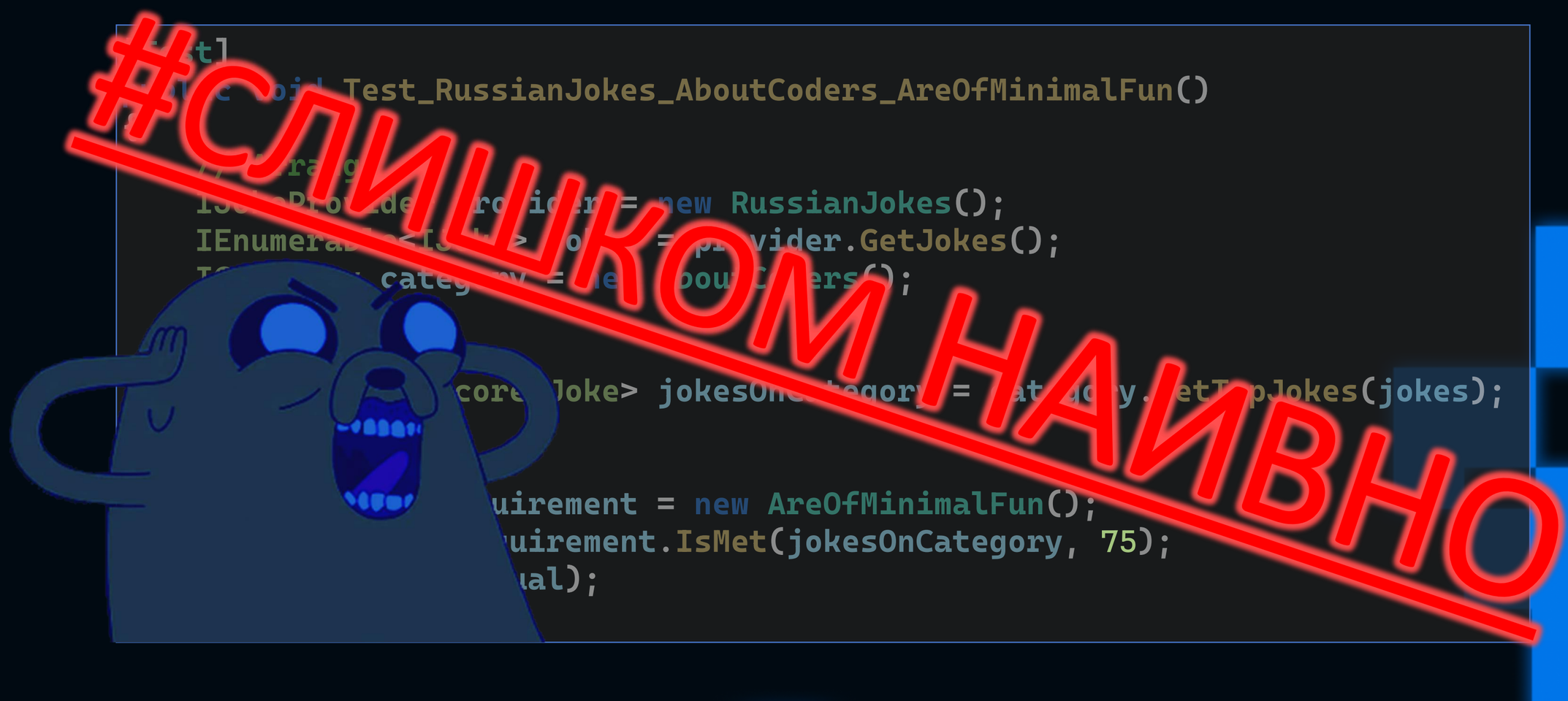
# Наивный юнец

```
[Test]
public void Test_RussianJokes_AboutCoders_AreOfMinimalFun()
{
    // Arrange
    IJokeProvider provider = new RussianJokes();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new AboutCoders();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new AreOfMinimalFun();
    bool actual = requirement.IsMet(jokesOnCategory, 75);
    Assert.IsTrue(actual);
}
```

# Наивный юнец

```
[Test]
public void Test_RussianJokes_AboutCoders_AreOfMinimalFun()
{
    // Arrange
    IJokeProvider provider = new RussianJokes();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new AboutCoders();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new AreOfMinimalFun();
    bool actual = requirement.IsMet(jokesOnCategory, 75);
    Assert.IsTrue(actual);
}
```

# Наивный юнец



# Бывалый капитан

```
enum Provider
{
    RussianJokes,
    GibraltarJokes
}
```

```
IJokeProvider CreateProvider(Provider providerType) =>
    providerType switch
    {
        Provider.RussianJokes => new RussianJokes(),
        Provider.GibraltarJokes => new GibraltarJokes(),
        _ => throw new ArgumentException(),
    };
```

# Бывалый капитан

```
enum Provider
{
    RussianJokes,
    GibraltarJokes
}
```

```
IJokeProvider CreateProvider(Provider providerType) =>
    providerType switch
    {
        Provider.RussianJokes => new RussianJokes(),
        Provider.GibraltarJokes => new GibraltarJokes(),
        _ => throw new ArgumentException(),
    };
```

# Бывалый капитан

```
enum Provider
{
    RussianJokes,
    GibraltarJokes
}
```

```
IJokeProvider CreateProvider(Provider providerType) =>
    providerType switch
    {
        Provider.RussianJokes => new RussianJokes(),
        Provider.GibraltarJokes => new GibraltarJokes(),
        _ => throw new ArgumentException(),
    };
```

# Бывалый капитан

```
[TestCase(Provider.RussianJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.RussianJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
public void Test(Provider providerType, Category categoryType, Requirement requirementType, int value)
{
    // Arrange
    IJokeProvider provider = CreateProvider(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = CreateCategory(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = CreateRequirement(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Бывалый капитан

```
[TestCase(Provider.RussianJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.RussianJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
public void Test(Provider providerType, Category categoryType, Requirement requirementType, int value)
{
    // Arrange
    IJokeProvider provider = CreateProvider(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = CreateCategory(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = CreateRequirement(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Бывалый капитан

```
[TestCase(Provider.RussianJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.RussianJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
public void Test(Provider providerType, Category categoryType, Requirement requirementType, int value)
{
    // Arrange
    IJokeProvider provider = CreateProvider(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = CreateCategory(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = CreateRequirement(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Бывалый капитан

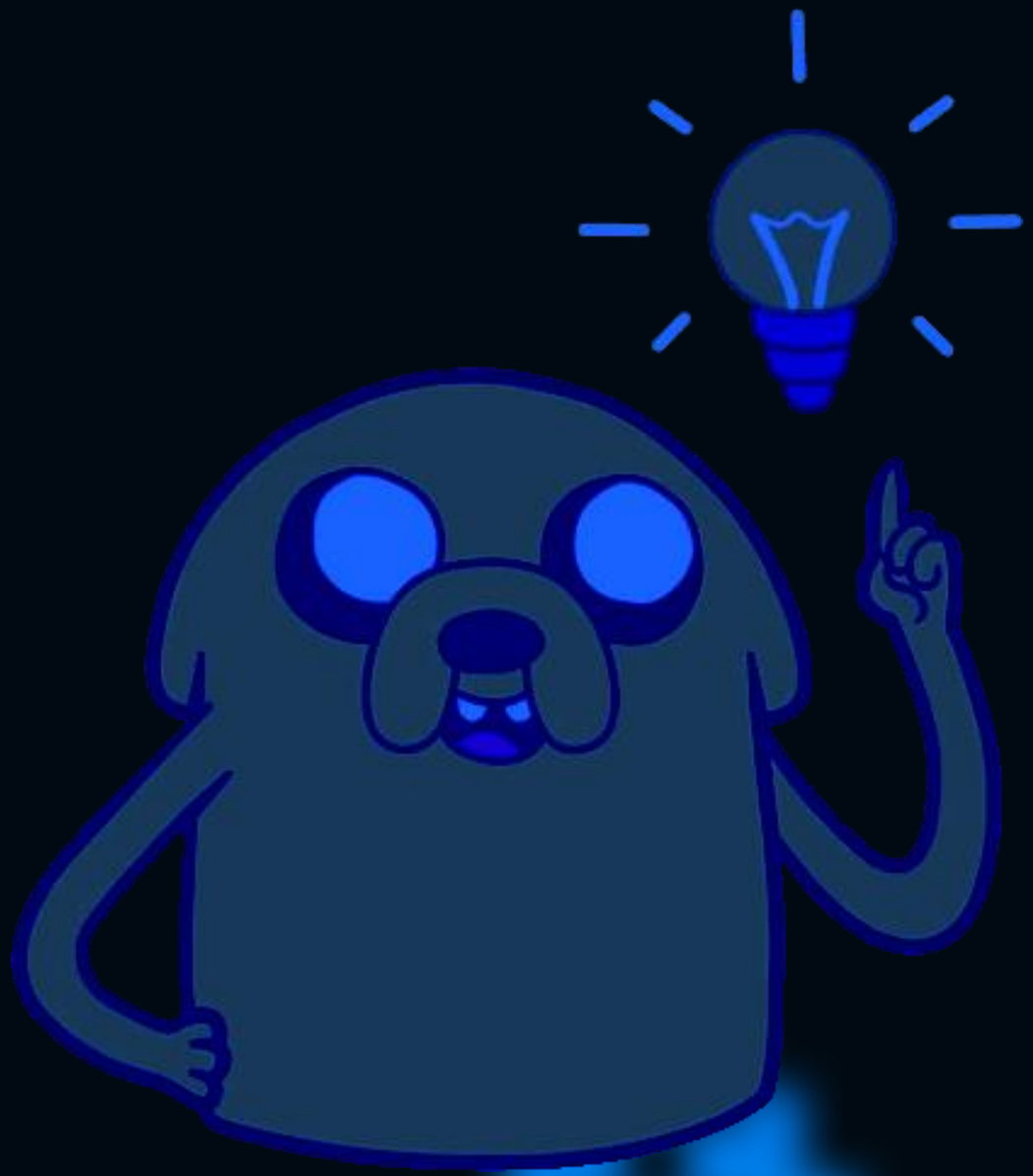
```
[TestCase(Provider.RussianJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.RussianJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
public void Test(Provider providerType, Category categoryType, Requirement requirementType, int value)
{
    // Arrange
    IJokeProvider provider = CreateProvider(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = CreateCategory(categoryType);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = CreateRequirement(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Бывалый капитан

```
[TestCase(Provider.RussianJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.RussianJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.AboutCoders, Requirement.AreOfMinimalFun, 75)]
[TestCase(Provider.GibraltarJokes, Category.ForParty, Requirement.AreOfMinimalFun, 75)]
public void Test(Provider providerType, Category categoryType, Requirement requirementType, int value)
{
    // Arrange
    IJokeProvider provider = CreateProvider(providerType);
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = CreateCategory(categoryType, jokes);

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = CreateRequirement(requirementType);
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

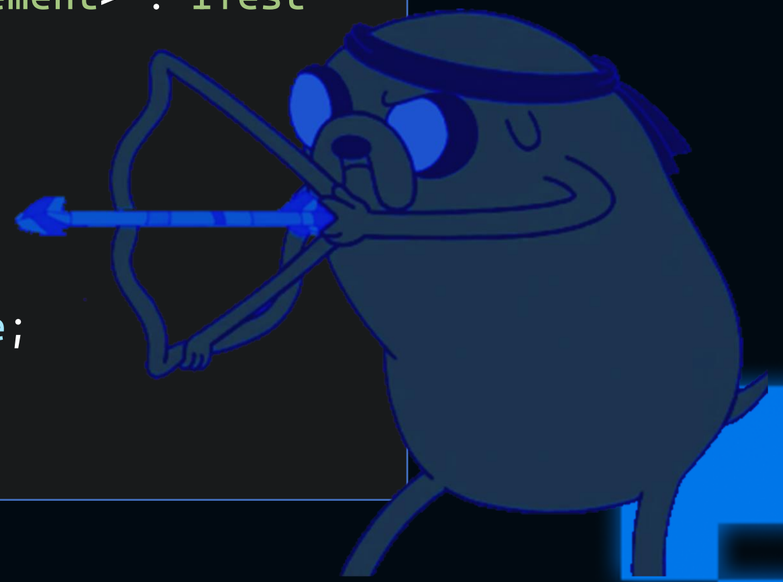
#НУ ТАКОЕ СЕБЕ

# Использование обобщений



# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public Check(int value) => this.value = value;

    // ...
}
```



```
public interface ITest
{
    void Execute();
}
```

# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public Check(int value) => this.value = value;

    // ...
}
```

```
public interface ITest
{
    void Execute();
}
```

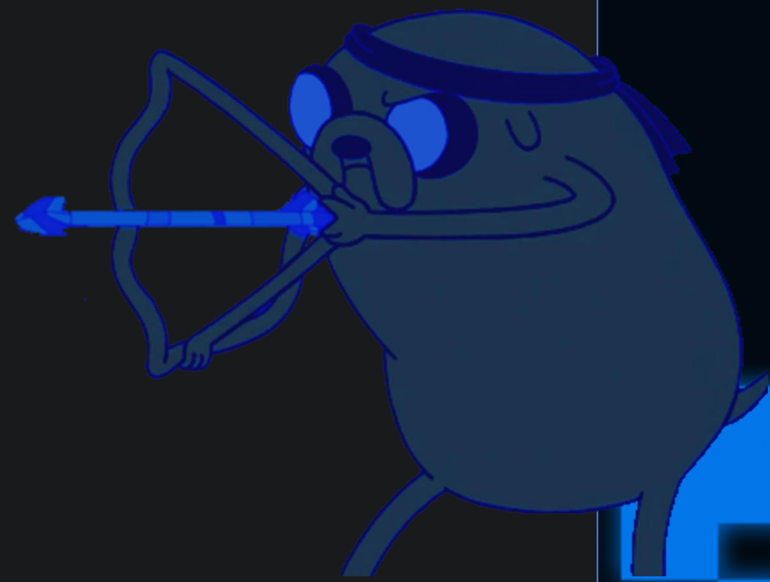
# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public Check(int value) => this.value = value;

    // ...
}
```

```
public interface ITest
{
    void Execute();
}
```

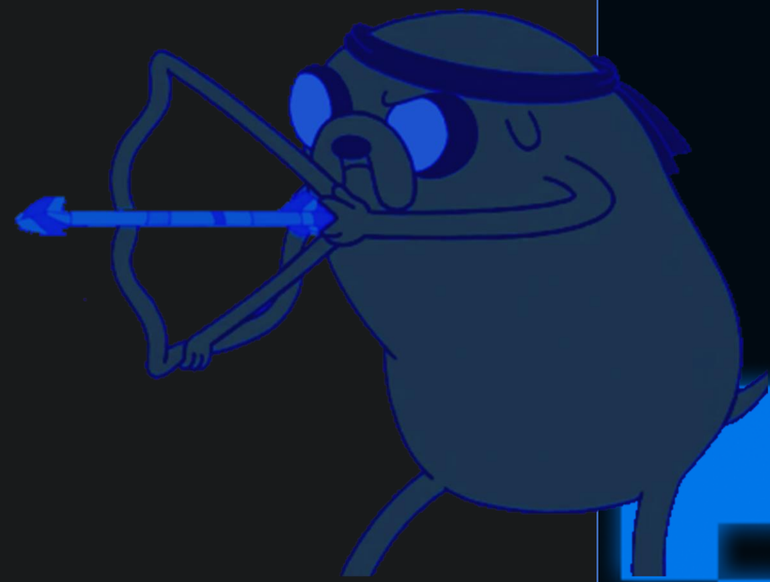
# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void Execute()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```



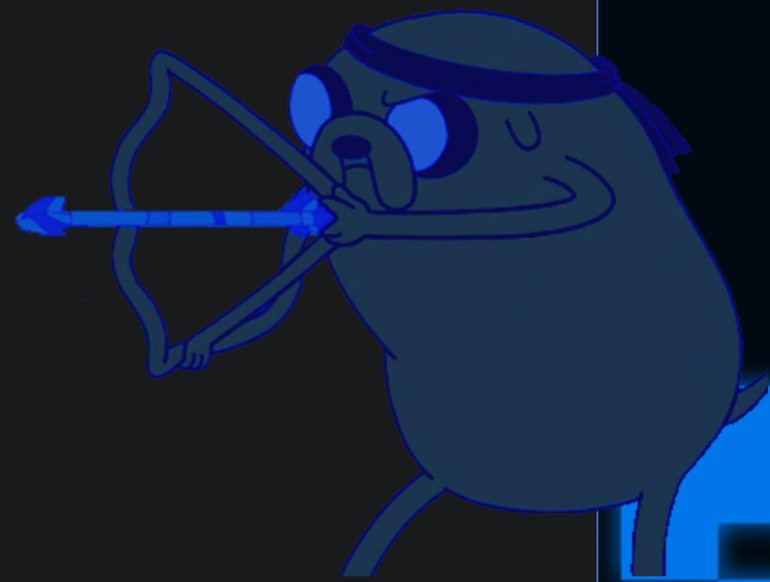
# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void Execute()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```



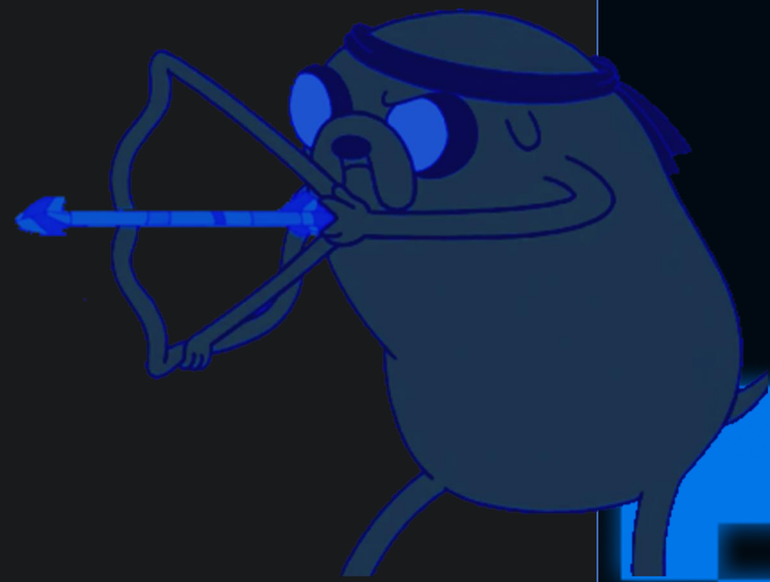
# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void Execute()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```



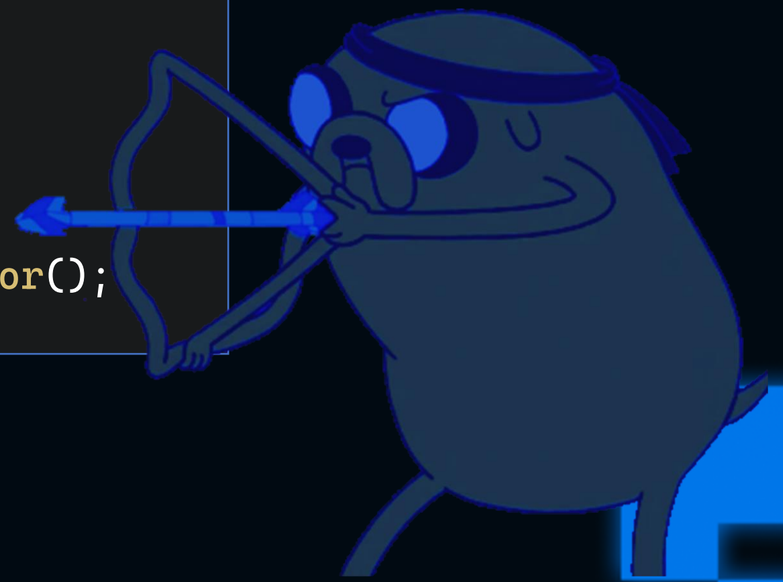
# Фреймворк-нинзя

```
public class Check<TProvider, TCategory, TRequirement> : ITest
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void Execute()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```



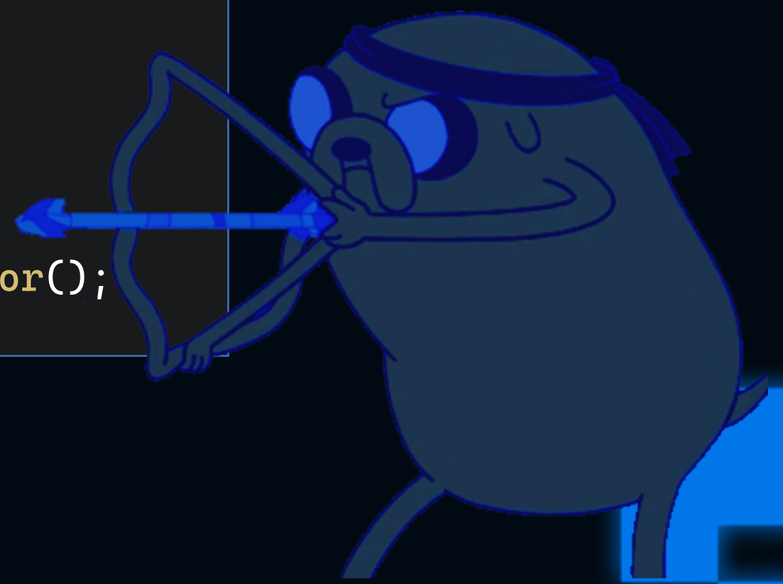
# Фреймворк-нинзя

```
public abstract class TestsSource : IEnumerable<ITest>
{
    protected abstract IEnumerable<ITest> Tests { get; }

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

    public IEnumerator<ITest> GetEnumerator() => Tests.GetEnumerator();
}
```



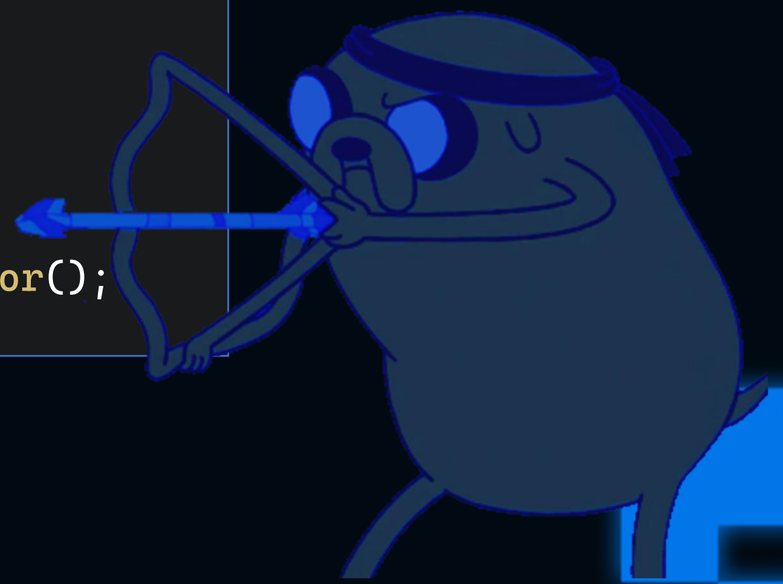
# Фреймворк-нинзя

```
public abstract class TestsSource : IEnumerable<ITest>
{
    protected abstract IEnumerable<ITest> Tests { get; }

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

    public IEnumerator<ITest> GetEnumerator() => Tests.GetEnumerator();
}
```



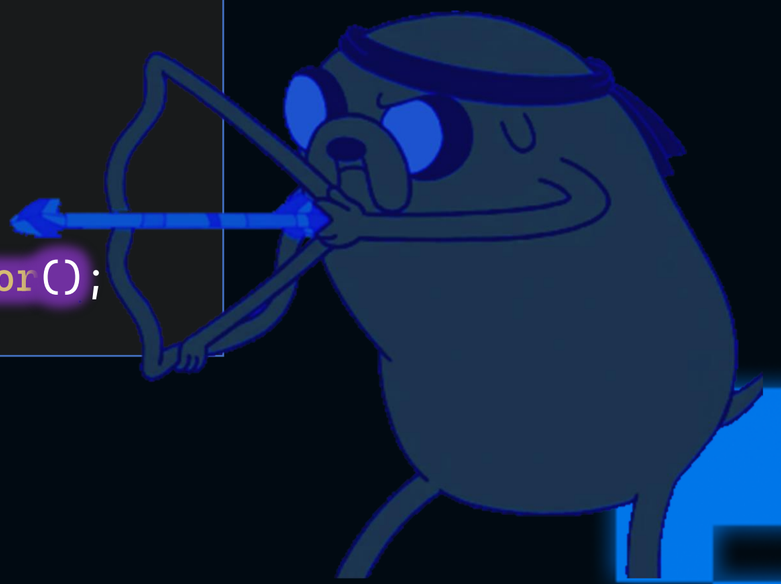
# Фреймворк-нинзя

```
public abstract class TestsSource : IEnumerable<ITest>
{
    protected abstract IEnumerable<ITest> Tests { get; }

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

    public IEnumerator<ITest> GetEnumerator() => Tests.GetEnumerator();
}
```



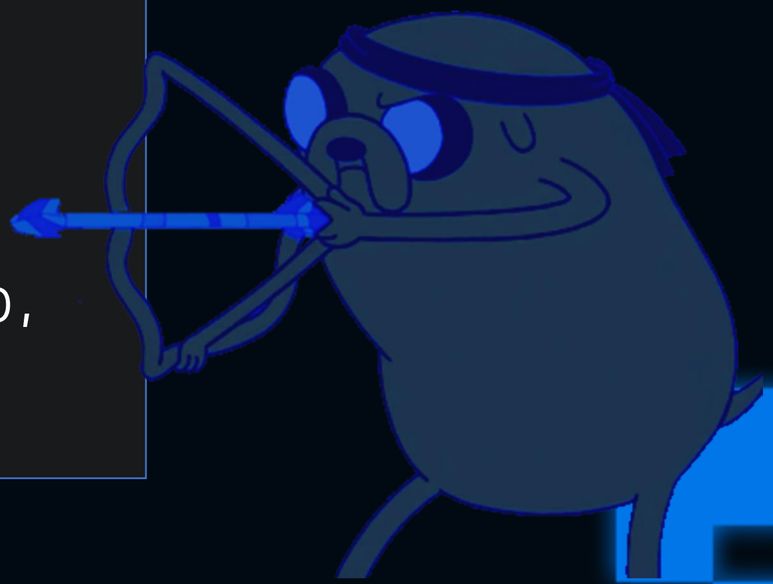
# Фреймворк-нинзя

```
public abstract class TestsSource : IEnumerable<ITest>
{
    protected abstract IEnumerable<ITest> Tests { get; }

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

    public IEnumerator<ITest> GetEnumerator() => Tests.GetEnumerator();
}
```



# Фреймворк-нинзя

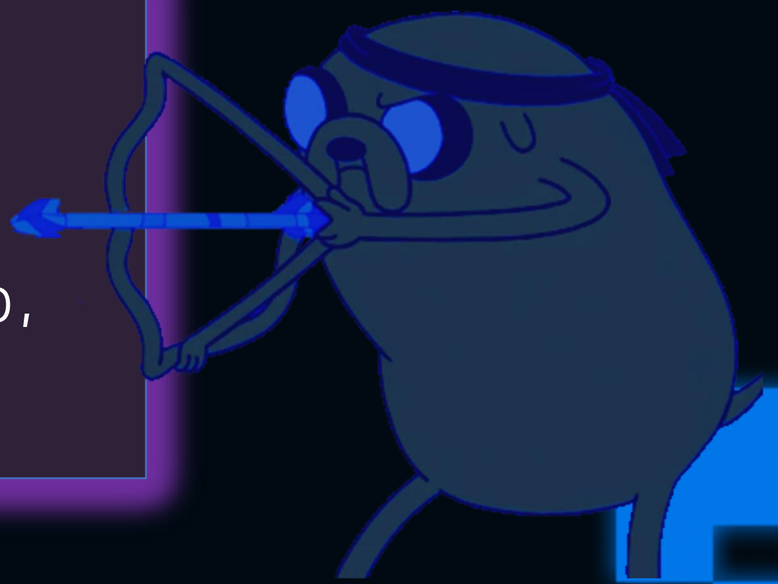
```
public class AllTests : TestsSource
{
    protected override IEnumerable<ITest> Tests => new ITest[]
    {
        new Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<RussianJokes, ForParty, AreOfMinimalFun>(76),
        new Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75),
    };
}
```



```
[TestCaseSource(typeof(AllTests))]
public void Test(ITest test) => test.Execute();
```

# Фреймворк-нинзя

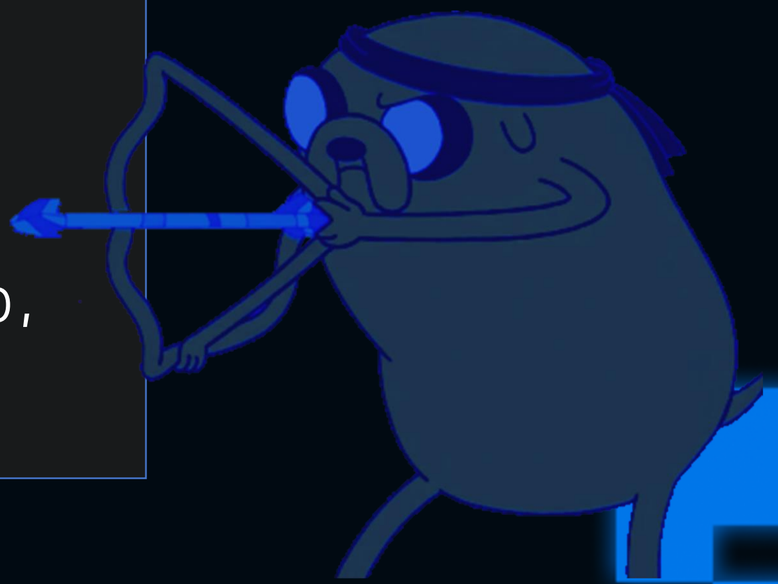
```
public class AllTests : TestsSource
{
    protected override IEnumerable<ITest> Tests => new ITest[]
    {
        new Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<RussianJokes, ForParty, AreOfMinimalFun>(76),
        new Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75),
    };
}
```



```
[TestCaseSource(typeof(AllTests))]
public void Test(ITest test) => test.Execute();
```

# Фреймворк-нинзя

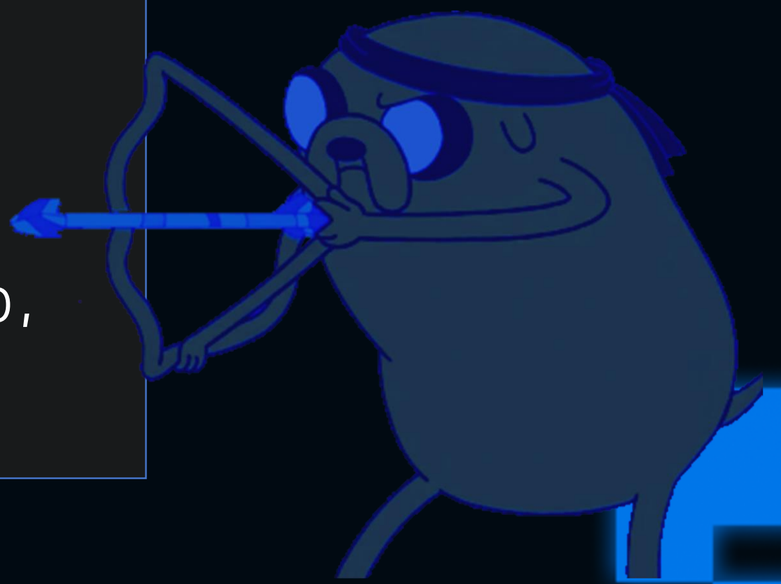
```
public class AllTests : TestsSource
{
    protected override IEnumerable<ITest> Tests => new ITest[]
    {
        new Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<RussianJokes, ForParty, AreOfMinimalFun>(76),
        new Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75),
    };
}
```



```
[TestCaseSource(typeof(AllTests))]
public void Test(ITest test) => test.Execute();
```

# Фреймворк-нинзя

```
public class AllTests : TestsSource
{
    protected override IEnumerable<ITest> Tests => new ITest[]
    {
        new Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<RussianJokes, ForParty, AreOfMinimalFun>(76),
        new Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75),
        new Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75),
    };
}
```



```
[TestCaseSource(typeof(AllTests))]
public void Test(ITest test) => test.Execute();
```

# Фреймворк-нинзя

```
public cl  Te  s : TestsSource
{
    prot  ov  de Enu erable<ITest> Tests => new ITest[]
    {
        new Ch  k<Rus  ar  K  bou  s, AreOfMinimalFun>(75),
        new  k<Ru  ia  es  or  rty, reOfMinimalFun>(76),
        new Chec  ibralarJok  , outC  ers, eOfMinimalFun>(75),
             <Gib  rJo  s, orPa  y, Mini  lFun>(75),
    }
}
```

```
CaseSource(typeof(All  ts))]
 void Test(ITest test) =>  .E  e  ();
```

#НИПАНЯТНА

# Безумный Демиург

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = false)]
public class RequirementAttribute : Attribute
{
    public int Value { get; }

    public RequirementAttribute(int value) => Value = value;
}
```

# Безумный Демиург

```
public abstract class Check<TProvider, TCategory, TRequirement>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public Check() =>
        value = GetType().GetCustomAttributes(false)
            .OfType<RequirementAttribute>().Single().Value;

    // ...
}
```

# Безумный Демиург

```
public abstract class Check<TProvider, TCategory, TRequirement>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    [Test]
    public void Test()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```

# Безумный Демиург

```
[Requirement(75)]
public class Test1 : Check<RussianJokes, AboutCoders, AreOfMinimalFun> { }

[Requirement(76)]
public class Test2 : Check<RussianJokes, ForParty, AreOfMinimalFun> { }

[Requirement(75)]
public class Test3 : Check<GibraltarJokes, AboutCoders, AreOfMinimalFun> { }

[Requirement(75)]
public class Test4 : Check<GibraltarJokes, ForParty, AreOfMinimalFun> { }
```

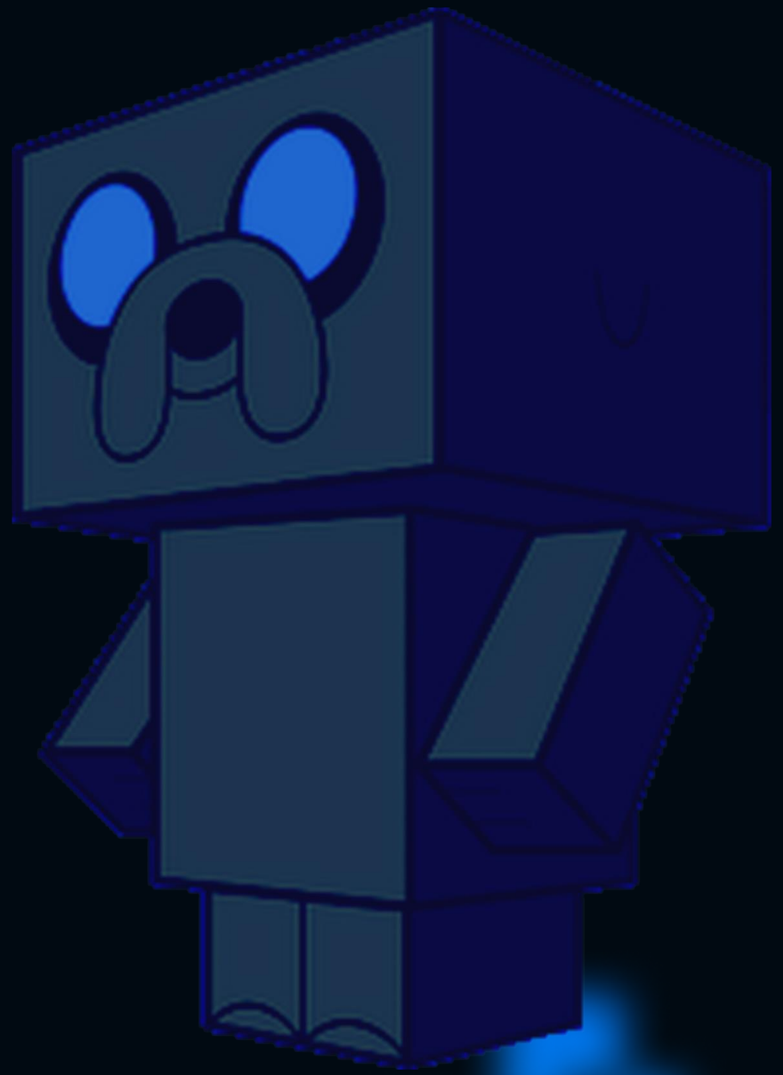
# Безумный Демиург

#БЕЗ КОММЕНТАРИЕВ

```
[Requirement(75)]
public class Test1 : Check<Russia
[Requirement(76)]
public class Test2 : Check<Russia
[Requirement(75)]
public class Test3 : Check<GibraltarJokes, AboutCoders, AreOfMinima
[Requirement(75)]
public class Test4 : Check<GibraltarJokes, ForParty, AreOfMinimal
```

# Поехали дальше ...

# А что есть не из коробки?



# Обобщенный метод

```
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.Ge opJ

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный метод

```
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.Ge opJ

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный метод

```
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.Ge opJ

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный метод

```
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.Ge opJ

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный метод

```
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.Ge opJ

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный метод

```
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.Ge opJ

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный атрибут

```
public class GCaseAttribute : TestCaseAttribute
{
    public GCaseAttribute(params object[] arguments) :
        base(arguments)
    {
    }
}
```

```
public class GCaseAttribute<T1, T2, T3> : GCaseAttribute
{
    public GCaseAttribute(params object[] arguments) :
        base(arguments)
    {
    }
}
```

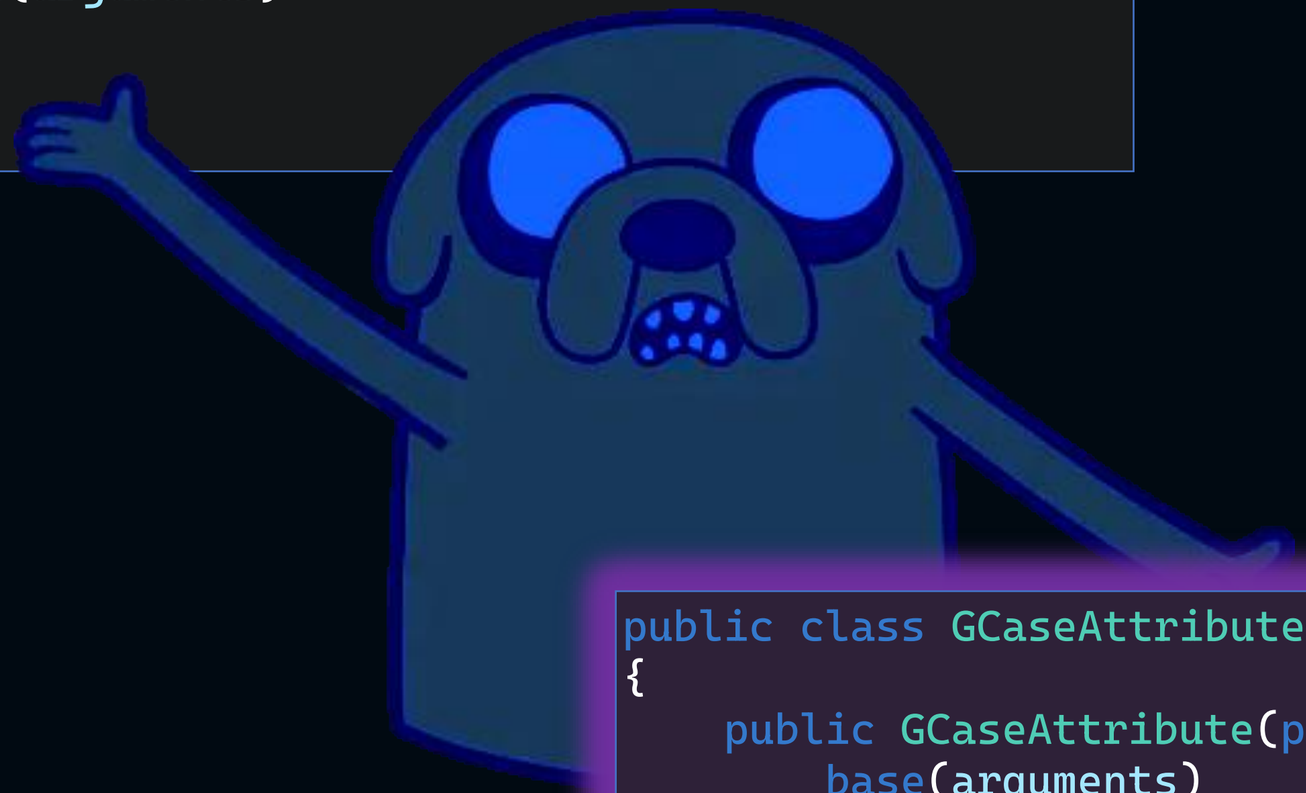
# Обобщенный атрибут

```
public class GCaseAttribute : TestCaseAttribute
{
    public GCaseAttribute(params object[] arguments) :
        base(arguments)
    {
    }
}
```

```
public class GCaseAttribute<T1, T2, T3> : GCaseAttribute
{
    public GCaseAttribute(params object[] arguments) :
        base(arguments)
    {
    }
}
```

# Обобщенный атрибут

```
public class GCaseAttribute : TestCaseAttribute
{
    public GCaseAttribute(params object[] arguments) :
        base(arguments)
    {
    }
}
```



```
public class GCaseAttribute<T1, T2, T3> : GCaseAttribute
{
    public GCaseAttribute(params object[] arguments) :
        base(arguments)
    {
    }
}
```

# Обобщенный тест

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Обобщенный тест

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# И снова запустим тестирование

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
```

Что будет дальше?

Проект не соберется

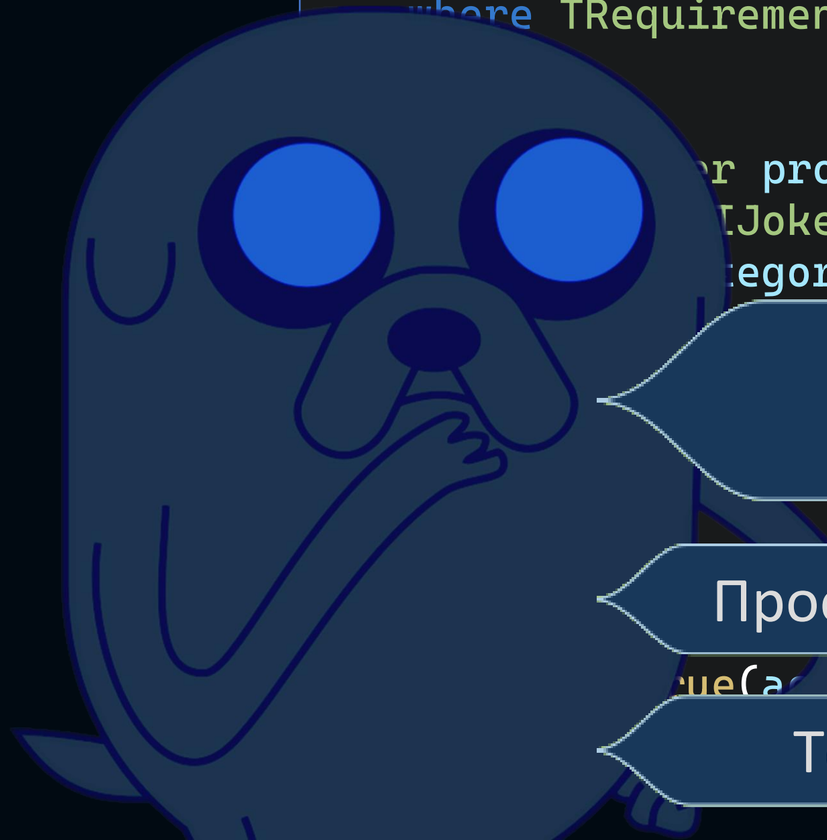
Тесты упадут

Тесты пройдут

Зависит от...

# И снова запустим тестирование

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()

    r provider = new TProvider();
    IJoke> jokes = provider.GetJokes();
    tegory = new TCategory();
```



Что будет дальше?

Проект не соберется

Тесты упадут

Тесты пройдут

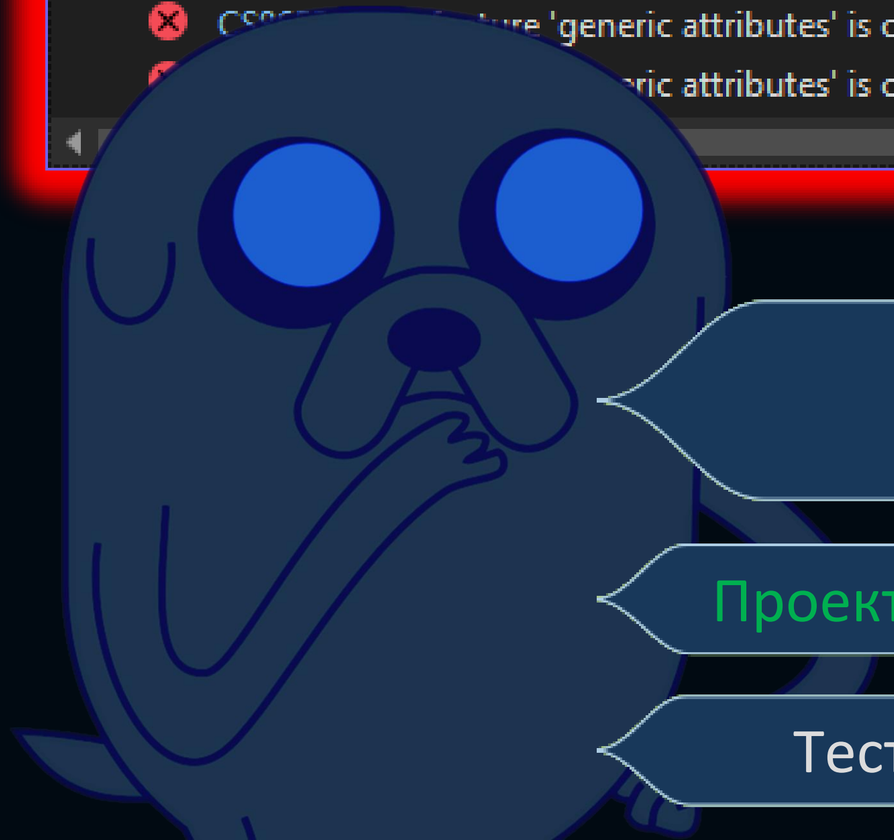
Зависит от...

# Версия C# <= 10

Error List

Entire Solution | 5 Errors | 0 Warnings | 0 of 1 Message | Search Error List

| Code | Description |
|---|---|
| CS8652 | The feature 'generic attributes' is currently in Preview and *unsupported*. To use Preview features, use the 'preview' language version. |
| CS8652 | The feature 'generic attributes' is currently in Preview and *unsupported*. To use Preview features, use the 'preview' language version. |
| CS8652 | The feature 'generic attributes' is currently in Preview and *unsupported*. To use Preview features, use the 'preview' language version. |
| CS8652 | ure 'generic attributes' is currently in Preview and *unsupported*. To use Preview features, use the 'preview' language version. |
| | ric attributes' is currently in Preview and *unsupported*. To use Preview features, use the 'preview' language version. |



Что будет дальше?

- Проект не соберется
- Тесты упадут
- Тесты пройдут
- Зависит от...

# Версия C# > 10

Test Explorer

1 0 1

Search (Ctrl+I)

Test run finished: 4 Tests (0 Passed, 4 Failed, 0 Skipped) run in 292 ms

0 Warnings 0 Errors

Test

- Jokes.Tests (1)
  - Jokes.Tests (1)
    - IntegrationTests (1)
      - Test<TProvider,TCategory,TRequirement>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequirement>(75)

Source: IntegrationTests.cs line 16

Test has multiple result outcomes

4 Failed

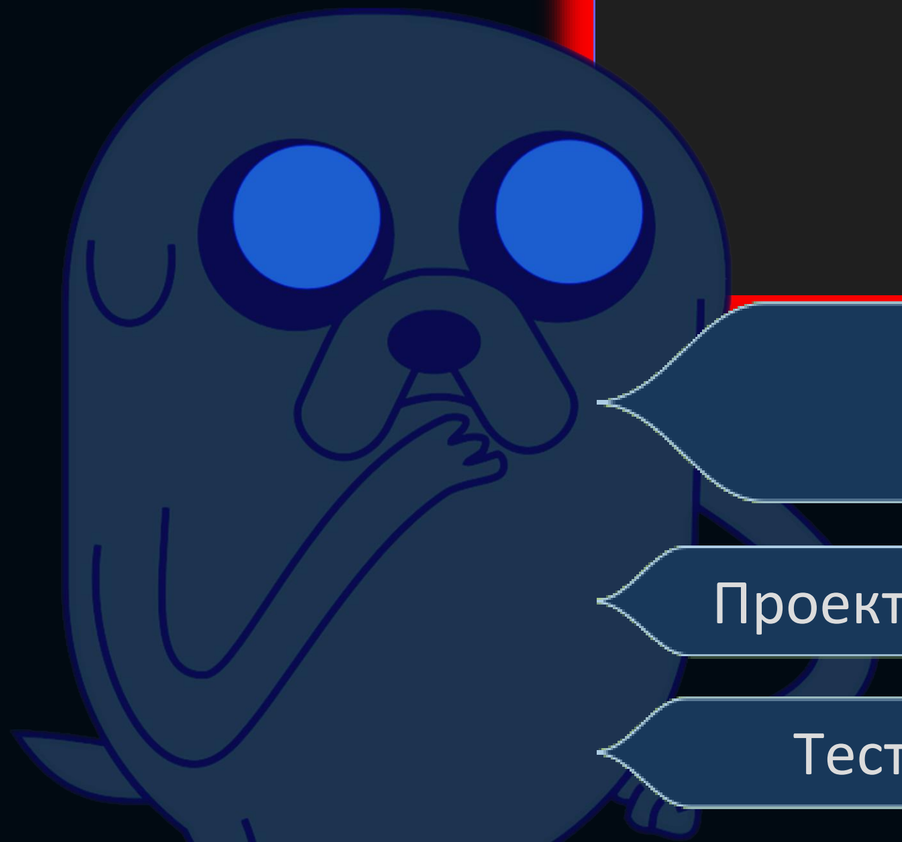
Results

1) Test<TProvider,TCategory,TRequirement>(75)

Duration: 2 ms

Message:

```
Unable to determine type arguments for method
```



Что будет дальше?

Проект не соберется

Тесты упадут

Тесты пройдут

Зависит от...

# А как же заставить это работать?

???

Что будет дальше?

Проект не соберется

Тесты упадут

Тесты пройдут

Зависит от...

# Лезем в документацию

```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited=false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
```

```
/// <summary>
/// The ITestBuilder interface is exposed by a class that knows how to
/// build tests from a specified method. In general, it is exposed
/// by an attribute which has additional information available to provide
/// the necessary test parameters to distinguish the test cases built.
/// </summary>
public interface ITestBuilder
{
    /// <summary>
    /// Builds any number of tests from the specified method an
    /// </summary>
    /// <param name="method">The method to be used as a test</param>
    /// <param name="suite">The TestSuite to which the method will be added</param>
    IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite);
}
```

DOTNEXT

# Лезем в документацию

```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited=false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
```

```
/// <summary>
/// The ITestBuilder interface is exposed by a class that knows how to
/// build tests from a specified method. In general, it is exposed
/// by an attribute which has additional information available to provide
/// the necessary test parameters to distinguish the test cases built.
/// </summary>
public interface ITestBuilder
{
    /// <summary>
    /// Builds any number of tests from the specified method an
    /// </summary>
    /// <param name="method">The method to be used as a test</param>
    /// <param name="suite">The TestSuite to which the method will be added</param>
    IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite);
}
```

# Лезем в документацию

```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited=false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
```

```
/// <summary>
/// The ITestBuilder interface is exposed by a class that knows how to
/// build tests from a specified method. In general, it is exposed
/// by an attribute which has additional information available to provide
/// the necessary test parameters to distinguish the test cases built.
/// </summary>
public interface ITestBuilder
{
    /// <summary>
    /// Builds any number of tests from the specified method an
    /// </summary>
    /// <param name="method">The method to be used as a test</param>
    /// <param name="suite">The TestSuite to which the method will be added</param>
    IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite);
}
```

# Лезем в документацию

```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited=false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
```

```
/// <summary>
/// The ITestBuilder interface is exposed by a class that knows how to
/// build tests from a specified method. In general, it is exposed
/// by an attribute which has additional information available to provide
/// the necessary test parameters to distinguish the test cases built.
/// </summary>
public interface ITestBuilder
{
    /// <summary>
    /// Builds any number of tests from the specified method an
    /// </summary>
    /// <param name="method">The method to be used as a test</param>
    /// <param name="suite">The TestSuite to which the method will be added</param>
    IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite);
}
```

DOTNEXT

# Лезем в ~~документацию~~ Google

Google

nunit generic test

Все | Картинки | Видео | Покупки | Новости | Ещё | Инструменты

Результатов: примерно 142 000 (0,25 сек.)

Совет. По этому запросу вы можете найти сайты **на русском языке**. Указать предпочтительные языки для результатов поиска можно в разделе Настройки.

https://stackoverflow.com › nu... ▾ Перевести эту страницу

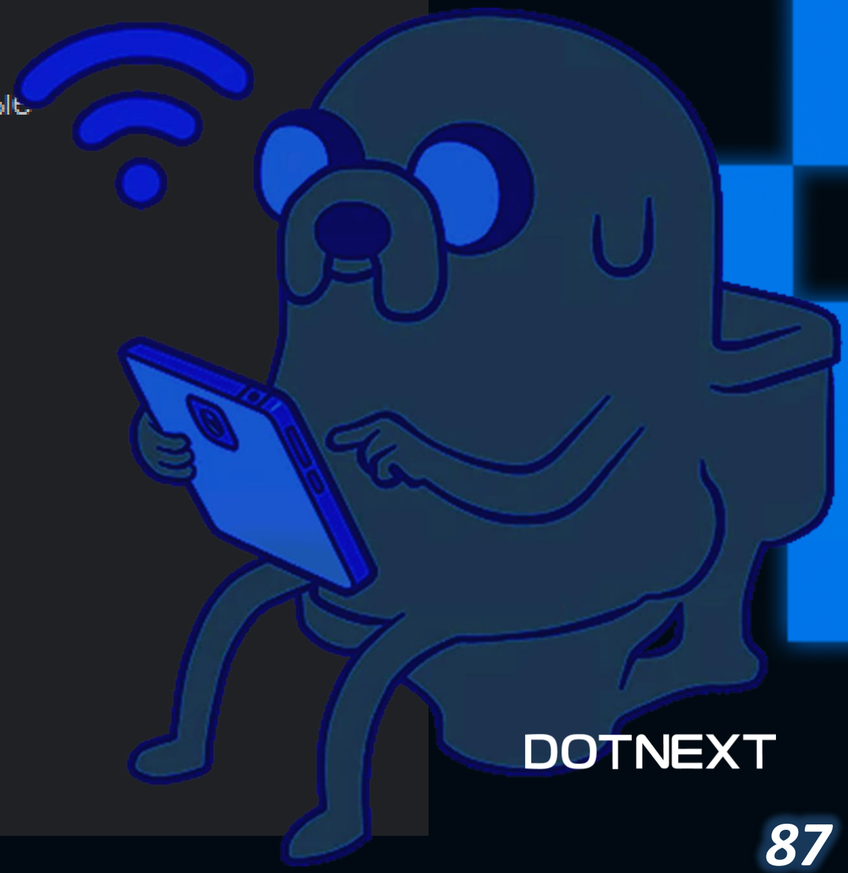
## NUnit TestCase with Generics - Stack Overflow

6 нояб. 2011 г. · 9 ответов

**NUnit test** methods actually can be **generic** as long as the **generic** type arguments can be inferred from parameters: [TestCase(42)] [TestCase("string")] ...

| | |
|---|---|
| How to make a **generic test** with **NUnit** (C#)? - Stack Overflow | 30 янв. 2018 г. |
| c# - How to use **NUnit** TestCases with Generics with where ... | 26 мар. 2021 г. |
| c# - Unit **Test** with Generics types - Stack Overflow | 5 янв. 2019 г. |
| .net - How to write **nunit test** case(c#) for **generic** class which ... | 2 апр. 2021 г. |

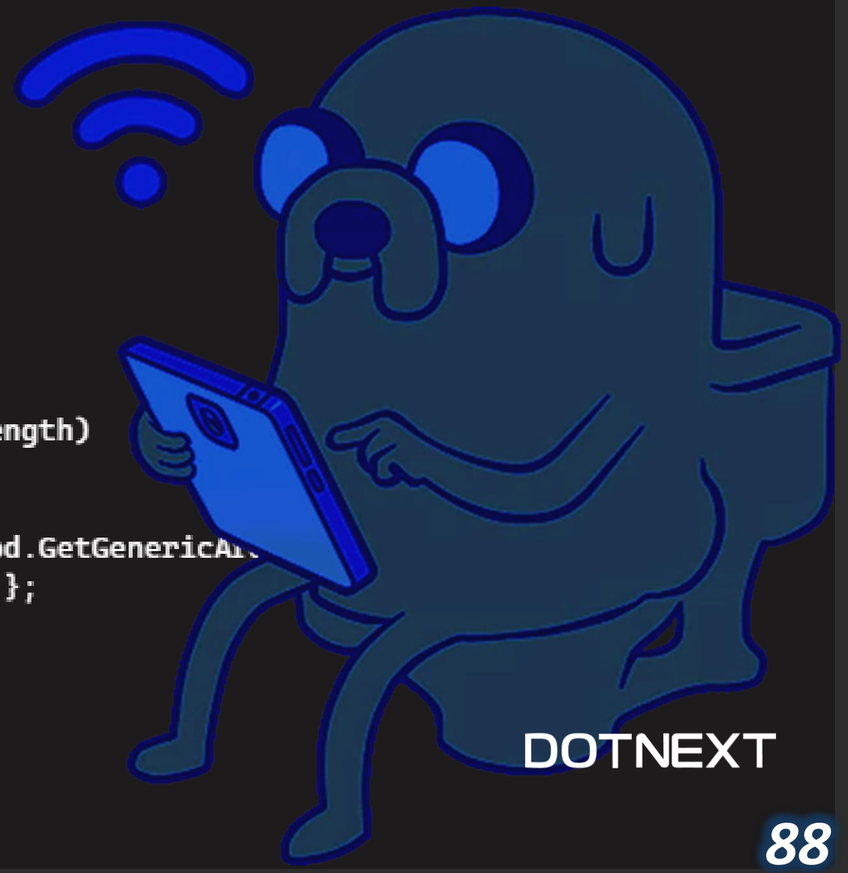
Другие результаты с сайта stackoverflow.com



# Лезем в ~~документацию~~ Google

But in this case you can implement a custom attribute:

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
public class TestCaseGenericAttribute : TestCaseAttribute, ITestBuilder
{
    public TestCaseGenericAttribute(params object[] arguments)
        : base(arguments)
    {
    }

    public Type[] TypeArguments { get; set; }

    IEnumerable<TestMethod> ITestBuilder.BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        if (TypeArguments == null || TypeArguments.Length != method.GetGenericArguments().Length)
        {
            var parms = new TestCaseParameters { RunState = RunState.NotRunnable };
            parms.Properties.Set("_SKIPREASON", $"{nameof(TypeArguments)} should have {method.GetGenericAr
            return new[] { new NUnitTestCaseBuilder().BuildTestMethod(method, suite, parms) };
        }

        var genMethod = method.MakeGenericMethod(TypeArguments);
        return base.BuildFrom(genMethod, suite);
    }
}
```



# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

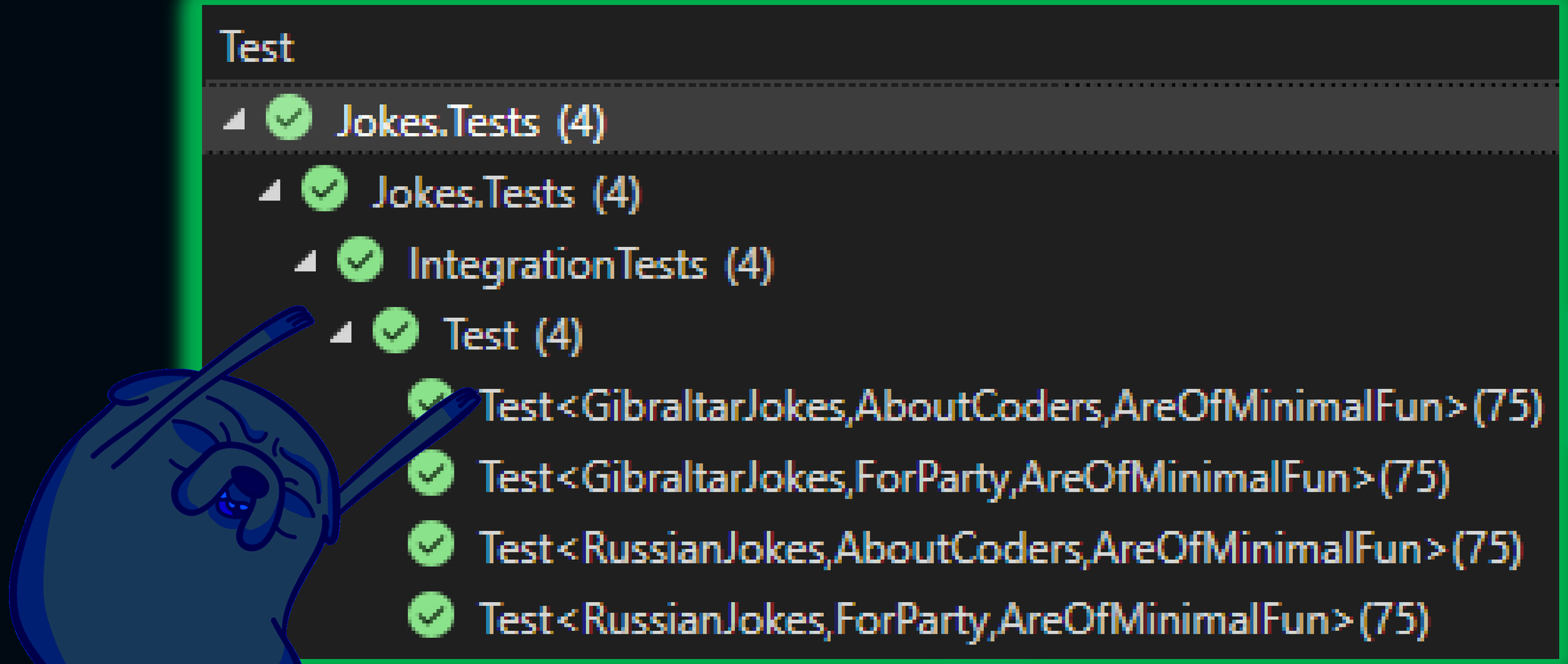
# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# Реализуем интерфейс

```
public class GCaseAttribute : TestCaseAttribute, ITestBuilder
{
    private readonly Type[] typeArguments;

    public GCaseAttribute(params object[] arguments) :
        base(arguments) =>
        typeArguments = GetType().GetGenericArguments();

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        if (!method.IsGenericMethodDefinition)
            return base.BuildFrom(method, suite);

        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
}
```

# И всё работает!!!



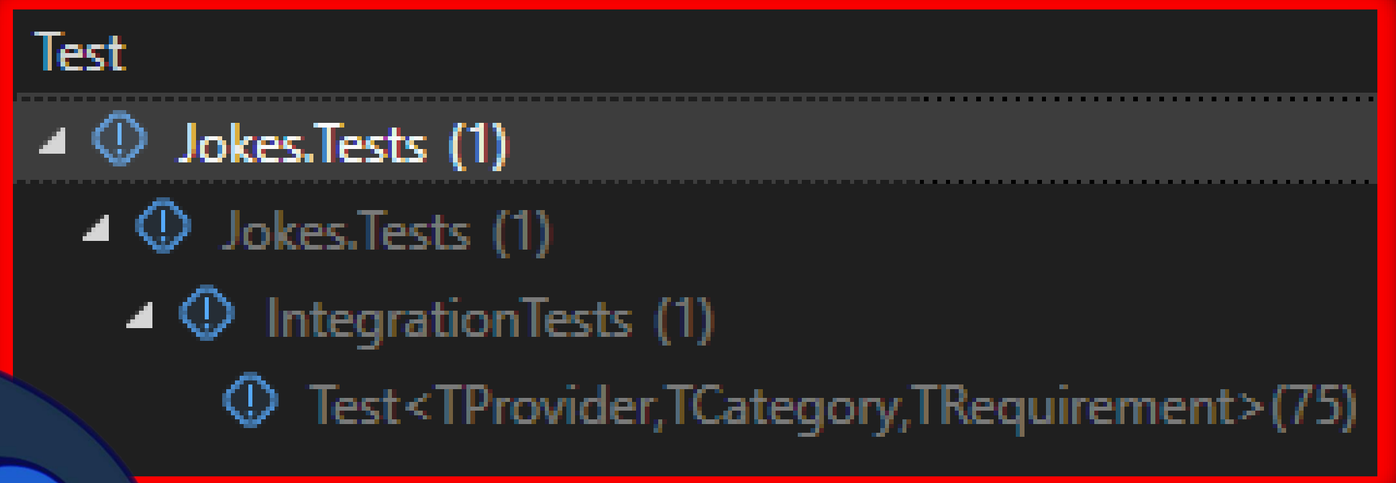
# Вернёмся к вечному вопросу

```
[GCase<object, object, object>(75)]
```

Что сделает этот тест?

- Ошибку компиляции
- Сломает Test Explorer
- Упадет
- Пройдет

# Вернёмся к вечному вопросу




Что сделает этот тест?

Ошибку компиляции

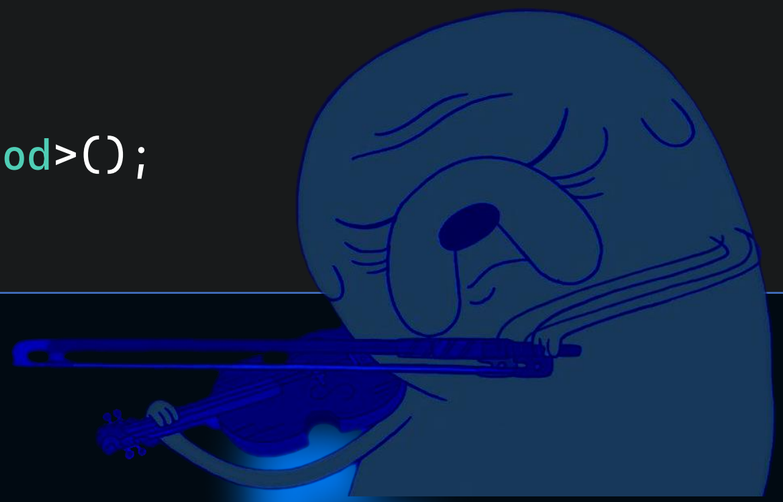
Сломает Test Explorer

Упадет

Пройдет

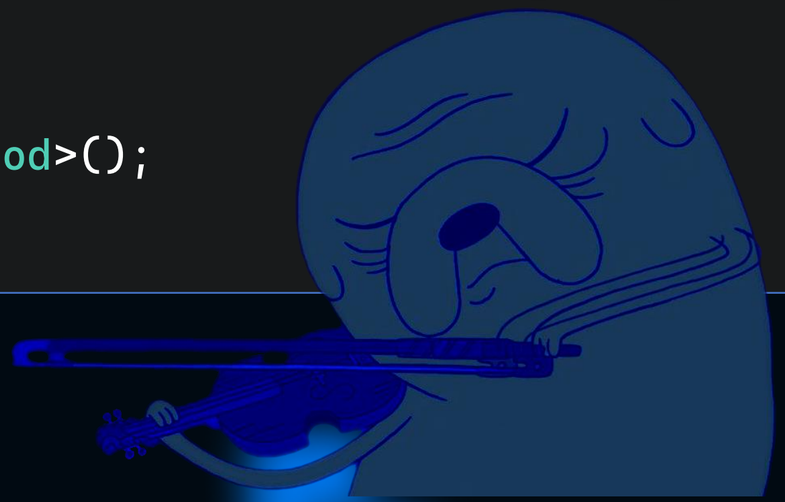
# Добавим обработку исключений

```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return Array.Empty<TestMethod>();
    }
}
```



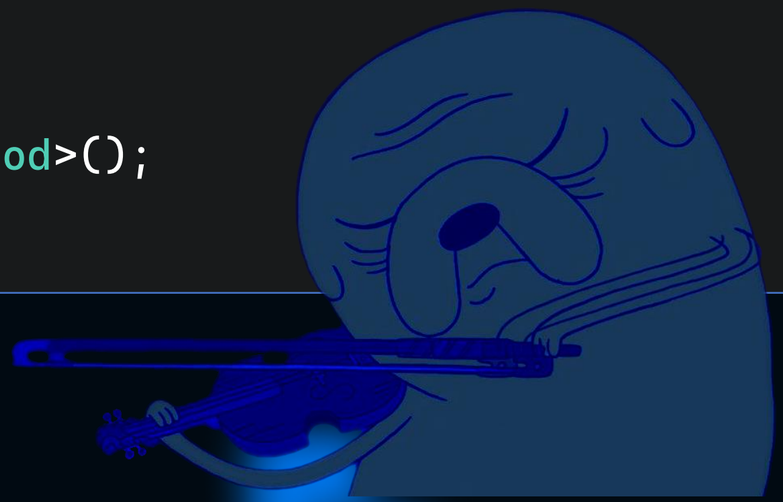
# Добавим обработку исключений

```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return Array.Empty<TestMethod>();
    }
}
```



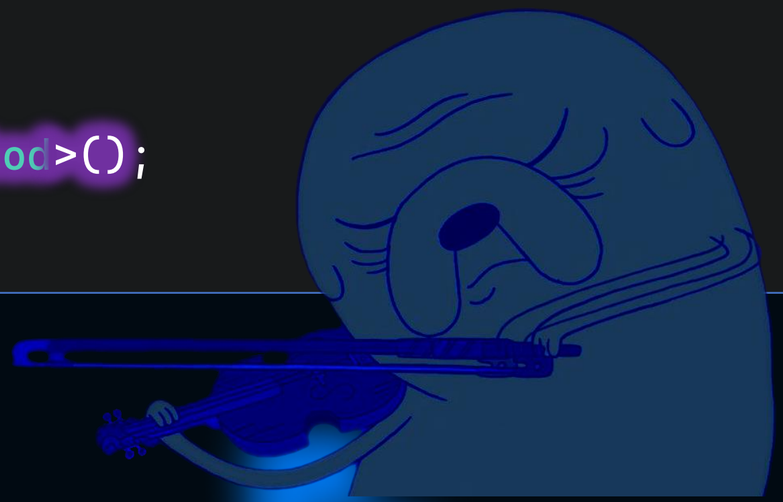
# Добавим обработку исключений

```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return Array.Empty<TestMethod>();
    }
}
```



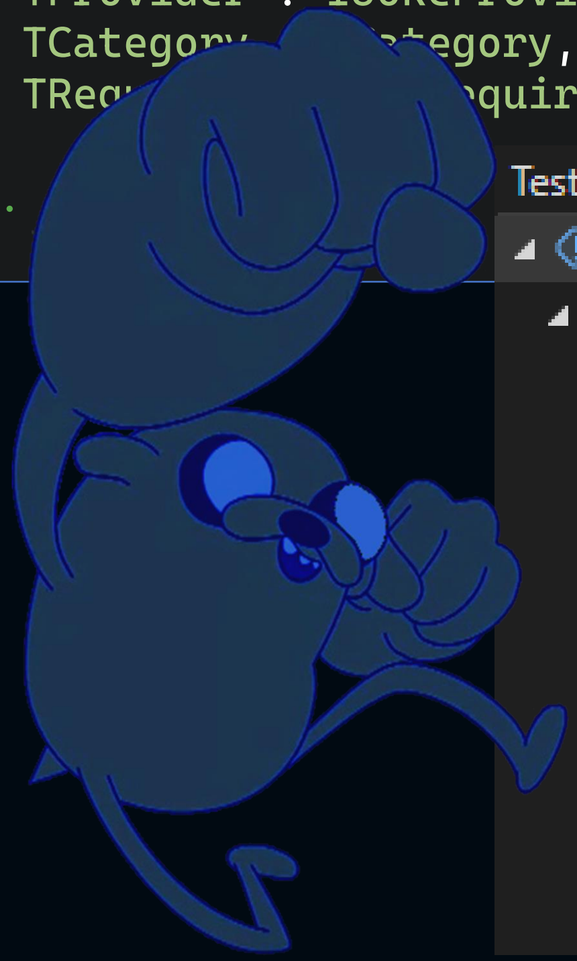
# Добавим обработку исключений

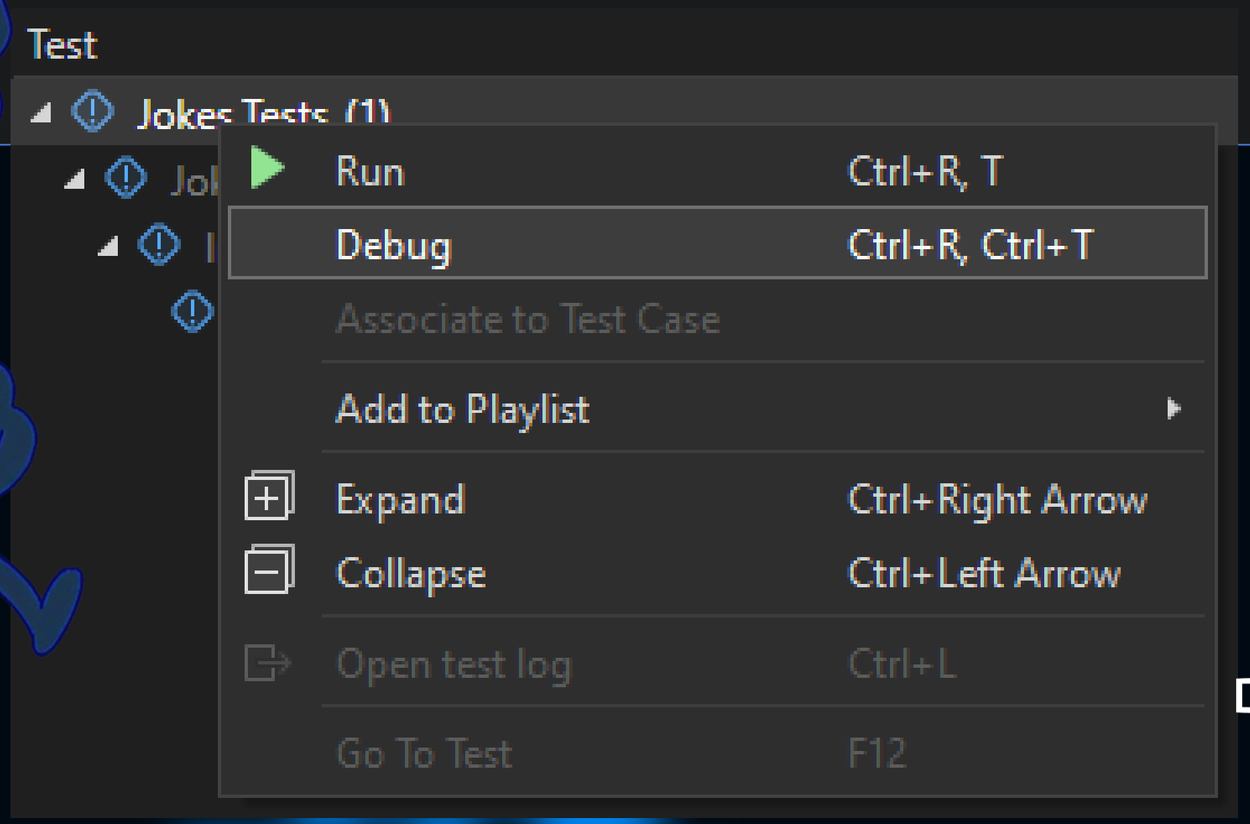
```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return Array.Empty<TestMethod>();
    }
}
```



# Дебажим!!!

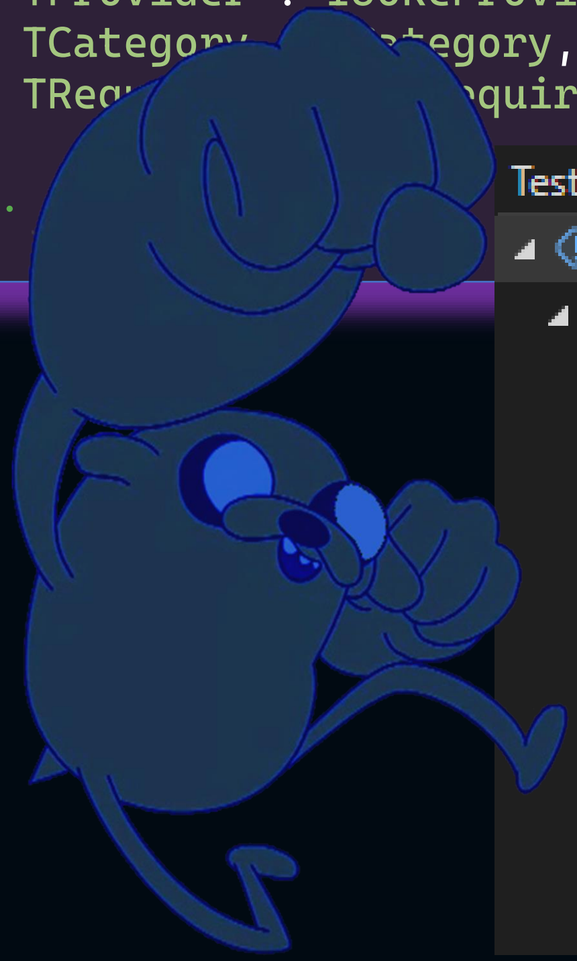
```
[GCase<object, object, object >(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

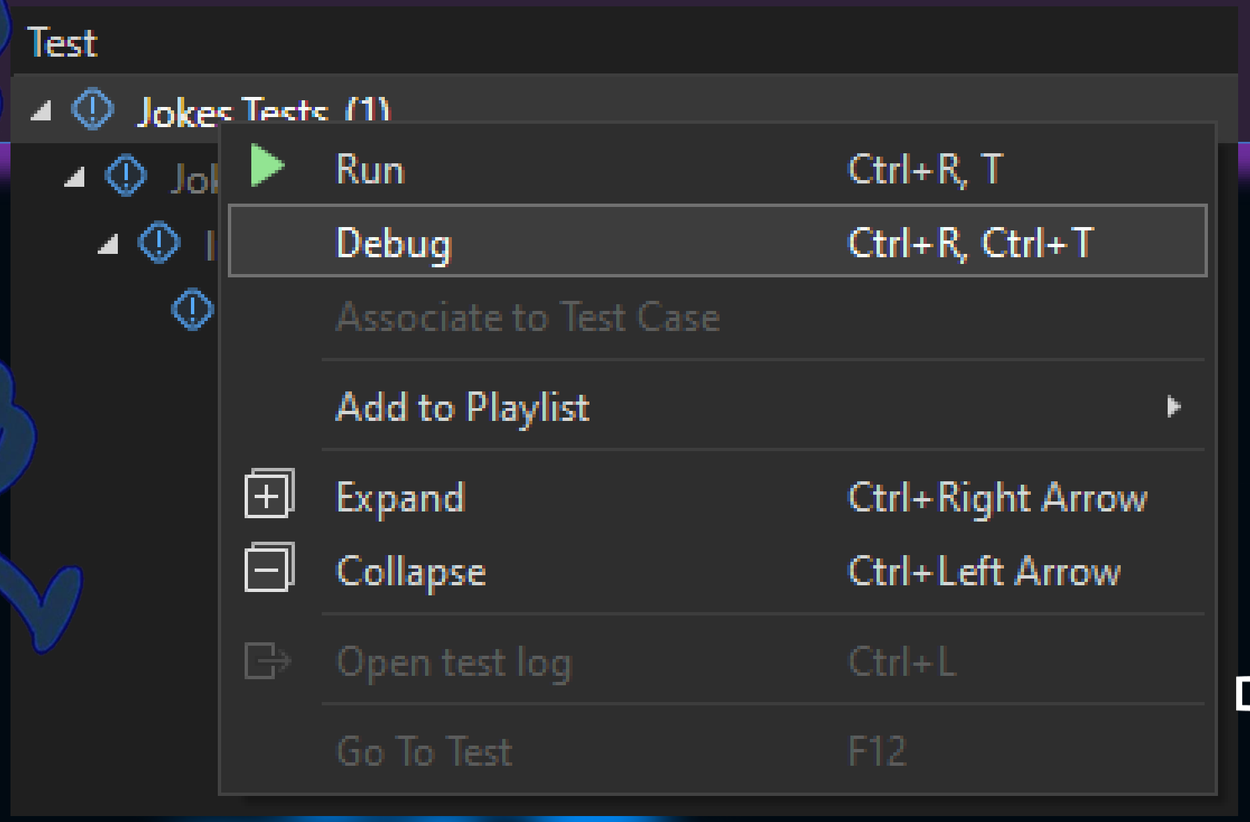




# Дебажим!!!

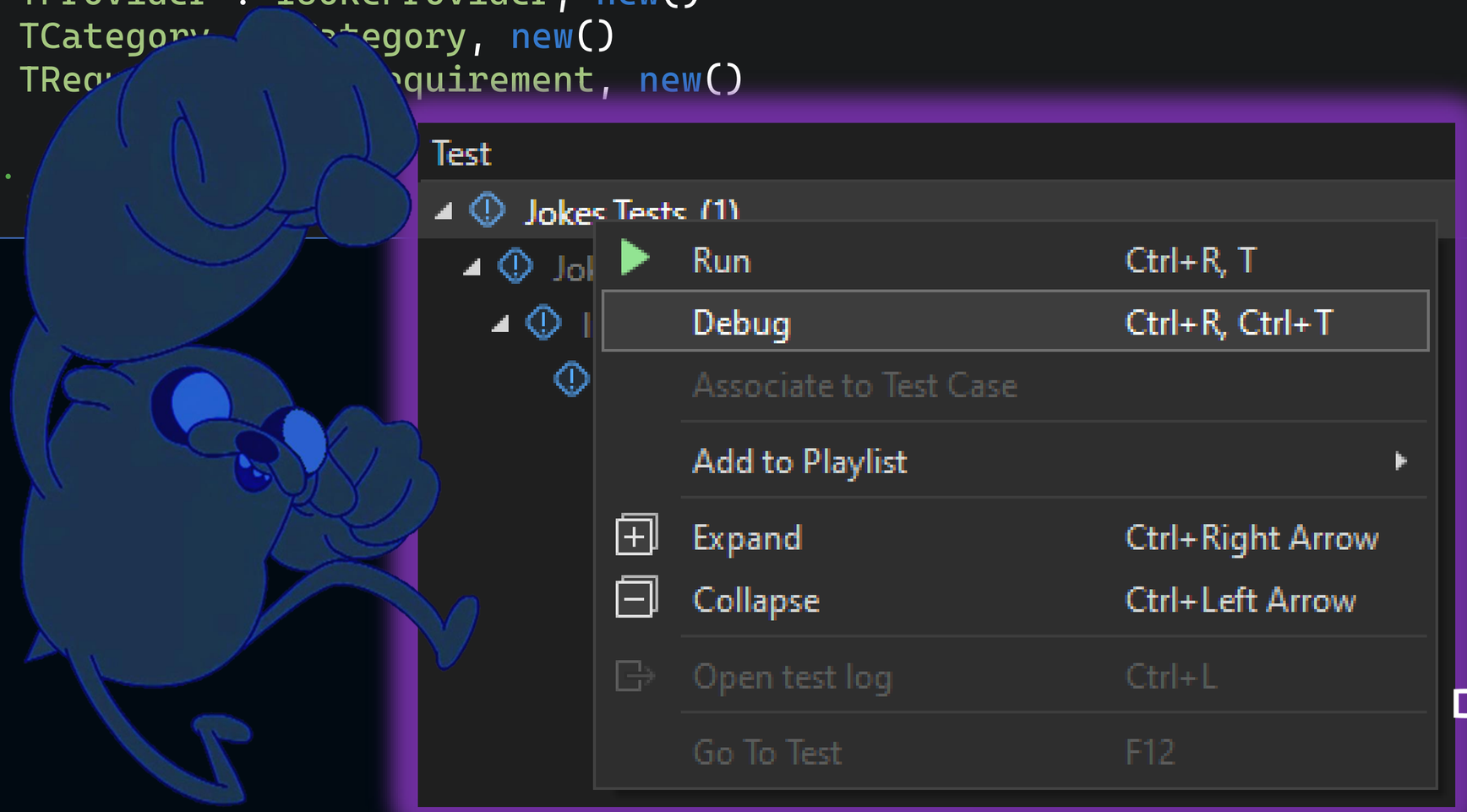
```
[GCase<object, object, object >(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ategory, new()
    where TReq      : equirement, new()
{
    // ...
}
```





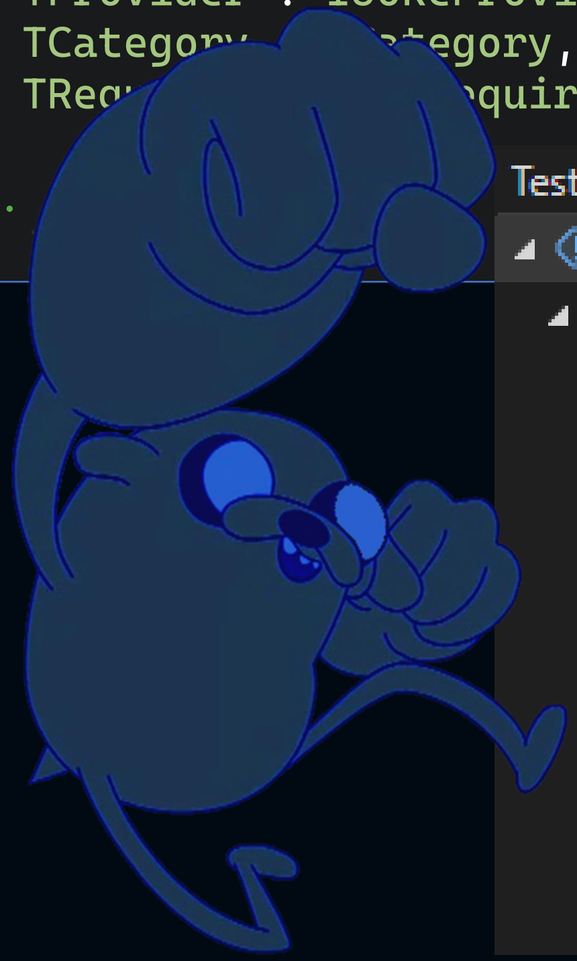
# Дебажим!!!

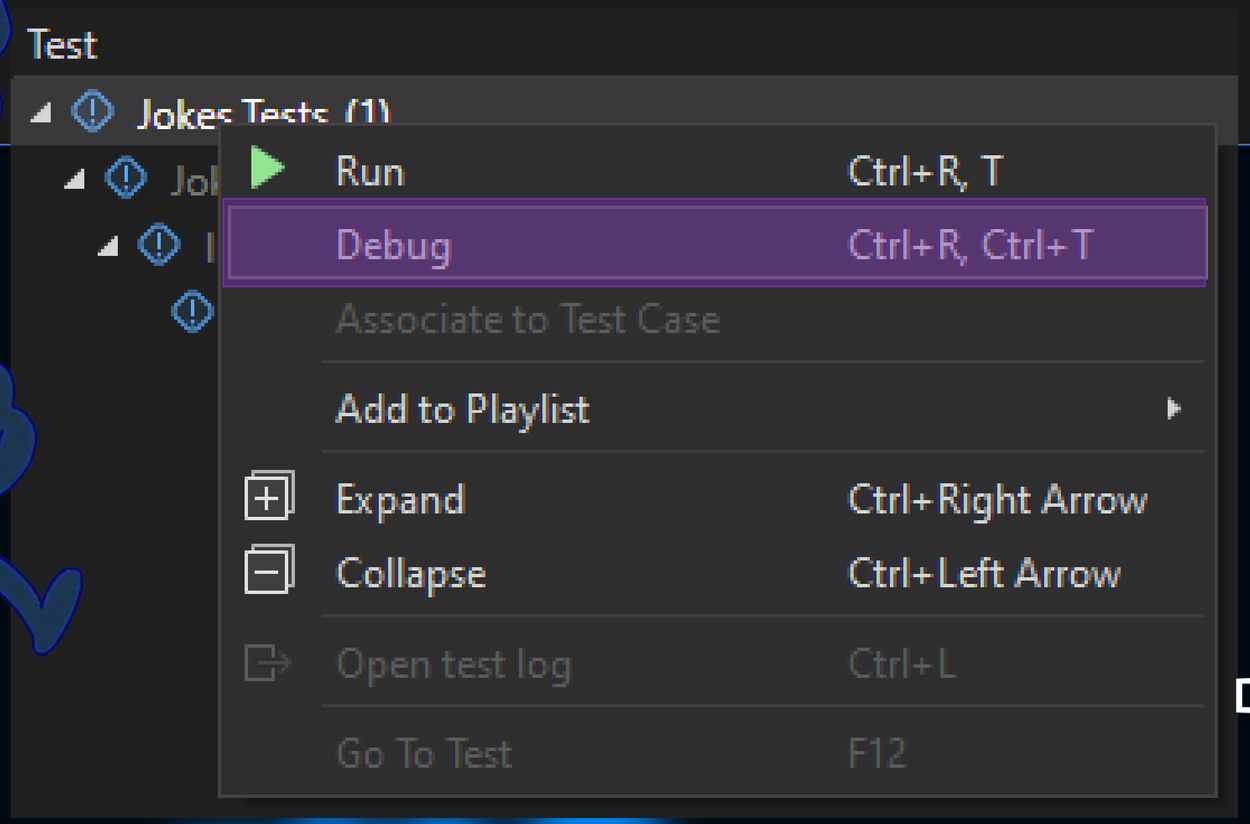
```
[GCase<object, object, object >(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```



# Дебажим!!!

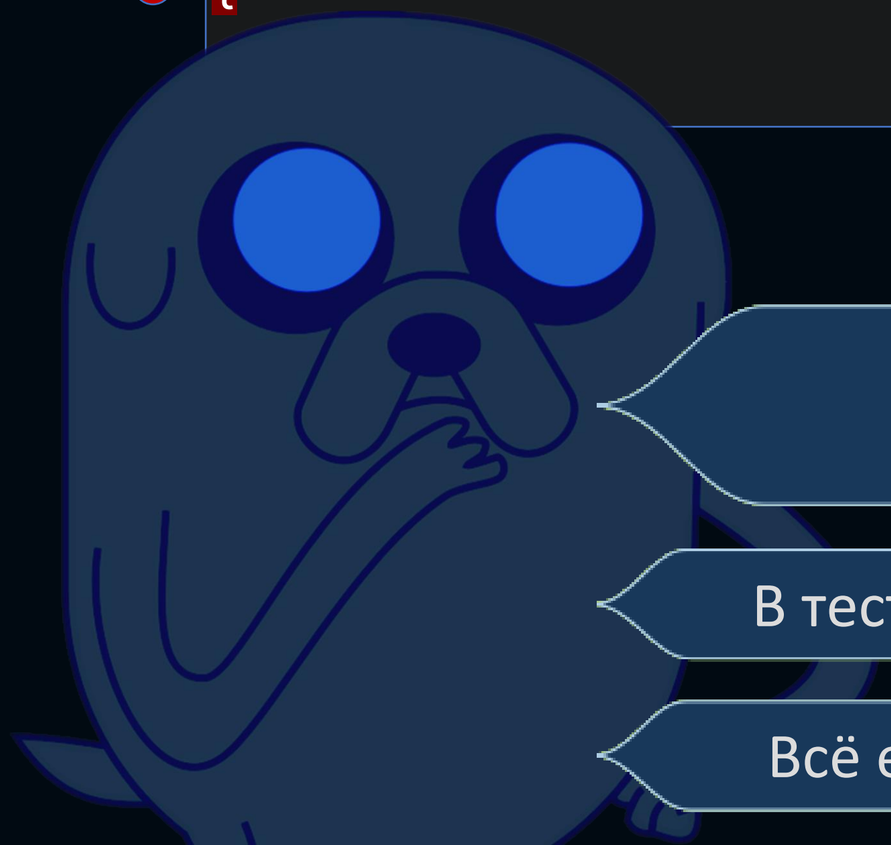
```
[GCase<object, object, object >(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ategory, new()
    where TReq      equirement, new()
{
    // ...
}
```





# Дебажим!!!

```
[GCase<object, object, object >(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
```



Куда нас это приведёт?

- В тестовый метод
- В конструктор атрибута
- Всё ещё никуда
- Куда-то ещё

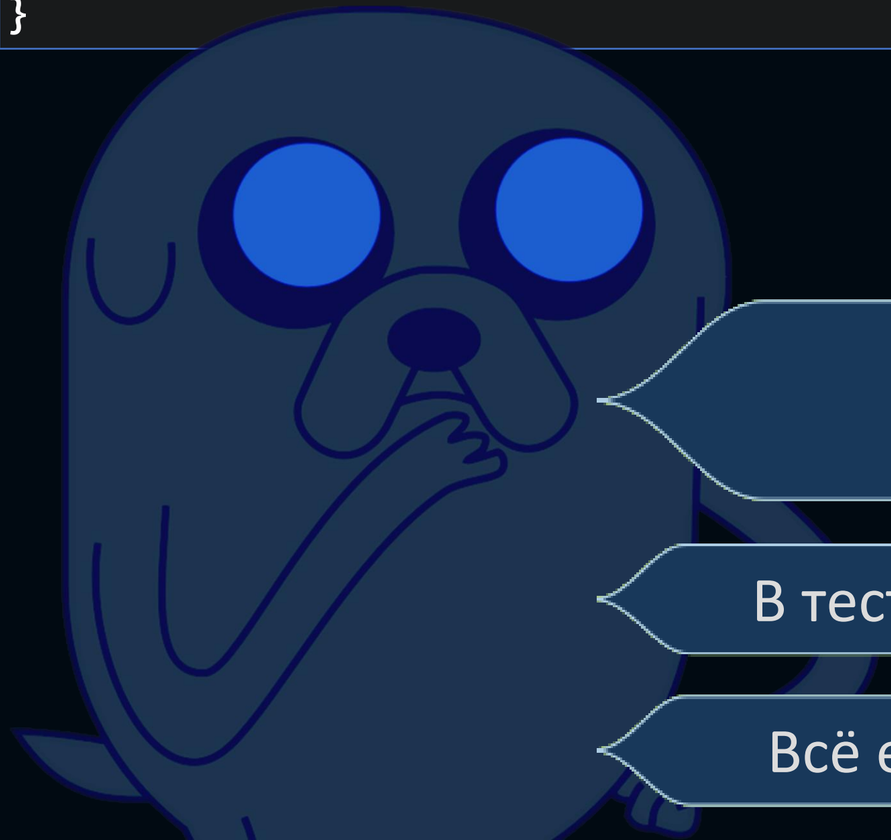
# Дебажим!!!

```
try
{
    return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
}
catch(Exception ex)
{
    return Array.Empty<TestMethod>();
}
```

Exception Thrown

**System.ArgumentException:** 'GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCategory,TRequirement](Int32)' violates the constraint of type 'TProvider'.'

Inner Exception
VerificationException: Method Jokes.Tests.IntegrationTests.Test: type argument 'System.Object' violates the constraint of type parameter 'TProvider'.

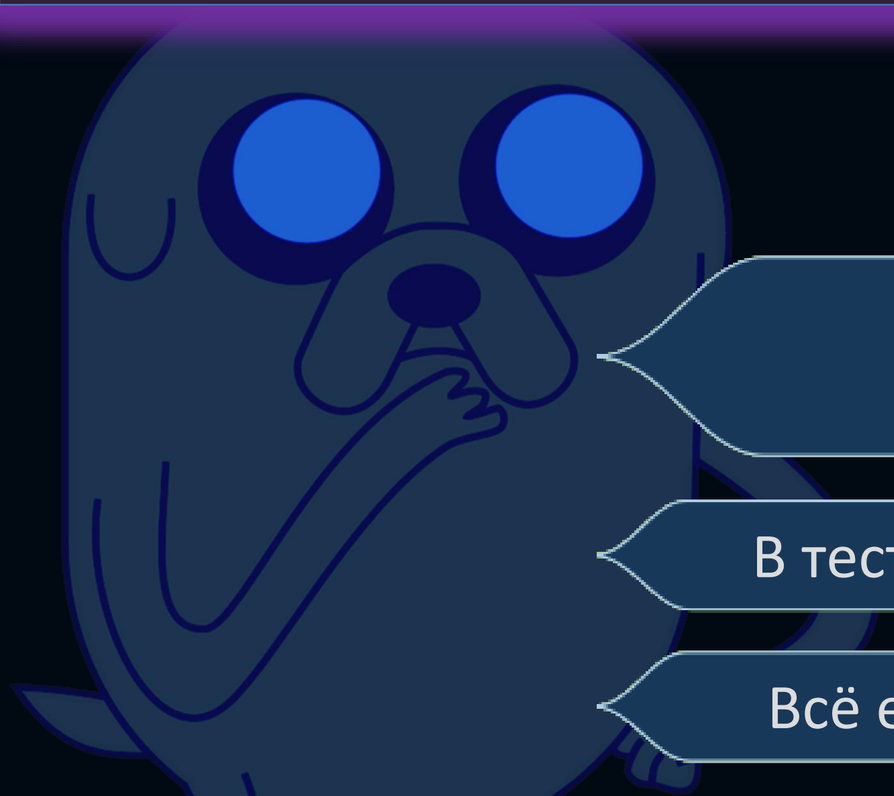


Куда нас это приведёт?

- В тестовый метод
- В конструктор атрибута
- Всё ещё никуда
- Куда-то ещё

# Дебажим!!!

```
try
{
    return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
}
catch(Exception ex)
{
    return Array.Empty<TestMethod>();
}
```



ments[0], 'System.Object',
ment](Int32)' violates the

Inner Exception
VerificationException: Method Jokes.Tests.IntegrationTests.Test: type argument 'System.Object' violates the constraint of type parameter 'TProvider'.

Куда нас это приведёт?

- В тестовый метод
- В конструктор атрибута
- Всё ещё никуда
- Куда-то ещё

# Дебажим!!!

```
try
{
    return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
}
catch(Exception ex)
{
    return Array.Empty<TestMethod>();
}
```

Exception Thrown

**System.ArgumentException:** 'GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCategory,TRequirement](Int32)' violates the constraint of type 'TProvider'.'

Inner Exception
VerificationException: Method Jokes.Tests.IntegrationTests.Test: type argument 'System.Object' violates the constraint of type parameter 'TProvider'.

Куда нас это приведёт?

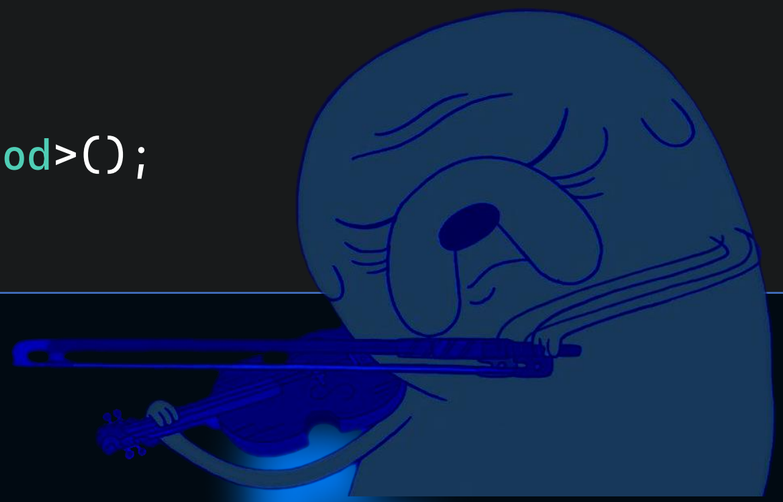
- В тестовый метод
- В конструктор атрибута
- Всё ещё никуда
- Куда-то ещё

# Занимательный факт

NUnit создаёт тесты непосредственно перед их выполнением и в том же самом процессе

# Всё-таки добавим плохой тест к остальным

```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return Array.Empty<TestMethod>();
    }
}
```



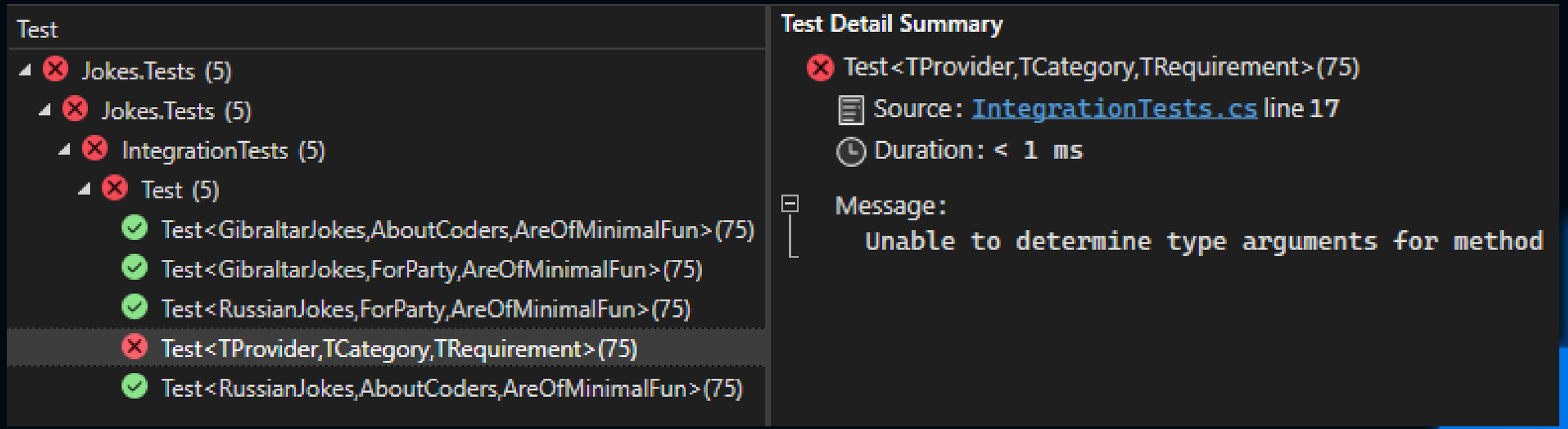
# Всё-таки добавим плохой тест к остальным

```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return Array.Empty<TestMethod>();
    }
}
```

# Всё-таки добавим плохой тест к остальным

```
public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
{
    if (!method.IsGenericMethodDefinition)
        return base.BuildFrom(method, suite);

    try
    {
        return base.BuildFrom(method.MakeGenericMethod(typeArguments), suite);
    }
    catch(Exception ex)
    {
        return base.BuildFrom(method, suite);
    }
}
```

# И получим ...


Test
Jokes.Tests (5)
Jokes.Tests (5)
IntegrationTests (5)
Test (5)
Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
Test<TProvider,TCategory,TRequirement>(75)
Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
Test Detail Summary
Test<TProvider,TCategory,TRequirement>(75)
Source: IntegrationTests.cs line 17
Duration: < 1 ms
Message:
Unable to determine type arguments for method

# И получим ...

Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequirement>(75)

Source: IntegrationTests.cs line 17

Duration: < 1 ms

Message:

```
Unable to determine type arguments for method
```

Ложь

# Инвалидируем правильно

```
/// <summary>
/// The Test abstract class represents a test within the framework.
/// </summary>
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    /// <summary>
    /// Mark the test as Invalid (not runnable) specifying a reason
    /// </summary>
    /// <param name="reason">The reason the test is not runnable</param>
    public void MakeInvalid(string reason)
    {
        Guard.ArgumentNotNullOrEmpty(reason, nameof(reason));

        RunState = RunState.NotRunnable;
        Properties.Add(PropertyNames.SkipReason, reason);
    }

    // ...
}
```

# Инвалидируем правильно

```
/// <summary>
/// The Test abstract class represents a test within the framework.
/// </summary>
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    /// <summary>
    /// Mark the test as Invalid (not runnable) specifying a reason
    /// </summary>
    /// <param name="reason">The reason the test is not runnable</param>
    public void MakeInvalid(string reason)
    {
        Guard.ArgumentNotNullOrEmpty(reason, nameof(reason));

        RunState = RunState.NotRunnable;
        Properties.Add(PropertyNames.SkipReason, reason);
    }

    // ...
}
```

# Инвалидируем правильно

```
/// <summary>
/// The Test abstract class represents a test within the framework.
/// </summary>
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    /// <summary>
    /// Mark the test as Invalid (not runnable) specifying a reason
    /// </summary>
    /// <param name="reason">The reason the test is not runnable</param>
    public void MakeInvalid(string reason)
    {
        Guard.ArgumentNotNullOrEmpty(reason, nameof(reason));

        RunState = RunState.NotRunnable;
        Properties.Add(PropertyNames.SkipReason, reason);
    }

    // ...
}
```

# Инвалидируем правильно

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.MakeInvalid(ex.Message);

            return test;
        });
}
```

# Инвалидируем правильно

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.MakeInvalid(ex.Message);

            return test;
        });
}
```

# Но что-то идёт не так ...

Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequirement>(75)

Source: IntegrationTests.cs line 17

Duration: < 1 ms

Message:

```
Unable to determine type arguments for method
```

# Но что-то идёт не так ...

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.MakeInvalid(ex.Message);

            return test;
        });
}
```

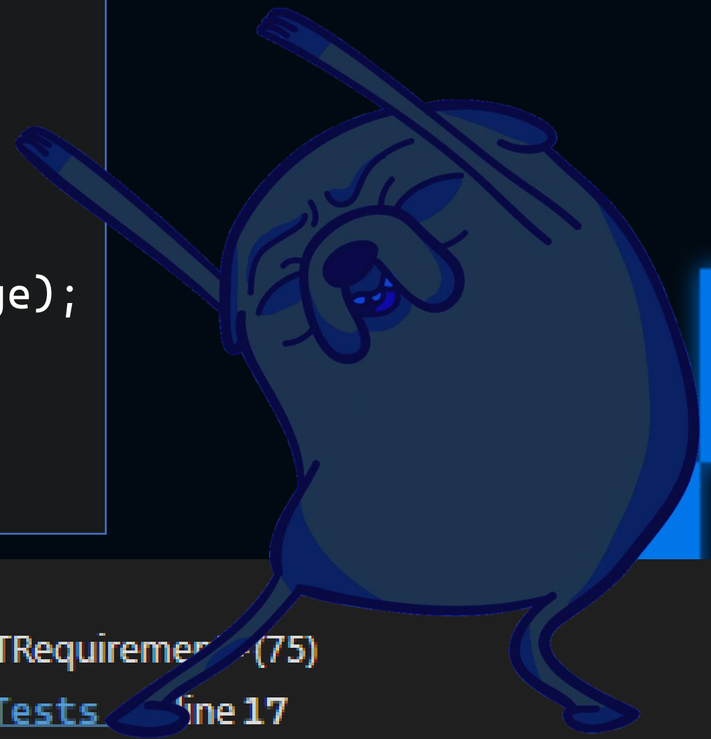
# Но что-то идёт не так ...

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.MakeInvalid(ex.Message);

            return test;
        });
}
```

| | |
|---|---|
| (new System.Collections.Generic.IDictionaryDebugView<string, System.Collections.ILi | View "_SKIPREASON" |
| (new System.Collections.Generic.IDictionaryDebugView<string, System.Collections.ILi | View Count = 2 |
| new System.Collections.Generic.ICollectionDebugView<object>((new System.Collecti | View "Unable to determine type arguments for method" |
| new System.Collections.Generic.ICollectionDebugView<object>((new System.Collecti | View "GenericArguments[0], 'System.Object', on 'Void Test |

# А мы чиним

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.RunState = RunState.NotRunnable;
            test.Properties.Set(PropertyNames.SkipReason, ex.Message);

            return test;
        });
}
```



Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequiremen>(75)

Source: IntegrationTests ine 17

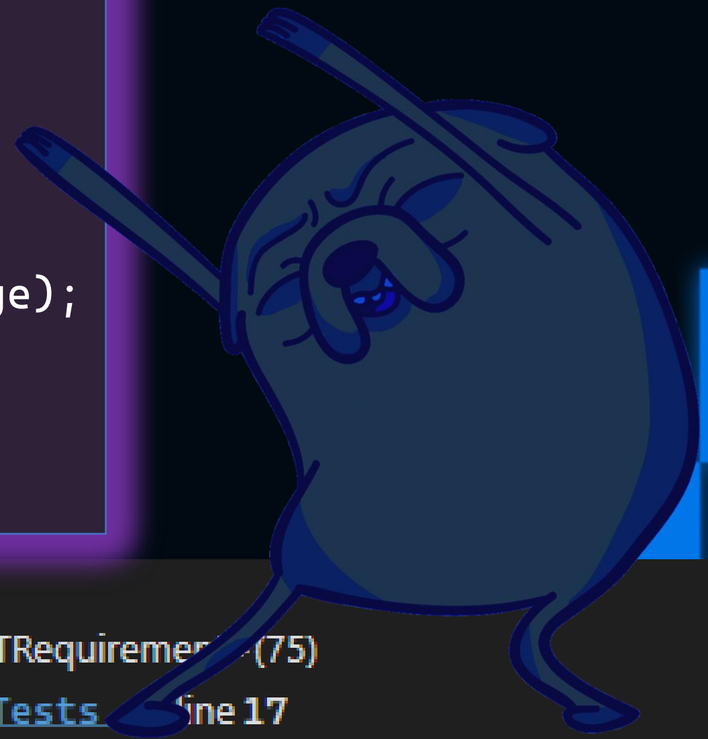
Duration: 5 ms

Message:

GenericArguments[0], 'System.Object', on 'Void Test

# А мы чиним

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.RunState = RunState.NotRunnable;
            test.Properties.Set(PropertyNames.SkipReason, ex.Message);

            return test;
        });
}
```



Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequirement>(75)

Source: IntegrationTests ine 17

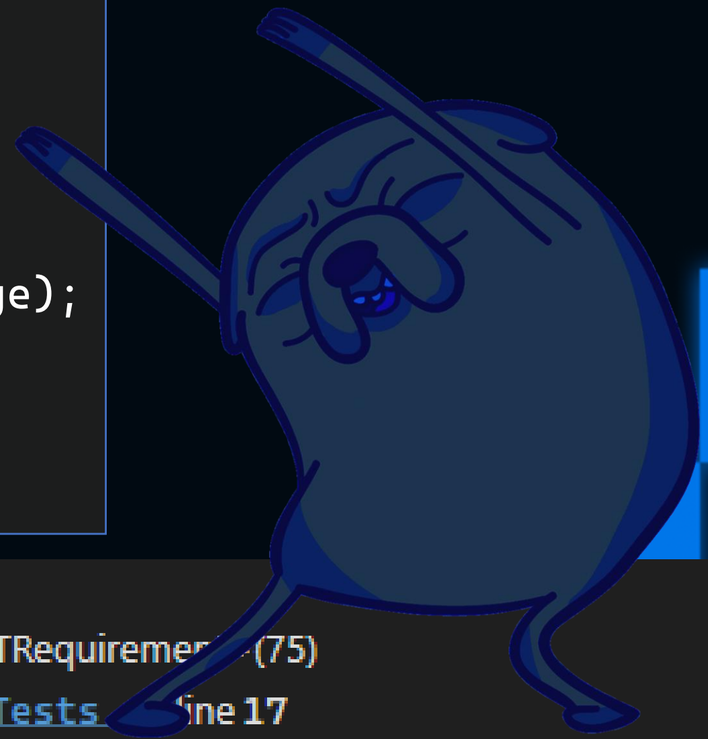
Duration: 5 ms

Message:

GenericArguments[0], 'System.Object', on 'Void Test

# А мы чиним

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.RunState = RunState.NotRunnable;
            test.Properties.Set(PropertyNames.SkipReason, ex.Message);

            return test;
        });
}
```



Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequirement>(75)

Source: IntegrationTests ine 17

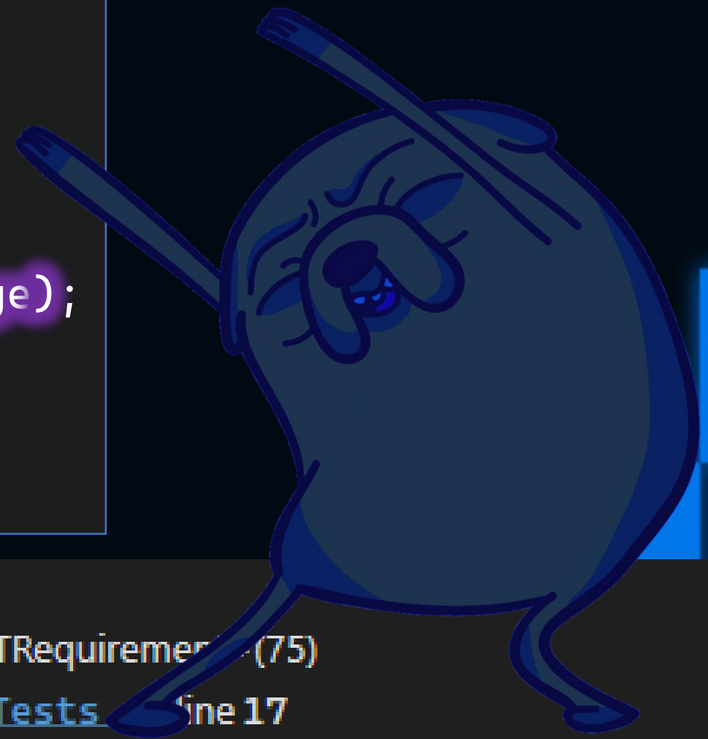
Duration: 5 ms

Message:

GenericArguments[0], 'System.Object', on 'Void Test

# А мы чиним

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.RunState = RunState.NotRunnable;
            test.Properties.Set(PropertyNames.SkipReason, ex.Message);

            return test;
        });
}
```



Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequiremen(75)

Source: IntegrationTests ine 17

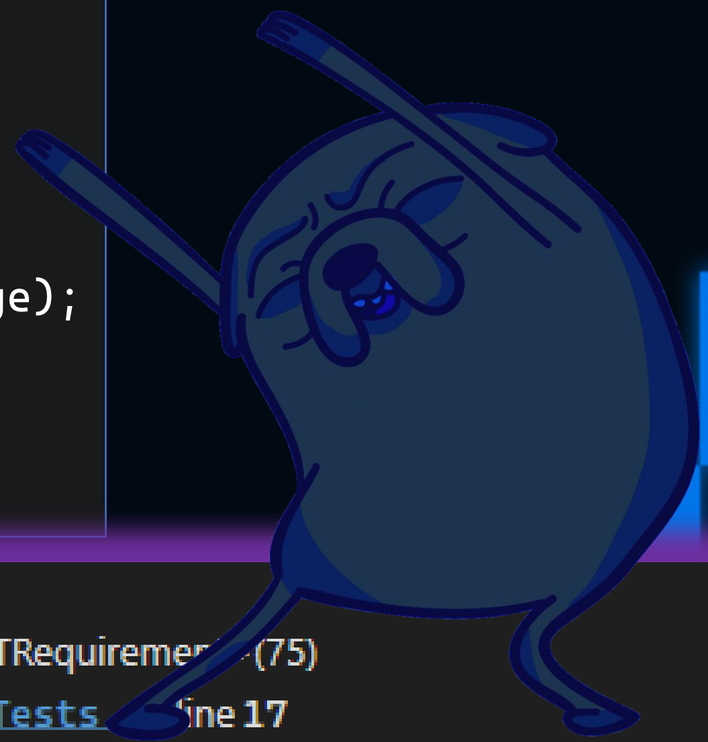
Duration: 5 ms

Message:

GenericArguments[0], 'System.Object', on 'Void Test

# А мы чиним

```
catch(Exception ex)
{
    return base.BuildFrom(method, suite)
        .Select(test =>
        {
            test.RunState = RunState.NotRunnable;
            test.Properties.Set(PropertyNames.SkipReason, ex.Message);

            return test;
        });
}
```



Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequiremen(75)

Source: IntegrationTests ine 17

Duration: 5 ms

Message:

GenericArguments[0], 'System.Object', on 'Void Test

# Check Point

- ✓ NUnit.Framework.Interfaces.ITestBuilder — удобная точка расширения NUnit
- ✓ Необработанные исключения в IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite) ломают обнаружение тестов
- ✓ NUnit.Framework.Internal.TestMethod — позволяет гибко настраивать возвращаемые тесты

# Check Point

- ✓ NUnit.Framework.Interfaces.ITestBuilder — удобная точка расширения NUnit
- ✓ Необработанные исключения в IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite) ломают обнаружение тестов
- ✓ NUnit.Framework.Internal.TestMethod — позволяет гибко настраивать возвращаемые тесты

# Check Point

- ✓ NUnit.Framework.Interfaces.ITestBuilder — удобная точка расширения NUnit
- ✓ Необработанные исключения в IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite) ломают обнаружение тестов
- ✓ NUnit.Framework.Internal.TestMethod — позволяет гибко настраивать возвращаемые тесты

# Check Point

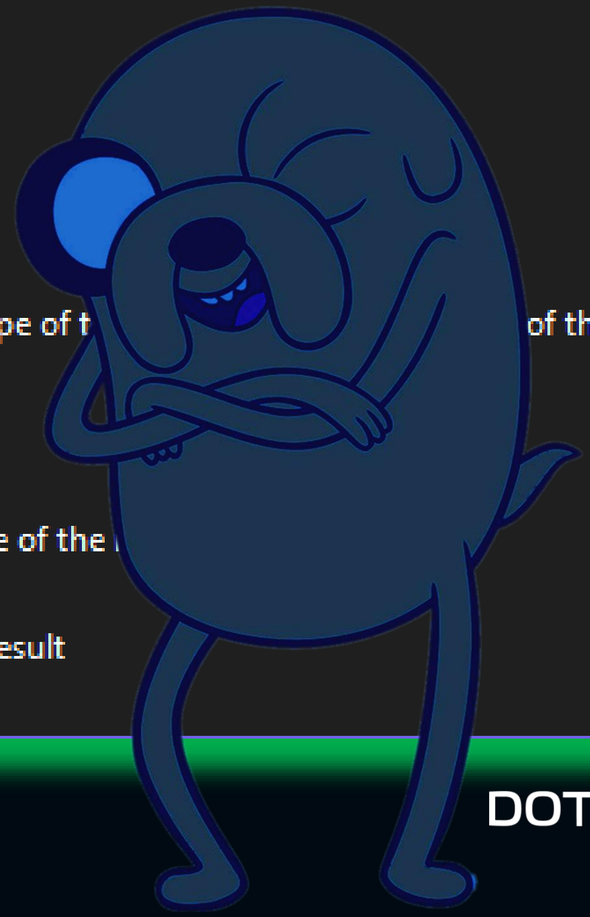
- ✓ NUnit.Framework.Interfaces.ITestBuilder — удобная точка расширения NUnit
- ✓ Необработанные исключения в IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite) ломают обнаружение тестов
- ✓ NUnit.Framework.Internal.TestMethod — позволяет гибко настраивать возвращаемые тесты

# Занимательный факт

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Jokes.Tests
  - Dependencies
    - Analyzers
      - Microsoft.CodeAnalysis.CSharp.NetAnalyzers
      - Microsoft.CodeAnalysis.NetAnalyzers
      - nunit.analyzers
        - NUnit1001: The individual arguments provided by a TestCaseAttribute must match the type of t of the method
        - NUnit1002: The TestCaseSource should use nameof operator to specify target
        - NUnit1003: The TestCaseAttribute provided too few arguments
        - NUnit1004: The TestCaseAttribute provided too many arguments
        - NUnit1005: The type of the value specified via ExpectedResult must match the return type of the
        - NUnit1006: ExpectedResult must not be specified when the method returns void
        - NUnit1007: The method has non-void return type, but no result is expected in ExpectedResult
        - NUnit1008: Specifying ParallelScope.Self on assembly level has no effect



# Занимательный факт

```
[TestCase]
public void SimpleTest(int value)
```

# Занимательный факт

```
[TestCase]
public void SimpleTest(int value)
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 4 Messages | Build + IntelliSense

Search Error List

| | Code | Description | Project | File | Line | Category | Suppression State |
|---|---|---|---|---|---|---|---|
| ▷ | NUnit1003 | The TestCaseAttribute provided too few arguments. Expected '1', but got '0'. | Jokes.Tests | IntegrationTests.cs | 12 | Structure | Active |

# Добавим второй несовместимый атрибут

```
[GCase<object, object, object>(75)]
[GCase<int, int, int>(75)]
```

Что произойдет с Test Explorer?

- Опять сломается
- Отобразит новый тест
- Ничего
- Зависнет

# Добавим второй несовместимый атрибут

Test

- Jokes.Tests (5)
  - Jokes.Tests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<TProvider,TCategory,TRequirement>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)

**Test Detail Summary**

Test<TProvider,TCategory,TRequirement>(75)

Source: IntegrationTests.cs line 18

Test has multiple result outcomes

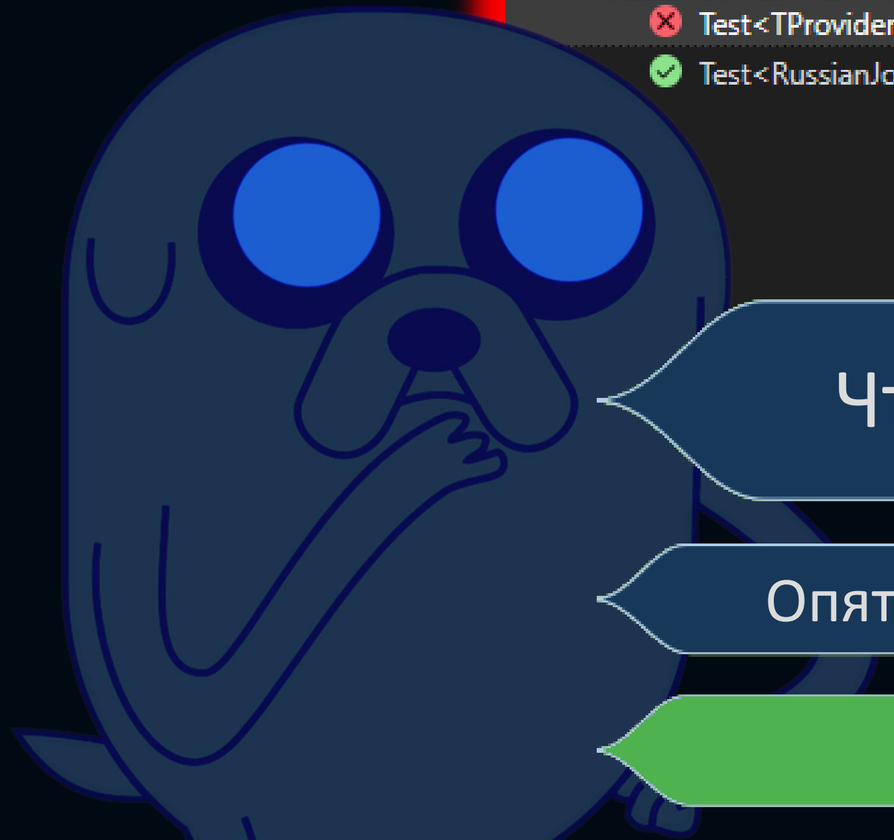
2 Failed

Results

1) Test<TProvider,TCategory,TRequirement>(75)
   Duration: < 1 ms
   Message:
   GenericArguments[0], 'System.Object', on

2) Test<TProvider,TCategory,TRequirement>(75)
   Duration: < 1 ms

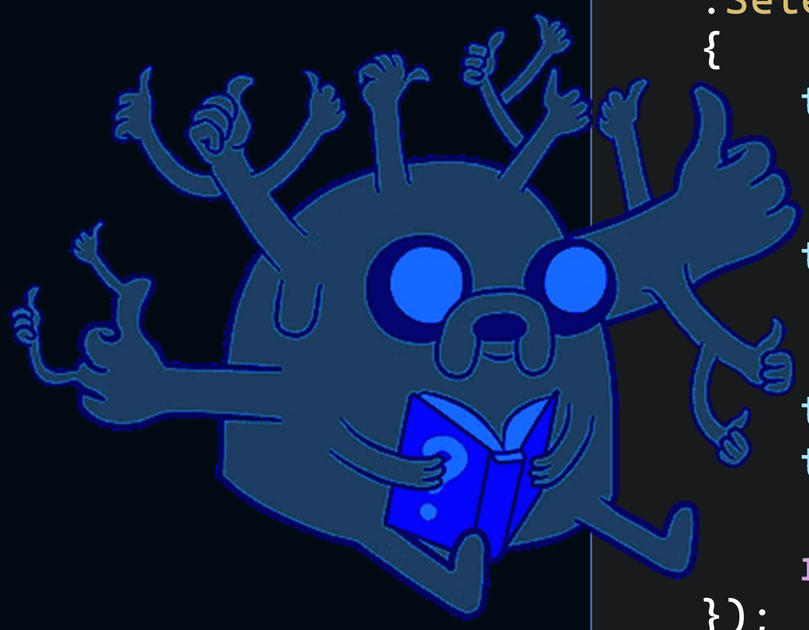
...m.Int32', on



Что произойдет с Test Explorer?

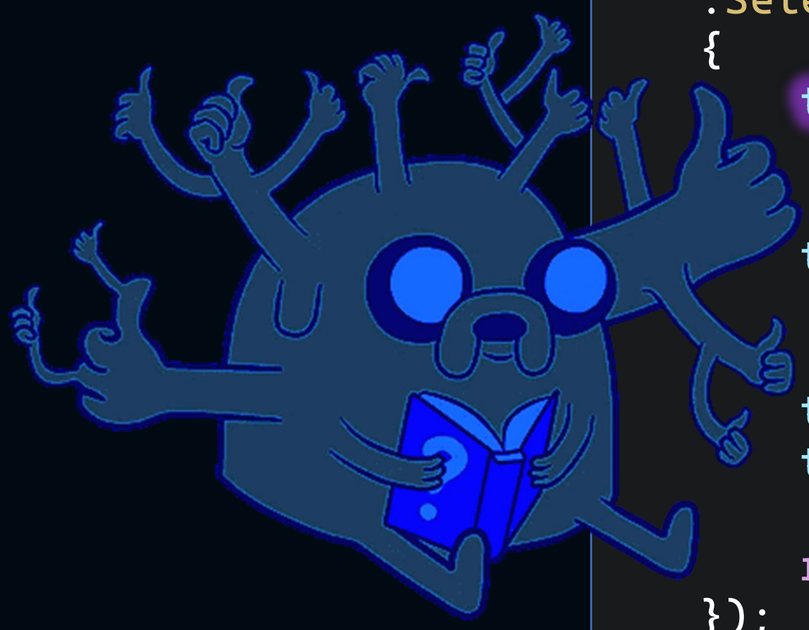
- Опять сломается
- Отобразит новый тест
- Ничего
- Зависнет

# Методом тыка находим решение проблемы, убивая сразу двух зайцев



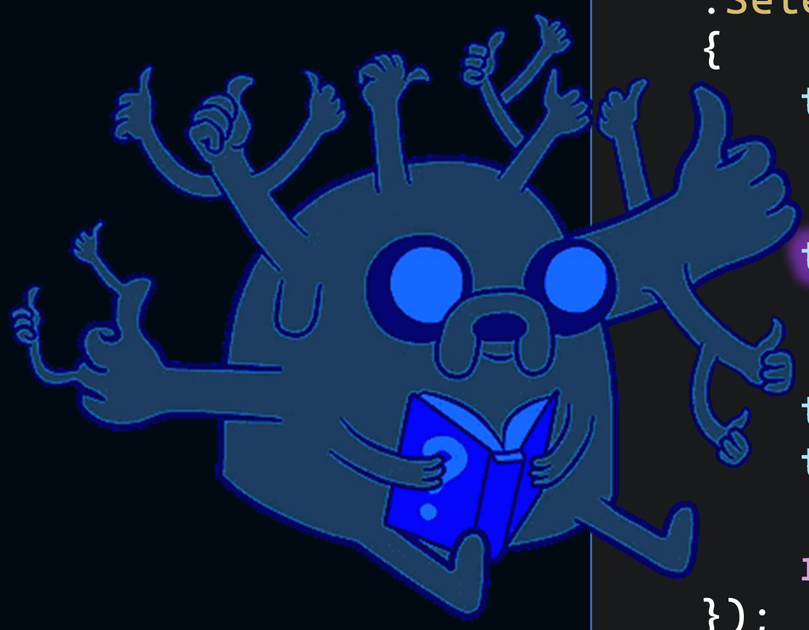
```
return base.BuildFrom(method, suite)
    .Select(test =>
    {
        test.Name = $"{method.Name}<{
            string.Join(',', typeArguments.Select(type => type.Name))}>({
            string.Join(',', Arguments)})";
        test.FullName = $"{method.MethodInfo.DeclaringType.FullName}.{
            test.Name}";

        test.RunState = RunState.NotRunnable;
        test.Properties.Set(PropertyNames.SkipReason, ex.Message);

        return test;
    });
```

# Методом тыка находим решение проблемы, убивая сразу двух зайцев



```
return base.BuildFrom(method, suite)
    .Select(test =>
    {
        test.Name = $"{method.Name}<{
            string.Join(',', typeArguments.Select(type => type.Name))}>({
            string.Join(',', Arguments)})";
        test.FullName = $"{method.MethodInfo.DeclaringType.FullName}.{
            test.Name}";

        test.RunState = RunState.NotRunnable;
        test.Properties.Set(PropertyNames.SkipReason, ex.Message);

        return test;
    });
```

# Методом тыка находим решение проблемы, убивая сразу двух зайцев



```
return base.BuildFrom(method, suite)
    .Select(test =>
    {
        test.Name = $"{method.Name}<{
            string.Join(',', typeArguments.Select(type => type.Name))}>({
            string.Join(',', Arguments)})";
        test.FullName = $"{method.MethodInfo.DeclaringType.FullName}.{
            test.Name}";

        test.RunState = RunState.NotRunnable;
        test.Properties.Set(PropertyNames.SkipReason, ex.Message);

        return test;
    });
```

# И всё работает (даже переход к методу)

Test

- Jokes.Tests (6)
  - Jokes.Tests (6)
    - IntegrationTests (6)
      - Test (6)
        - Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
        - Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - Test<Int32,Int32,Int32>(75)
        - Test<Object,Object,Object>(75)
        - Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
        - Test<RussianJokes,AboutCoders,AreOfMinimalFun>(75)

**Group Summary**

Jokes.Tests

Tests in group: 6

Total Duration: 4 ms

Outcomes

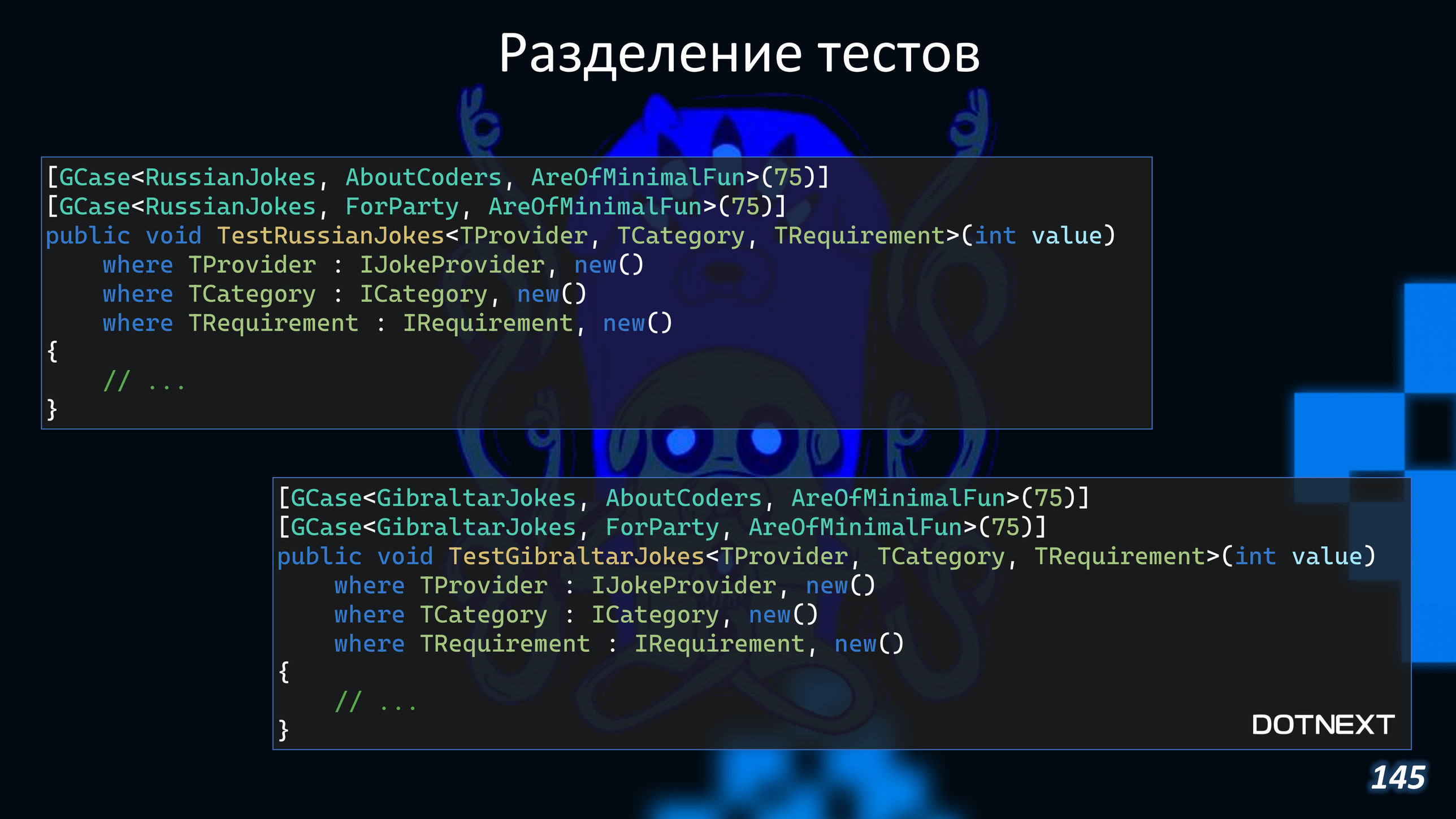
4 Passed

2 Failed

# Итак, всё работает

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Разделение тестов



```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

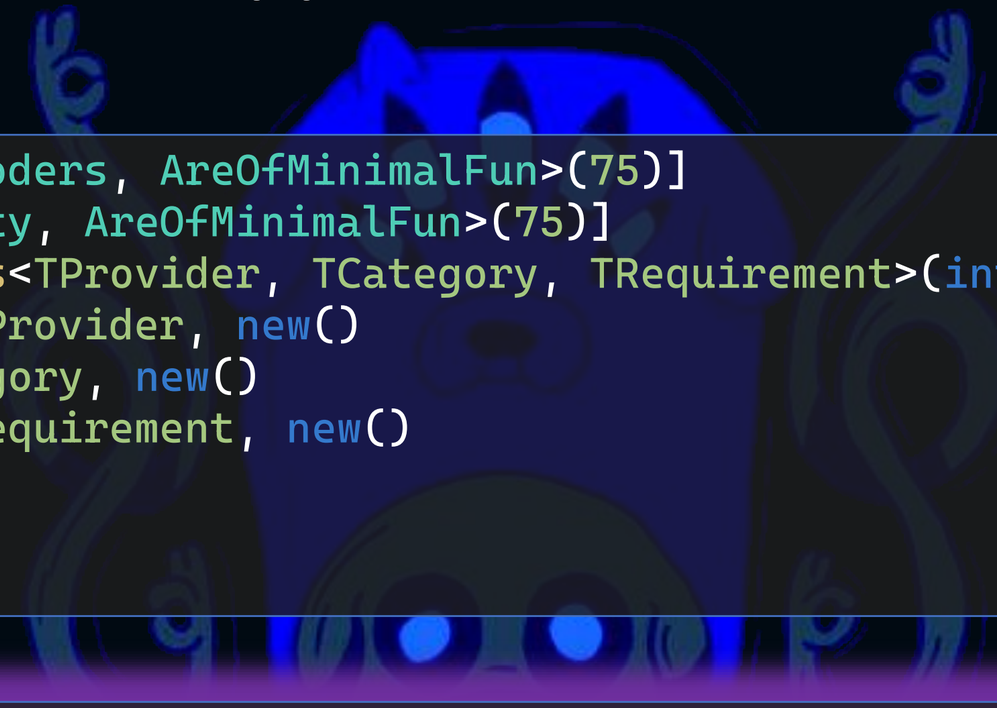
```
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

# Разделение тестов

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

```
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

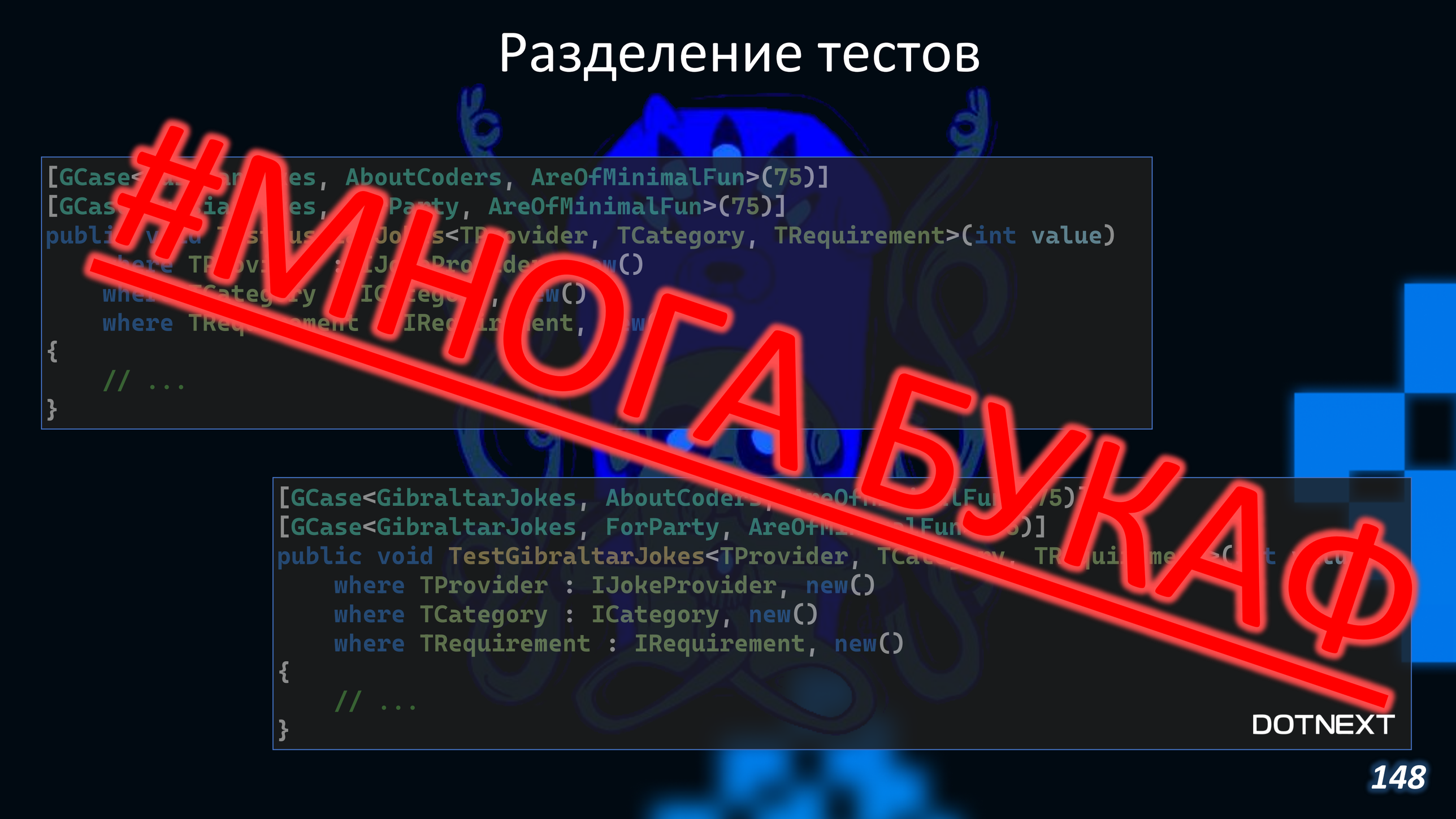
# Разделение тестов



```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

```
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

# Разделение тестов



```
[GCase<...es, AboutCoders, AreOfMinimalFun>(75)]
[GCas...ia...es, ...Party, AreOfMinimalFun>(75)]
publi... v...d Tes...us...Jo...s<TP...ovider, TCategory, TRequirement>(int value)
    ...here TP...ov... : IJo...Pro...der, ...w()
    whe...Categ...y : IC...tego..., ...w()
    where TReq...ement : IRe...ir...ent, ...w
{
    // ...
}
```

```
[GCase<GibraltarJokes, AboutCoders, ...eOf...alFu...75)]
[GCase<GibraltarJokes, ForParty, AreOf...alFun...)]
public void TestGibraltarJokes<TProvider, TCa...y, TR...qui...me...>(...t ...u
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

#МНОГА БУКАФ

DOTNEXT

# Переиспользование дублирующегося кода

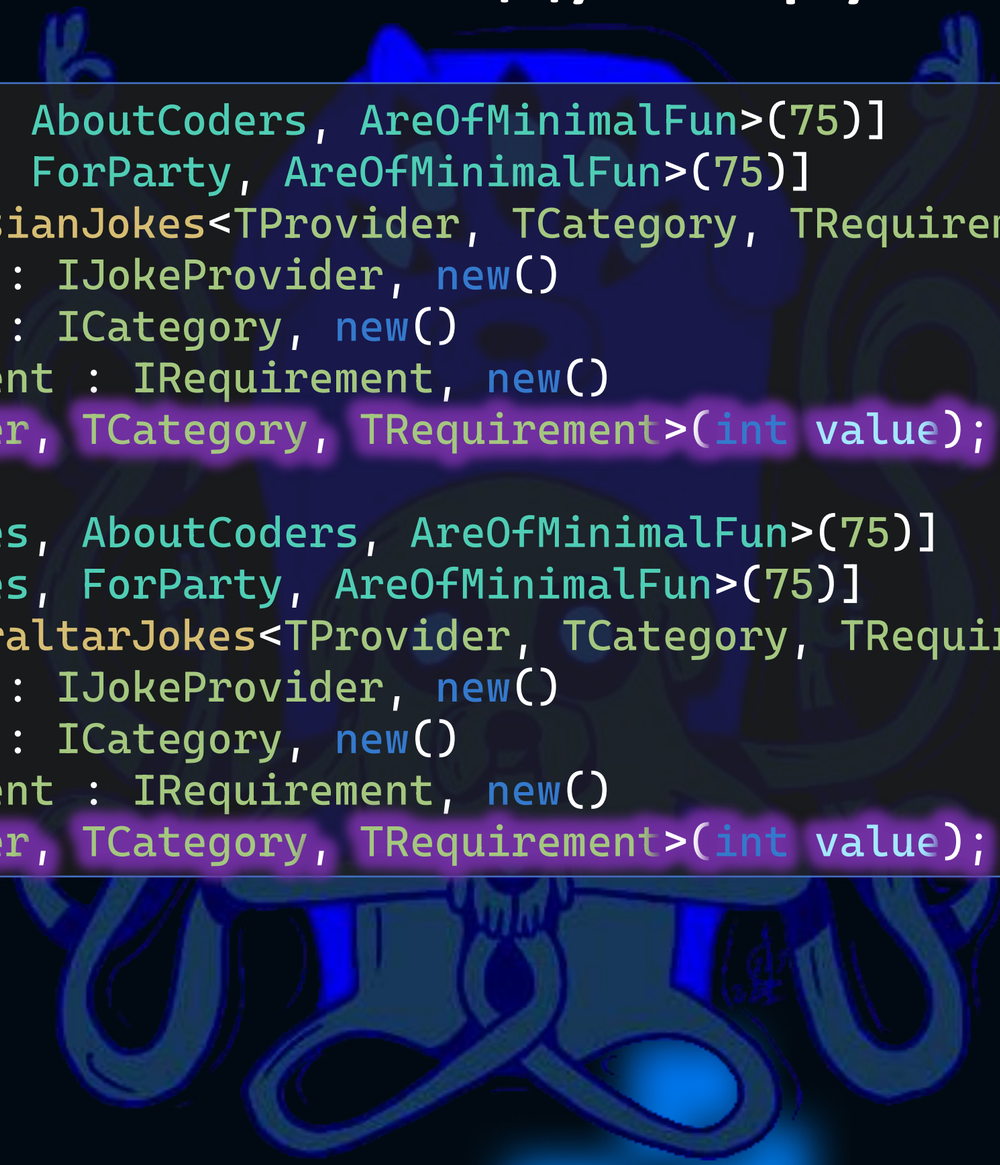
```
private void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Переиспользование дублирующегося кода

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
    => Test<TProvider, TCategory, TRequirement>(int value)

[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
    => Test<TProvider, TCategory, TRequirement>(int value)
```

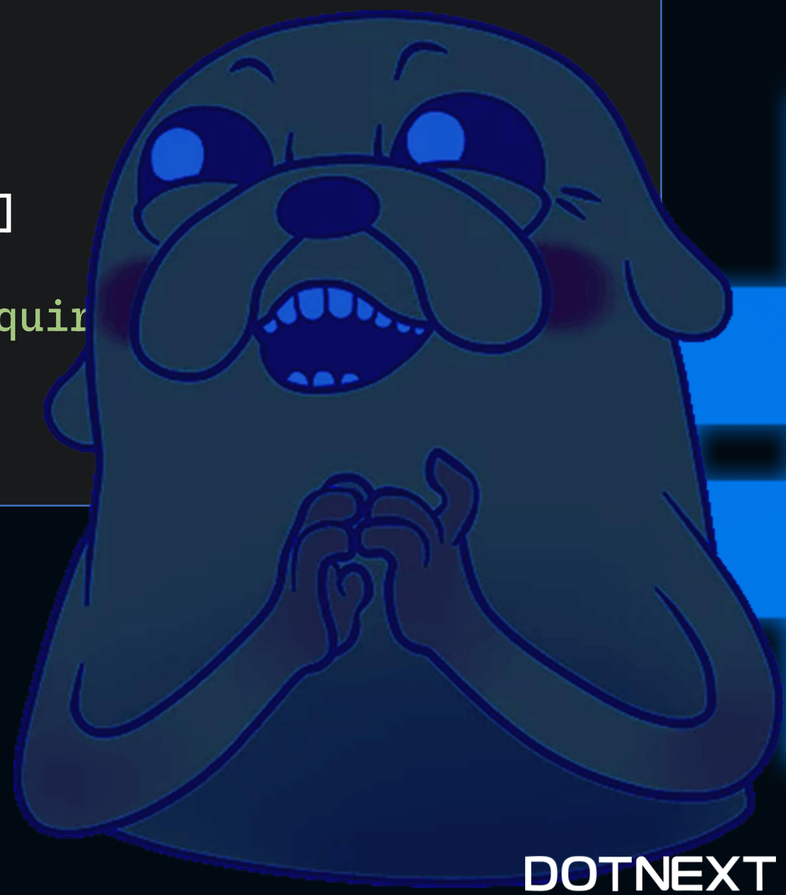
# Переиспользование дублирующегося кода

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
    => Test<TProvider, TCategory, TRequirement>(int value);

[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
    => Test<TProvider, TCategory, TRequirement>(int value);
```



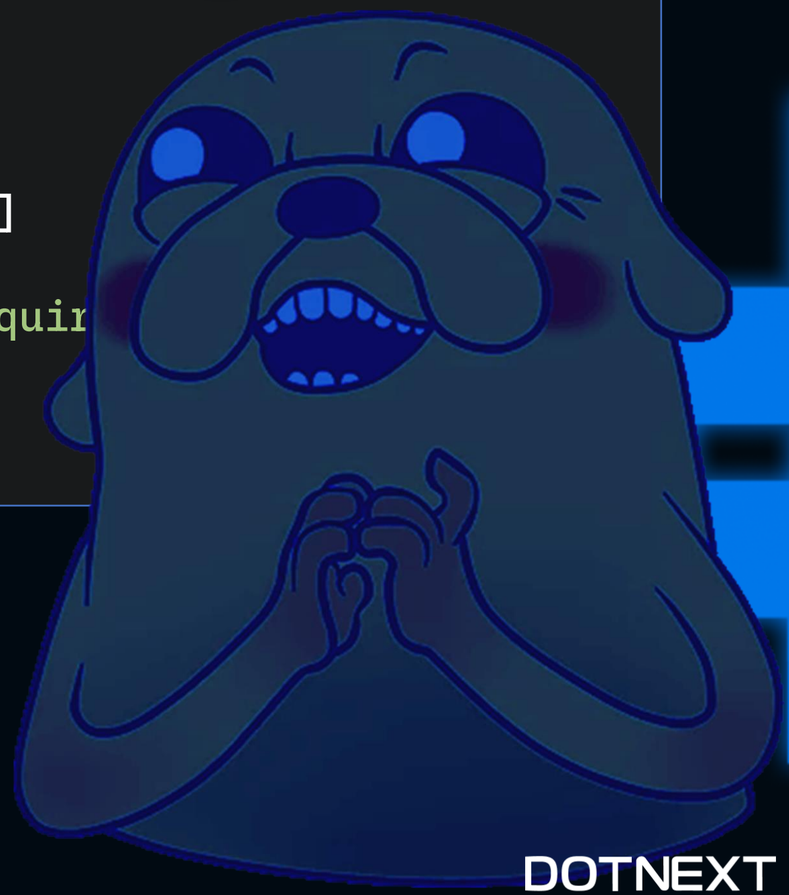
# Безумная идея

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>()
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new ()
    where TRequirement : IRequirement, new () { }

[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequir
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new ()
    where TRequirement : IRequirement, new () { }
```



# Безумная идея

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes<TProvider, TCategory, TRequirement>()
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new ()
    where TRequirement : IRequirement, new () { }

[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes<TProvider, TCategory, TRequir
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new ()
    where TRequirement : IRequirement, new () { }
```



# Аргументы против

Стандартные практики

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)] очень удобно читается

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)] очень удобно читается

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)] очень удобно читается

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)] очень удобно читается

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)] очень удобно читается

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)] очень удобно читается

# Аргументы за

- ✓ Это просто занимательная идея, а не требование
- ✓ Код теста лежит в неочевидном месте, но всё ещё очевиден
- ✓ Между добавлением атрибута и отображением результатов происходит куча неочевидных вещей
- ✓ Выполнение теста и так делегируется
- ✓ Убирается избыточный вспомогательный код
- ✓ [ Check < RussianJokes , AboutCoders , AreOfMinimalFun > (75)] очень удобно читается

# Приступим

```
[Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[Check<RussianJokes, ForParty, AreOfMinimalFun>(75)]
public void TestRussianJokes() { }

[Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes() { }
```

# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckAttribute(int value) : base(value) => this.value = value;

    // ...
}
```

# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckAttribute(int value) : base(value) => this.value = value;

    // ...
}
```

# Решение

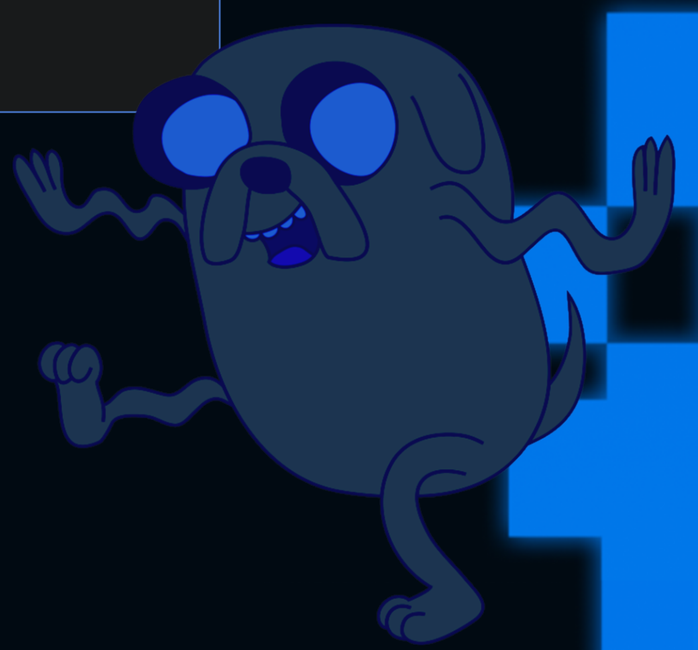
```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckAttribute(int value) : base(value) => this.value = value;

    // ...
}
```

# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckAttribute(int value) : base(value) => this.value = value;

    // ...
}
```

# Решение

```
public interface ICheckable
{
    void Check<TProvider, TCategory, TRequirement>(int value)
        where TProvider : IJokeProvider, new()
        where TCategory : ICategory, new()
        where TRequirement : IRequirement, new();
}
```

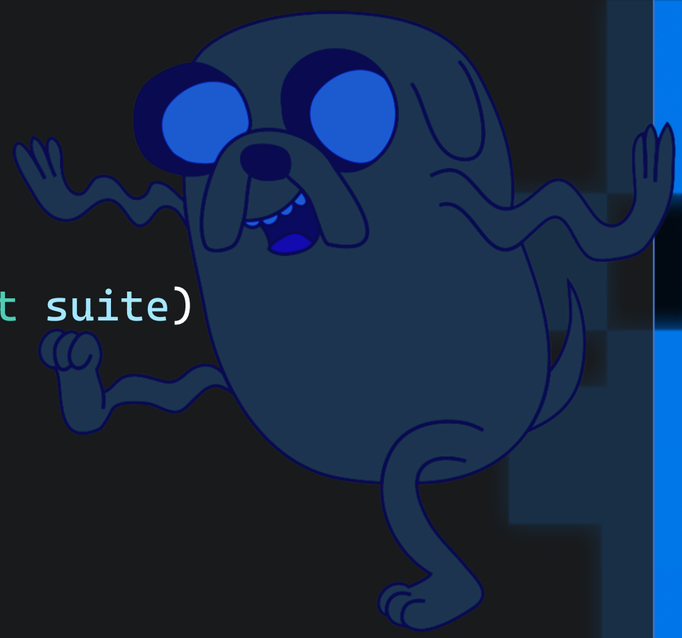


# Решение

```
public class IntegrationTests : ICheckable
{
    // ...

    public void Check<TProvider, TCategory, TRequirement>(int value)
        where TProvider : IJokeProvider, new()
        where TCategory : ICategory, new()
        where TRequirement : IRequirement, new()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```

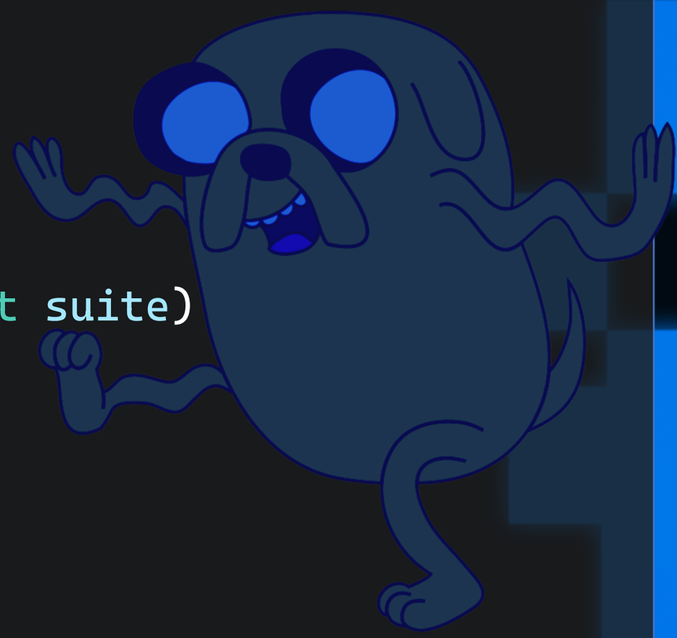
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        var type = suite.TypeInfo.Type;

        return base.BuildFrom(
            new MethodWrapper(type, nameof(ICheckable.Check)),
            suite);
    }
}
```



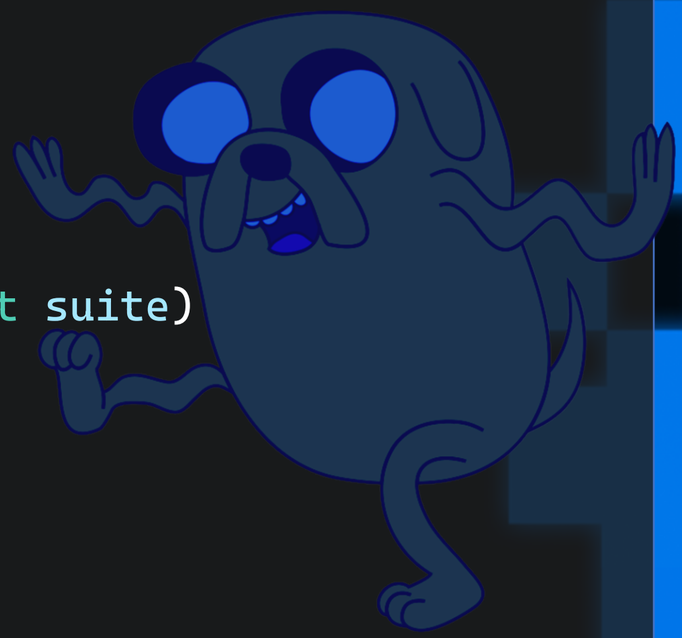
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        var type = suite.TypeInfo.Type;

        return base.BuildFrom(
            new MethodWrapper(type, nameof(ICheckable.Check)),
            suite);
    }
}
```



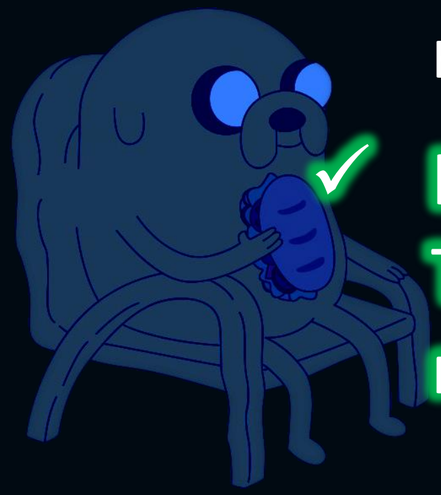
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> : GCaseAttribute, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
    {
        var type = suite.TypeInfo.Type;

        return base.BuildFrom(
            new MethodWrapper(type, nameof(ICheckable.Check)),
            suite);
    }
}
```

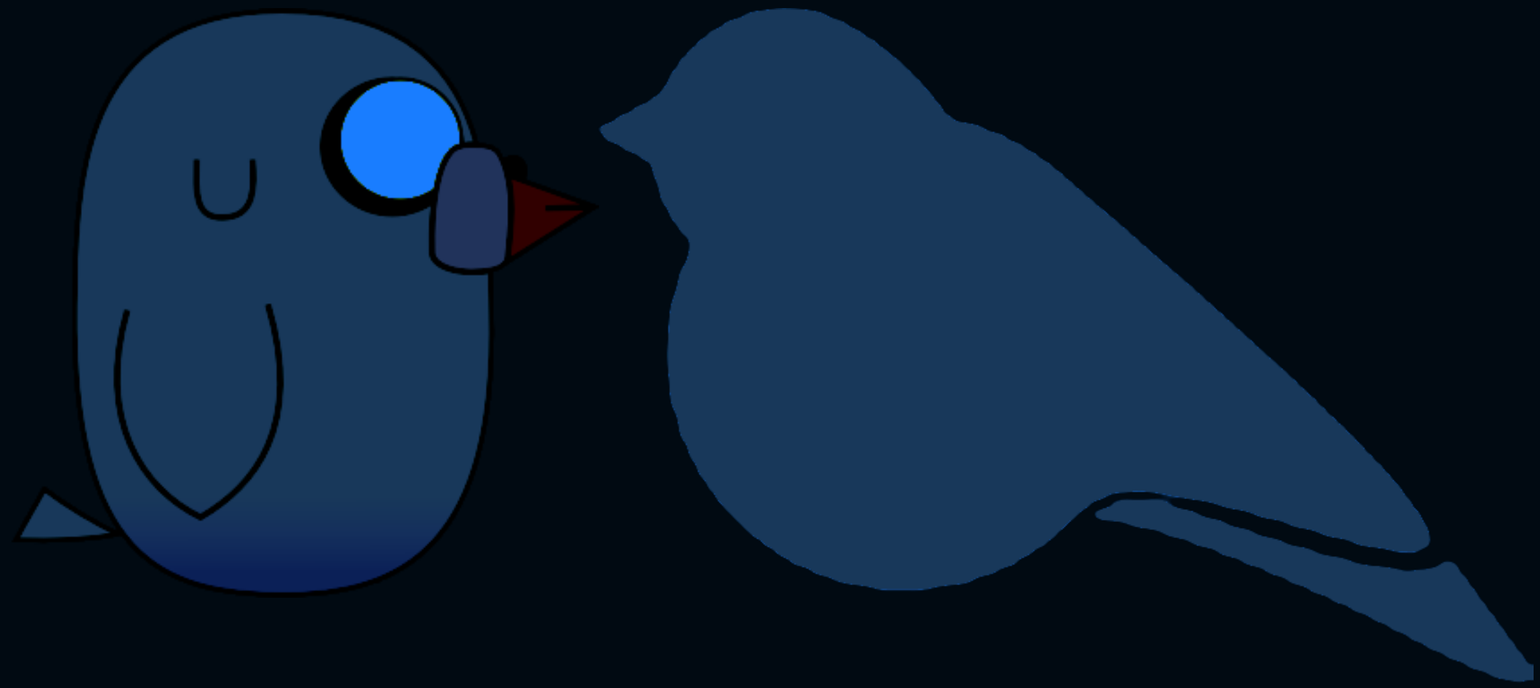


# Check Point

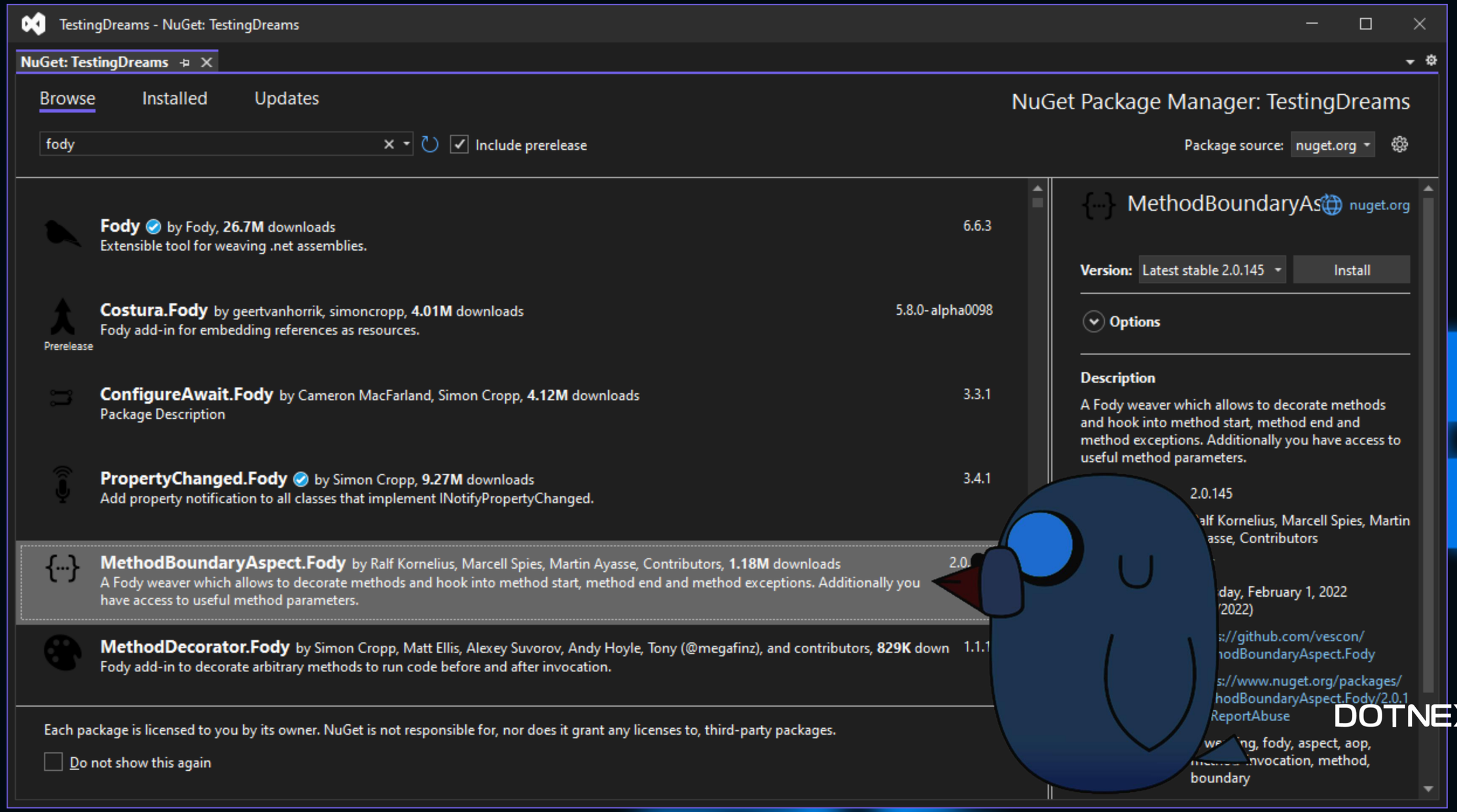
- ✓ NUnit.Framework.Interfaces.ITestBuilder — удобная точка расширения NUnit
- ✓ Необработанные исключения в IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite) ломают обнаружение тестов
- ✓ NUnit.Framework.Internal.TestMethod — позволяет гибко настраивать возвращаемые тесты
- ✓ IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite) позволяет с некоторыми ограничениями даже подменить тестовый метод



# Fody



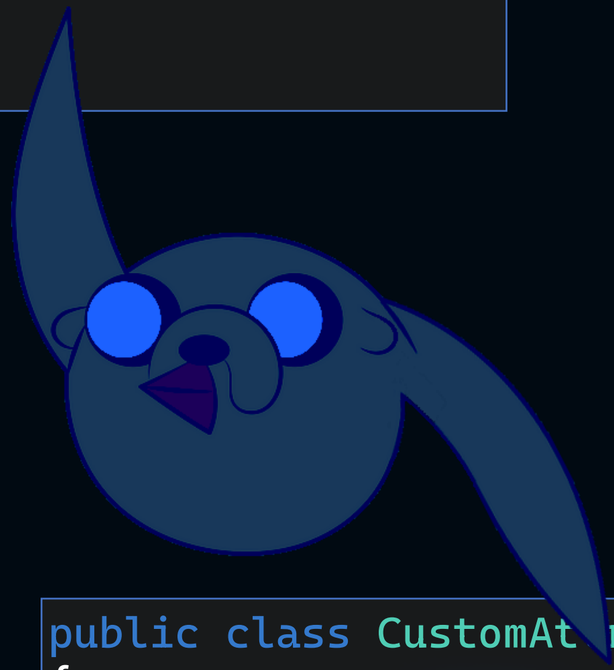
# Выбор


TestingDreams - NuGet: TestingDreams
NuGet: TestingDreams
Browse
Installed
Updates
NuGet Package Manager: TestingDreams
fody
Include prerelease
Package source: nuget.org
Fody by Fody, 26.7M downloads
Extensible tool for weaving .net assemblies.
6.6.3
Costura.Fody by geertvanhorrik, simoncropp, 4.01M downloads
Fody add-in for embedding references as resources.
5.8.0-alpha0098
Prerelease
ConfigureAwait.Fody by Cameron MacFarland, Simon Cropp, 4.12M downloads
Package Description
3.3.1
PropertyChanged.Fody by Simon Cropp, 9.27M downloads
Add property notification to all classes that implement INotifyPropertyChanged.
3.4.1
MethodBoundaryAspect.Fody by Ralf Kornelius, Marcell Spies, Martin Ayasse, Contributors, 1.18M downloads
A Fody weaver which allows to decorate methods and hook into method start, method end and method exceptions. Additionally you have access to useful method parameters.
MethodDecorator.Fody by Simon Cropp, Matt Ellis, Alexey Suvorov, Andy Hoyle, Tony (@megafinz), and contributors, 829K down
Fody add-in to decorate arbitrary methods to run code before and after invocation.
1.1.1
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.
Do not show this again
MethodBoundaryAs
nuget.org
Version: Latest stable 2.0.145
Install
Options
Description
A Fody weaver which allows to decorate methods and hook into method start, method end and method exceptions. Additionally you have access to useful method parameters.
2.0.145
day, February 1, 2022
'2022)
://github.com/vescon/
odBoundaryAspect.Fody
s://www.nuget.org/packages/
hodBoundaryAspect.Fody/2.0.1
ReportAbuse
ng, fody, aspect, aop,
nvocation, method,
boundary

# Выбор

## MethodBoundaryAspect.Fody

```
public class CustomAttribute : OnMethodBoundaryAspect
{
    // ...
}
```



## MethodDecorator.Fody

```
public class CustomAttribute : Attribute, IMethodDecorator
{
    // ...
}
```

# Выбор

## MethodBoundaryAspect.Fody

```
public class CustomAttribute : OnMethodBoundaryAspect
{
    // ...
}
```

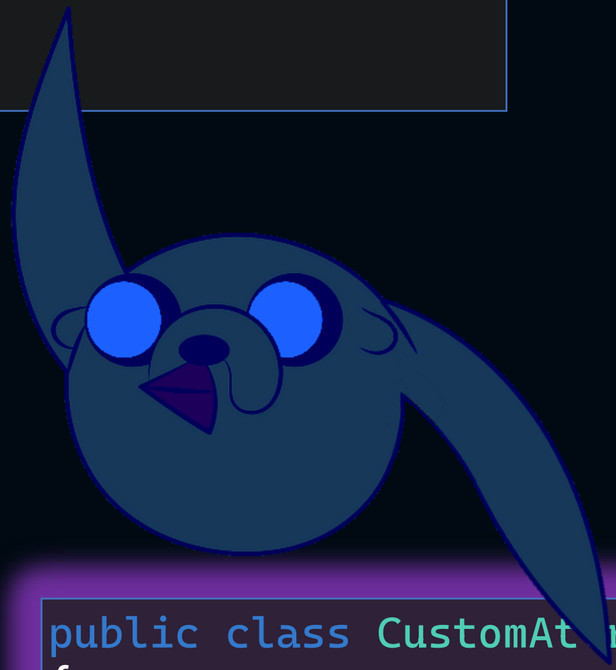
## MethodDecorator.Fody

```
public class CustomAttribute : Attribute, IMethodDecorator
{
    // ...
}
```

# Выбор

## MethodBoundaryAspect.Fody

```
public class CustomAttribute : OnMethodBoundaryAspect
{
    // ...
}
```



## MethodDecorator.Fody

```
public class CustomAttribute : Attribute, IMethodDecorator
{
    // ...
}
```

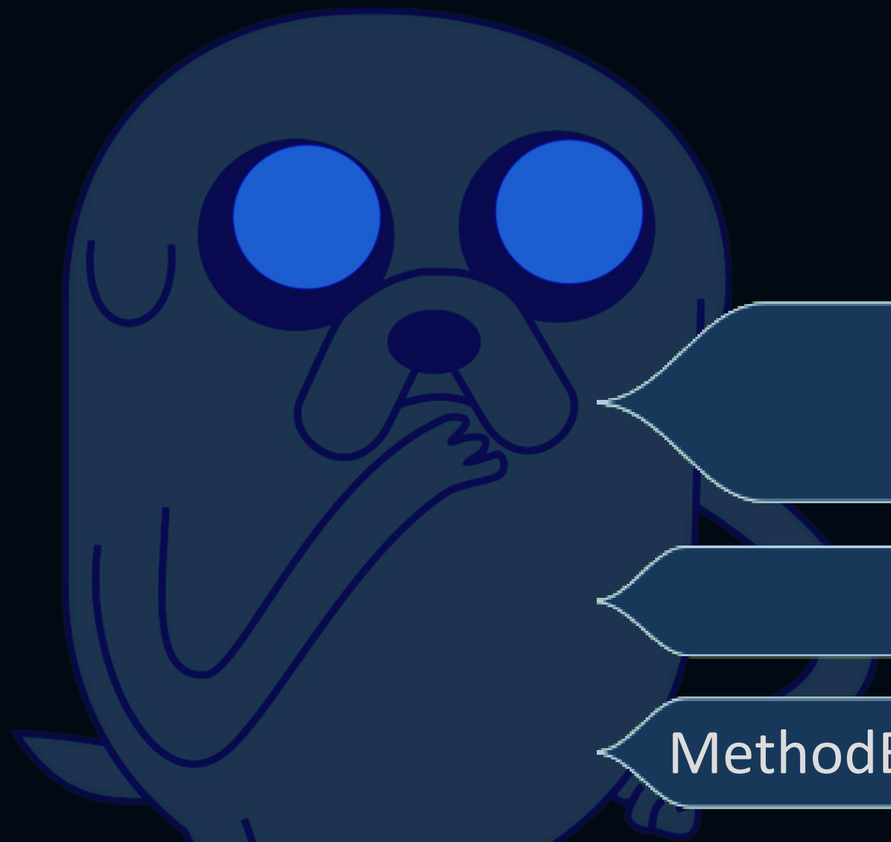
# Выбор

## MethodBoundaryAspect.Fody

```
public class CustomAttribute : OnMethodBoundaryAspect
{
    // ...
}
```

## MethodDecorator.Fody

```
public class CustomAttribute : Attribute, IMethodDecorator
{
    // ...
}
```



Какой можно использовать в CheckAttribute?

- Оба
- Ни один
- MethodBoundaryAspect
- MethodDecorator

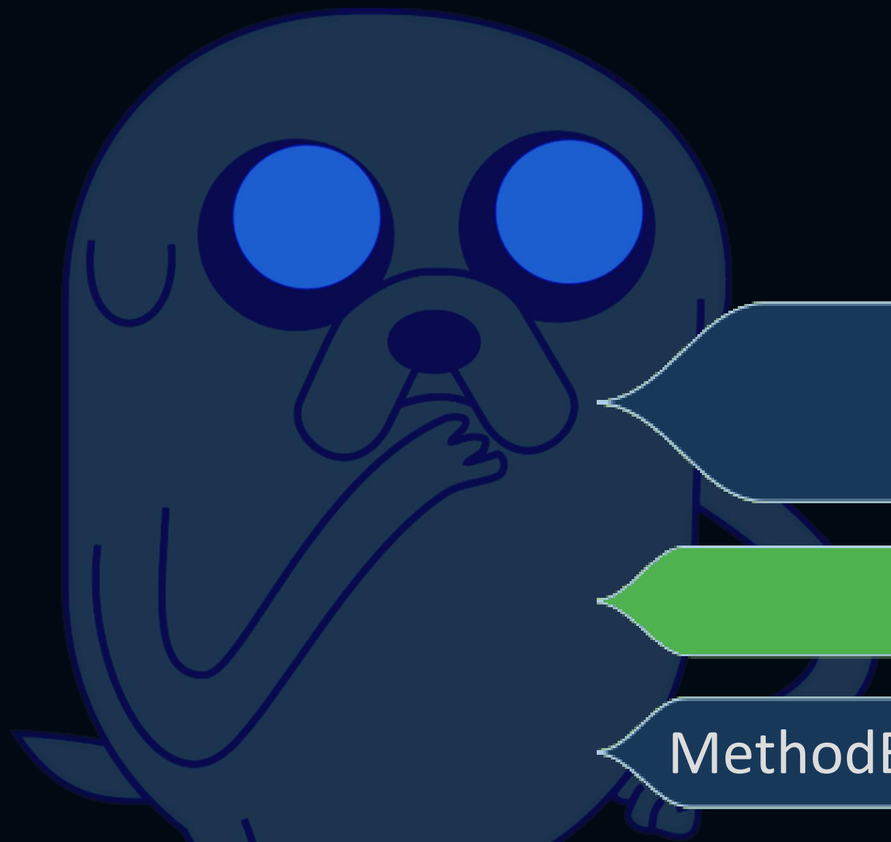
# Выбор

## MethodBoundaryAspect.Fody

```
public class CustomAttribute : OnMethodBoundaryAspect
{
    // ...
}
```

## MethodDecorator.Fody

```
public class CustomAttribute : Attribute, IMethodDecorator
{
    // ...
}
```



Какой можно использовать в CheckAttribute?

- Оба
- Ни один
- MethodBoundaryAspect
- MethodDecorator

# Занимательный факт

```
/// <summary>
/// Marks the method as callable from the NUnit test runner.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = true)]
public class TestAttribute : NUnitAttribute, ISimpleTestBuilder, IApplyToTest, IImplyFixture
{
}
```

```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
{
}
```

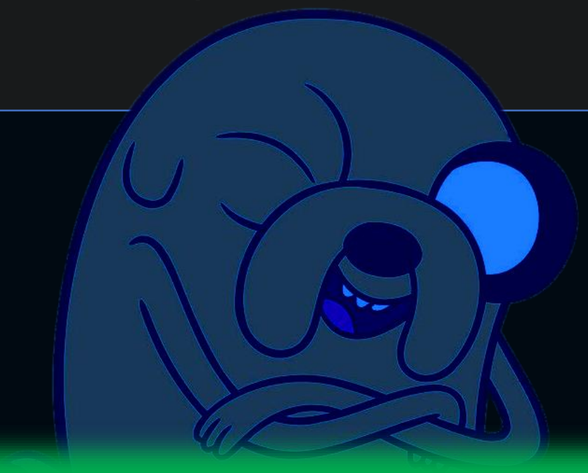
# Занимательный факт

```
/// <summary>
/// Marks the method as callable from the NUnit test runner.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = true)]
public class TestAttribute : NUnitAttribute, ISimpleTestBuilder, IApplyToTest, IImplyFixture
{
}
```

```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
{
}
```

# Занимательный факт

```
/// <summary>
/// Marks the method as callable from the NUnit test runner.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = true)]
public class TestAttribute : NUnitAttribute, ISimpleTestBuilder, IApplyToTest, IImplyFixture
{
}
```



```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
{
}
```

# Занимательный факт

```
/// <summary>
/// Marks the method as callable from the NUnit test runner.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = true)]
public class TestAttribute : NUnitAttribute, ISimpleTestBuilder, IApplyToTest, IImplyFixture
{
}
```

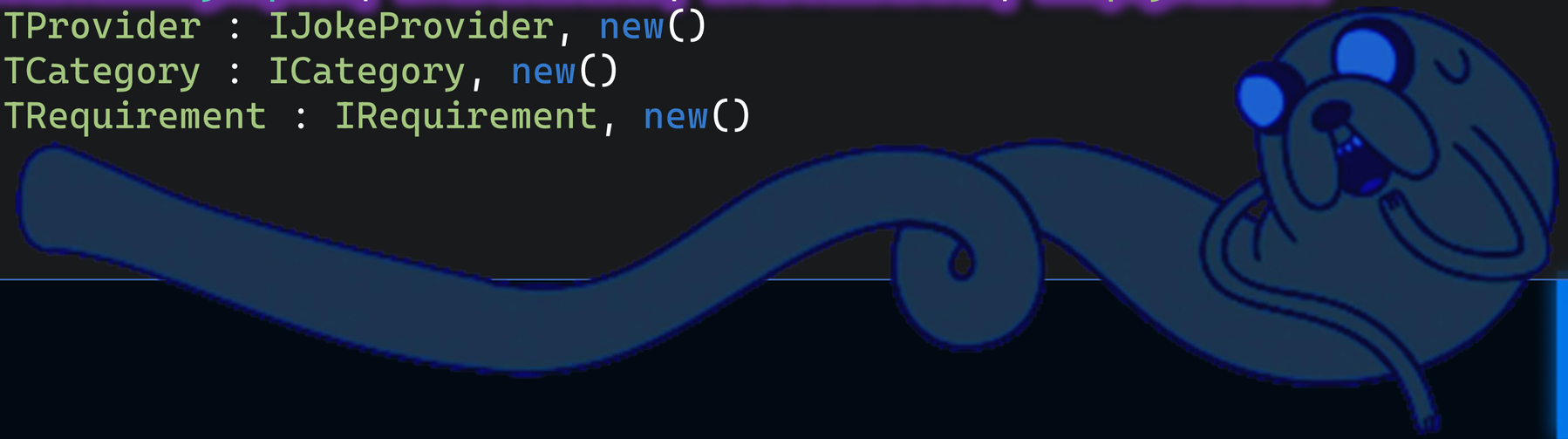
```
/// <summary>
/// Marks a method as a parameterized test suite and provides arguments for each test case.
/// </summary>
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = false)]
public class TestCaseAttribute : NUnitAttribute, ITestBuilder, ITestCaseData, IImplyFixture
{
}
```

# Занимательный факт

```
/// <summary>
/// IImplyFixture is an empty marker interface used by attributes like
/// TestAttribute that cause the class where they are used to be treated
/// as a TestFixture even without a TestFixtureAttribute.
/// </summary>
public interface IImplyFixture
{
}
```

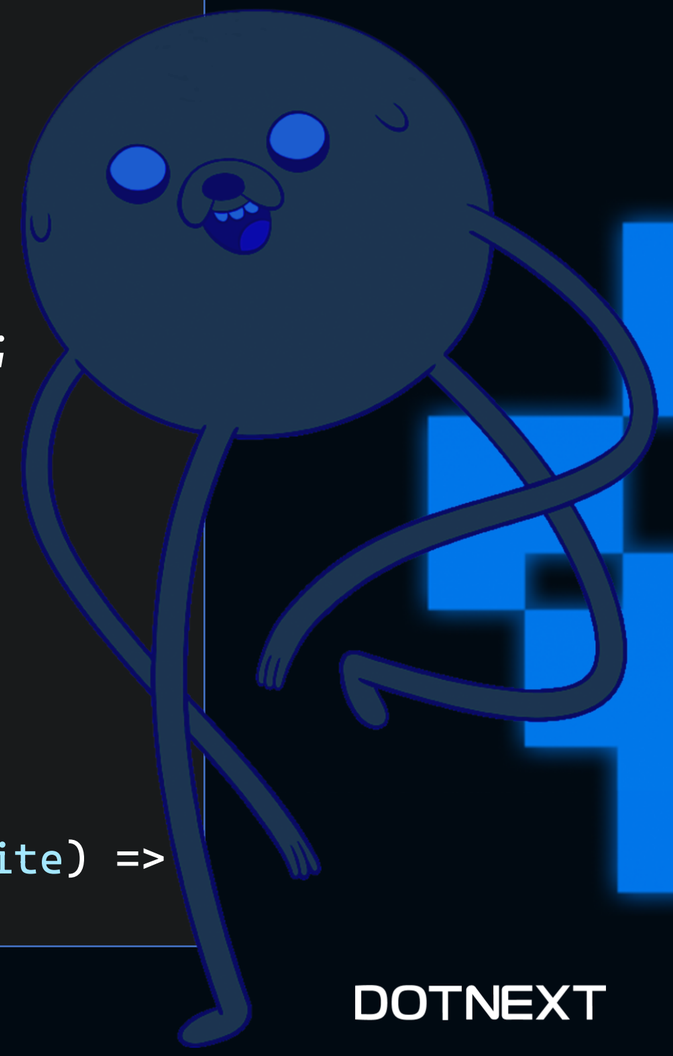
# Пойдём длинным путём

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = false)]
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    OnMethodBoundaryAspect, ITestBuilder, ITestCaseData, IImplyFixture
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```



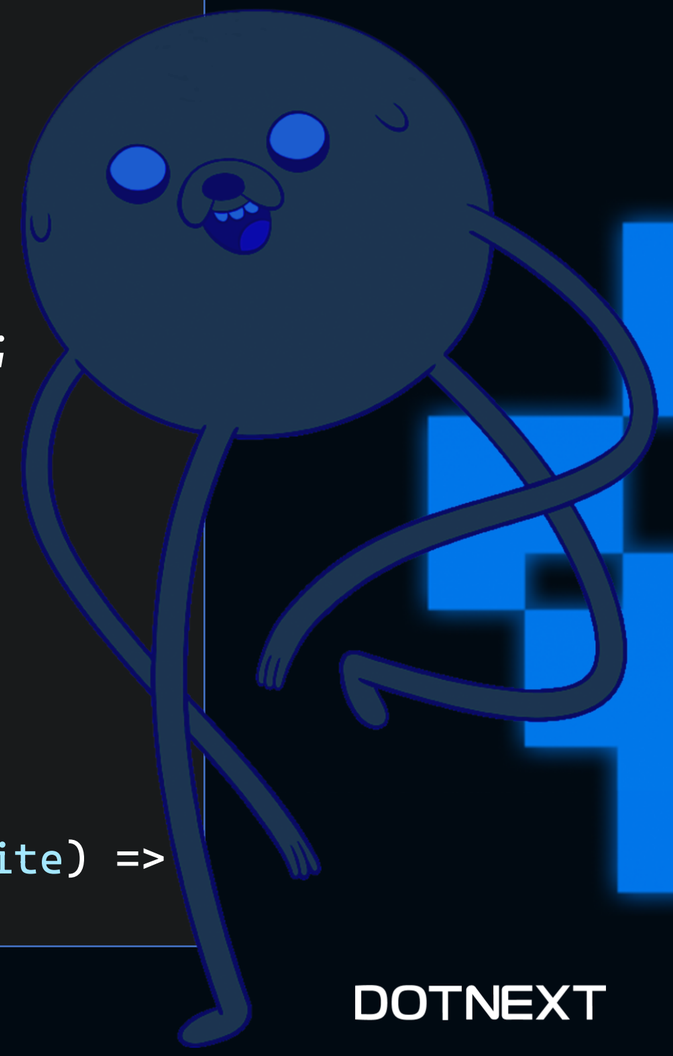
# Пойдём длинным путём

```
private readonly int value;
private readonly TestCaseAttribute testCaseAttribute = new();

public CheckAttribute(int value) => this.value = value;

public object ExpectedResult => testCaseAttribute.ExpectedResult;

public bool HasExpectedResult => testCaseAttribute.HasExpectedResult;

public string TestName => testCaseAttribute.TestName;

public RunState RunState => testCaseAttribute.RunState;

public object[] Arguments => testCaseAttribute.Arguments;

public IPropertyBag Properties => testCaseAttribute.Properties;

public IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite) =>
    testCaseAttribute.BuildFrom(method, suite);
```



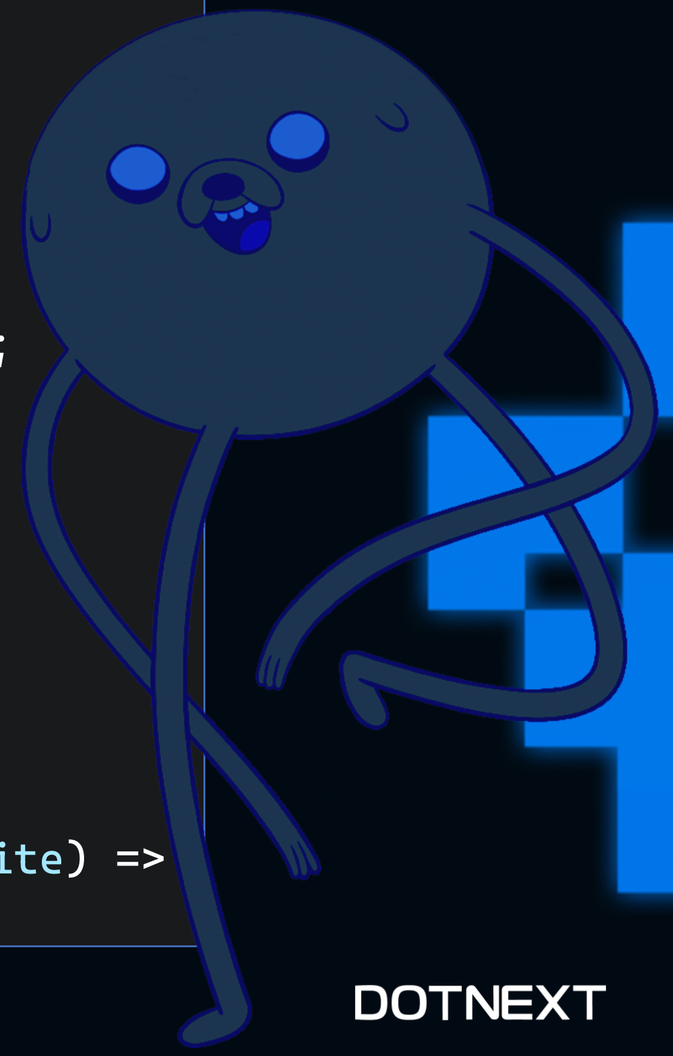
# Пойдём длинным путём

```
private readonly int value;
private readonly TestCaseAttribute testCaseAttribute = new();

public CheckAttribute(int value) => this.value = value;

public object ExpectedResult => testCaseAttribute.ExpectedResult;

public bool HasExpectedResult => testCaseAttribute.HasExpectedResult;

public string TestName => testCaseAttribute.TestName;

public RunState RunState => testCaseAttribute.RunState;

public object[] Arguments => testCaseAttribute.Arguments;

public IPropertyBag Properties => testCaseAttribute.Properties;

public IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite) =>
    testCaseAttribute.BuildFrom(method, suite);
```



# Пойдём длинным путём

```
private readonly int value;
private readonly TestCaseAttribute testCaseAttribute = new();

public CheckAttribute(int value) => this.value = value;

public object ExpectedResult => testCaseAttribute.ExpectedResult;

public bool HasExpectedResult => testCaseAttribute.HasExpectedResult;

public string TestName => testCaseAttribute.TestName;

public RunState RunState => testCaseAttribute.RunState;

public object[] Arguments => testCaseAttribute.Arguments;

public IPropertyBag Properties => testCaseAttribute.Properties;

public IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite) =>
    testCaseAttribute.BuildFrom(method, suite);
```



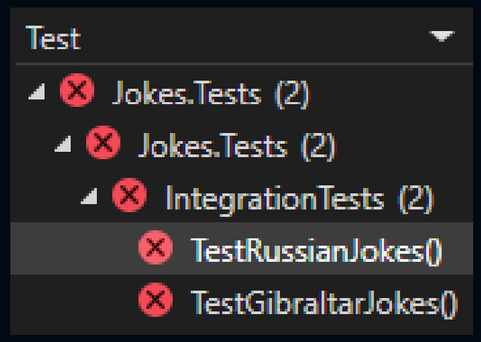
# Пойдём длинным путём

```
private readonly int value;
private readonly TestCaseAttribute testCaseAttribute = new();

public CheckAttribute(int value) => this.value = value;

public object ExpectedResult => testCaseAttribute.ExpectedResult;

public bool HasExpectedResult => testCaseAttribute.HasExpectedResult;

public string TestName => testCaseAttribute.TestName;

public RunState RunState => testCaseAttribute.RunState;

public object[] Arguments => testCaseAttribute.Arguments;

public IPropertyBag Properties => testCaseAttribute.Properties;

public IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite) =>
    testCaseAttribute.BuildFrom(method, suite);
```

# Реализуем цель

```
public override void OnEntry(MethodExecutionArgs arg)
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Не работает




**Test Detail Summary**

TestRussianJokes()

Source: IntegrationTests.cs line 13

Test has multiple result outcomes

2 Failed

Results

1) TestRussianJokes()

Duration: < 1 ms

Message:

```
System.InvalidOperationException : Could not execute the method because either the method itself or the containing type is not fully instantiated.
```

Stack Trace:

```
CheckAttribute`3.ctor(Int32 value)
IntegrationTests.TestRussianJokes()
```

2) TestRussianJokes()

Duration: < 1 ms

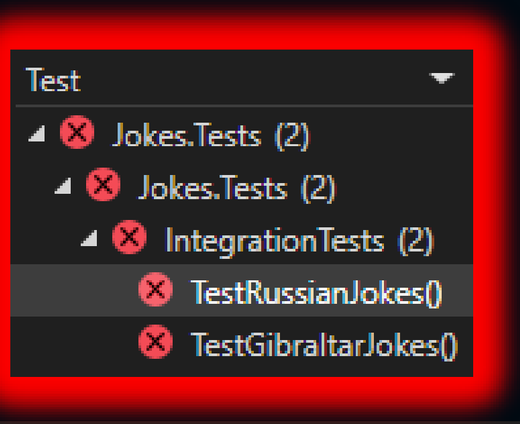
Message:

```
System.InvalidOperationException : Could not execute the method because either the method itself or the containing type is not fully instantiated.
```

Stack Trace:

```
CheckAttribute`3.ctor(Int32 value)
IntegrationTests.TestRussianJokes()
```

# Не работает




**Test Detail Summary**

TestRussianJokes()

Source: IntegrationTests.cs line 13

Test has multiple result outcomes

2 Failed

Results

1) TestRussianJokes()

Duration: < 1 ms

Message:

```
System.InvalidOperationException : Could not execute the method because either the method itself or the containing type is not fully instantiated.
```

Stack Trace:

```
CheckAttribute`3.ctor(Int32 value)
IntegrationTests.TestRussianJokes()
```

2) TestRussianJokes()

Duration: < 1 ms

Message:

```
System.InvalidOperationException : Could not execute the method because either the method itself or the containing type is not fully instantiated.
```

Stack Trace:

```
CheckAttribute`3.ctor(Int32 value)
IntegrationTests.TestRussianJokes()
```

# Не работает

Test

- Jokes.Tests (2)
  - Jokes.Tests (2)
    - IntegrationTests (2)
      - TestRussianJokes()
      - TestGibraltarJokes()

**Test Detail Summary**

TestRussianJokes()

Source: IntegrationTests.cs line 13

Test has multiple result outcomes

2 Failed

Results

1) TestRussianJokes()

Duration: < 1 ms

Message:

```
System.InvalidOperationException : Could not execute the method because either the method itself or the containing type is not fully instantiated.
```

Stack Trace:

```
CheckAttribute`3.ctor(Int32 value)
IntegrationTests.TestRussianJokes()
```

2) TestRussianJokes()

Duration: < 1 ms

Message:

```
System.InvalidOperationException : Could not execute the method because either the method itself or the containing type is not fully instantiated.
```

Stack Trace:

```
CheckAttribute`3.ctor(Int32 value)
IntegrationTests.TestRussianJokes()
```

# Ужасы под капотом

```
object[] __var_0 = new object[0];
MethodExecutionArgs __var_4 = new MethodExecutionArgs();
__var_4.Arguments = __var_0;
MethodBase __var_5 = (__var_4.Method =
    MethodInfos._methodInfo_16ABF7FC04B7A938132731E1D0376DEB95FCE2A7A434F201040687189A3FD09C);
IntegrationTests __var_1 = (IntegrationTests)(__var_4.Instance = this);
CheckAttribute<RussianJokes, AboutCoders, AreOfMinimalFun> __var_6 =
    (CheckAttribute<RussianJokes, AboutCoders, AreOfMinimalFun>)(object)new CheckAttribute<,,>(75);
CheckAttribute<RussianJokes, ForParty, AreOfMinimalFun> __var_8 =
    (CheckAttribute<RussianJokes, ForParty, AreOfMinimalFun>)(object)new CheckAttribute<,,>(75);
((CheckAttribute<,,>)(object)__var_6).OnEntry(__var_4);
object __var_7 = __var_4.MethodExecutionTag;
FlowBehavior __var_2 = __var_4.FlowBehavior;
if (__var_2 != FlowBehavior.Return)
{
    ((CheckAttribute<,,>)(object)__var_8).OnEntry(__var_4);
    object __var_9 = __var_4.MethodExecutionTag;
    FlowBehavior __var_3 = __var_4.FlowBehavior;
    if (__var_3 != FlowBehavior.Return)
    {
        $_executor_TestRussianJokes();
    }
}
```

# Ужасы под капотом

```
object[] __var_0 = new object[0];
MethodExecutionArgs __var_4 = new MethodExecutionArgs();
__var_4.Arguments = __var_0;
MethodBase __var_5 = (__var_4.Method =
    MethodInfos._methodInfo_16ABF7FC04B7A938132731E1D0376DEB95FCE2A7A434F201040687189A3FD09C);
IntegrationTests __var_1 = (IntegrationTests)(__var_4.Instance = this);
CheckAttribute<RussianJokes, AboutCoders, AreOfMinimalFun> __var_6 =
    (CheckAttribute<RussianJokes, AboutCoders, AreOfMinimalFun>)(object)new CheckAttribute<,,>(75);
CheckAttribute<RussianJokes, ForParty, AreOfMinimalFun> __var_8 =
    (CheckAttribute<RussianJokes, ForParty, AreOfMinimalFun>)(object)new CheckAttribute<,,>(75);
((CheckAttribute<,,>)(object)__var_6).OnEntry(__var_4);
object __var_7 = __var_4.MethodExecutionTag;
FlowBehavior __var_2 = __var_4.FlowBehavior;
if (__var_2 != FlowBehavior.Return)
{
    ((CheckAttribute<,,>)(object)__var_8).OnEntry(__var_4);
    object __var_9 = __var_4.MethodExecutionTag;
    FlowBehavior __var_3 = __var_4.FlowBehavior;
    if (__var_3 != FlowBehavior.Return)
    {
        $_executor_TestRussianJokes();
    }
}
```

# Новый план

```
[TestFixture]
public class IntegrationTests
{
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestRussianJokes() { }

    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
}
```

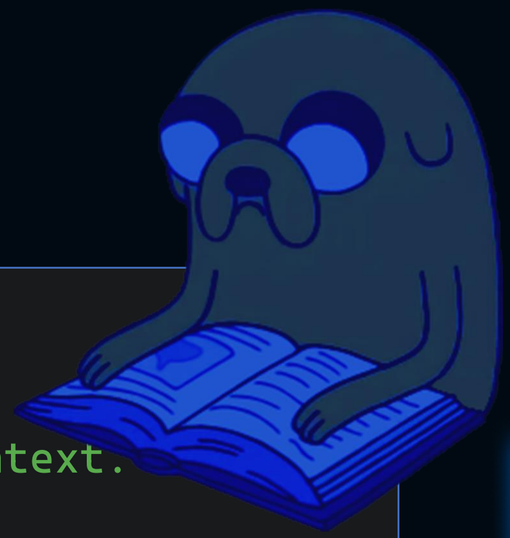
# Новый план

```
[TestFixture]
public class IntegrationTests
{
    [Checkable]
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestRussianJokes() { }

    [Checkable]
    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
}
```

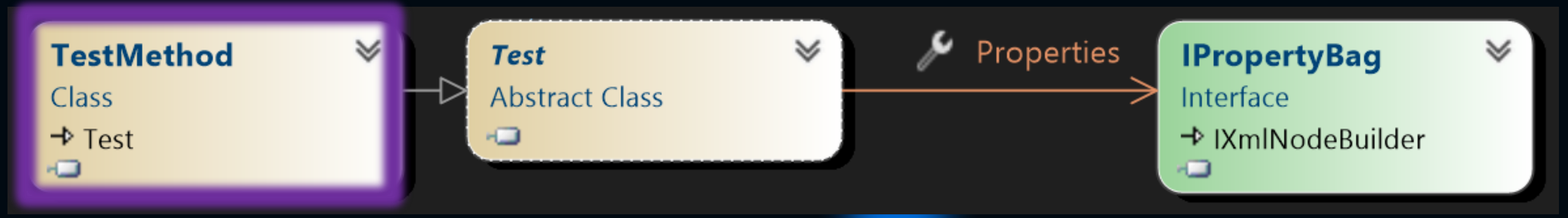
# Новый план

```
public class CheckableAttribute : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs arg)
    {
        // ???
    }
}
```

# Следите за руками

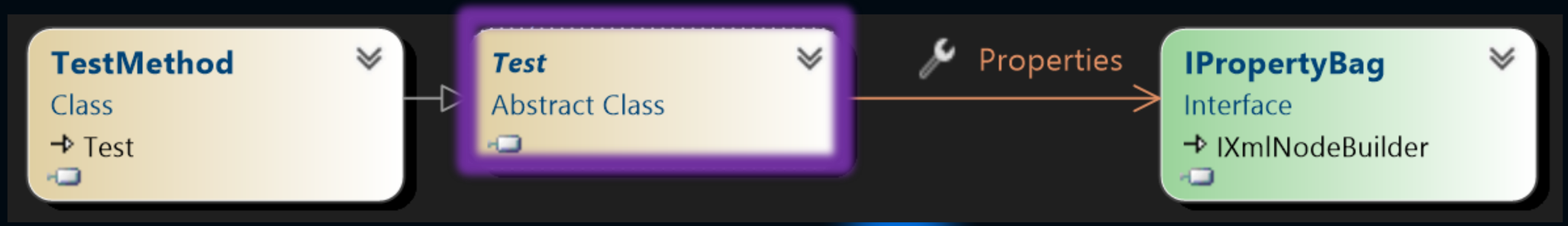


```
public interface ITestBuilder
{
    /// <summary>
    /// Builds any number of tests from the specified method and context.
    /// </summary>
    /// <param name="method">The method to be used as a test</param>
    /// <param name="suite">The TestSuite to which the method will be added</param>
    IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test? suite);
}
```



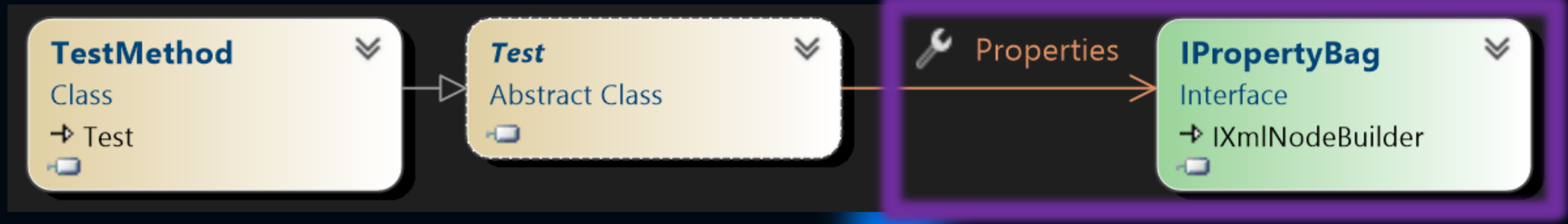
# Следите за руками

```
/// <summary>
/// The TestMethod class represents a Test implemented as a method.
/// </summary>
public class TestMethod : Test
{
    // ...
}
```



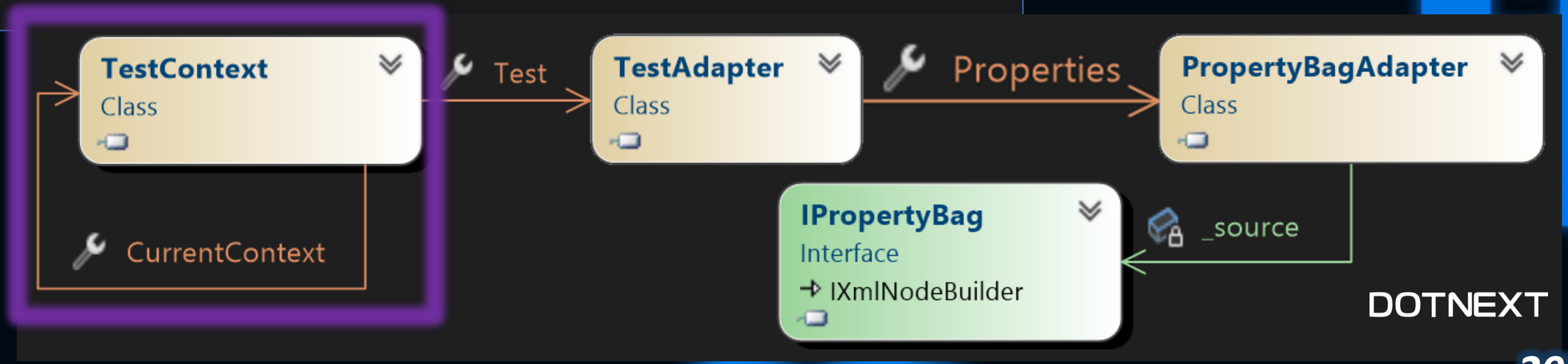
# Следите за руками

```
/// <summary>
/// The Test abstract class represents a test within the framework.
/// </summary>
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    /// <summary>
    /// Gets the properties for this test
    /// </summary>
    public IPropertyBag Properties { get; }
}
```



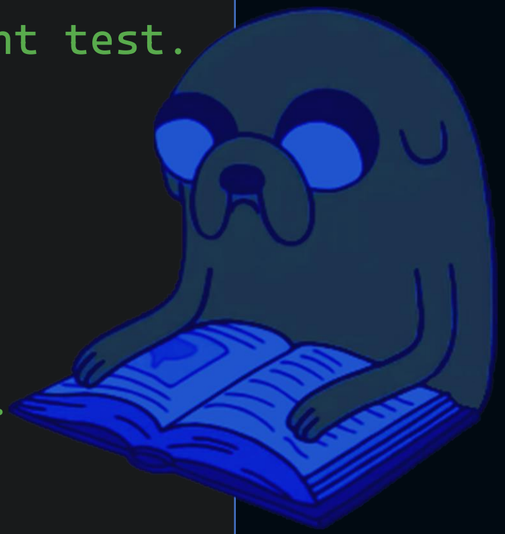
# Следите за руками

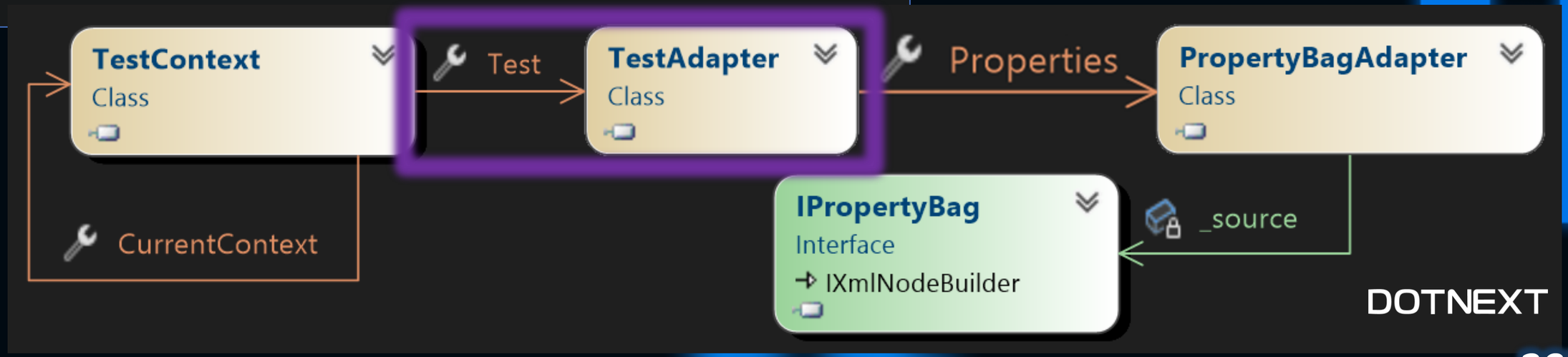
```
/// <summary>
/// Provide the context information of the current test.
/// </summary>
public class TestContext
{
    // ...

    /// <summary>
    /// Get the current test context.
    /// </summary>
    public static TestContext CurrentContext => /* ... */
}
```



# Следите за руками

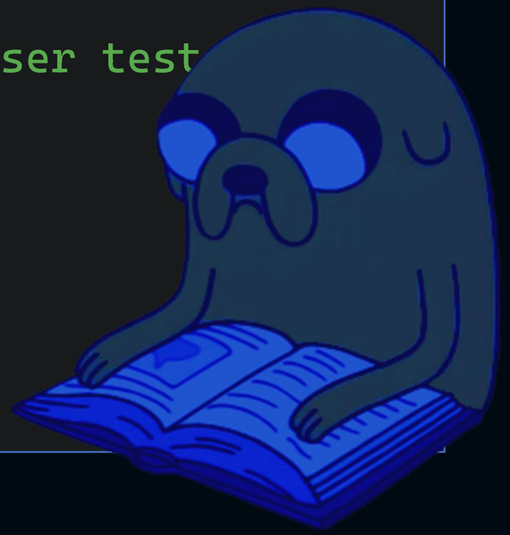
```
/// <summary>
/// Provide the context information of the current test.
/// </summary>
public class TestContext
{
    // ...

    /// <summary>
    /// Get a representation of the current test.
    /// </summary>
    public TestAdapter Test => /* ... */
}
```

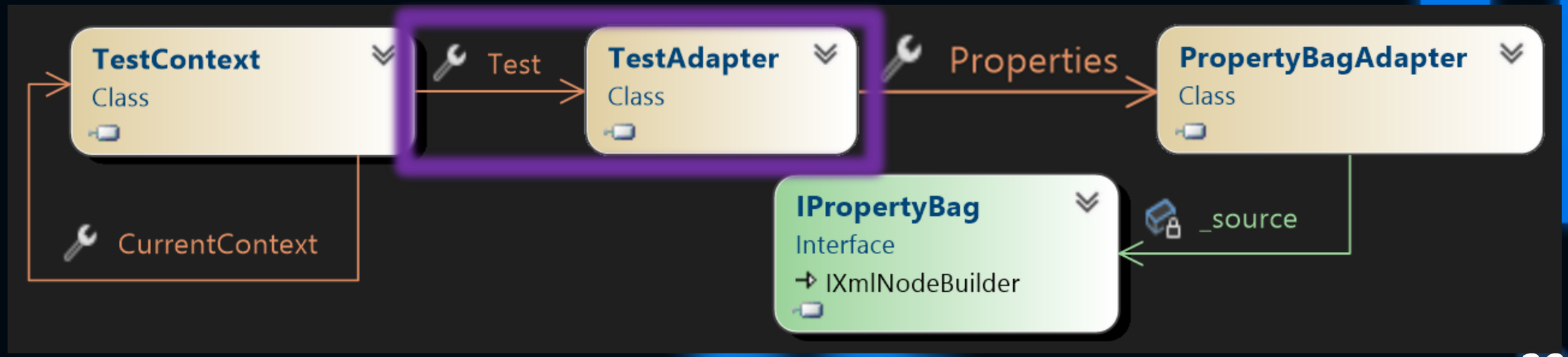




# Следите за руками

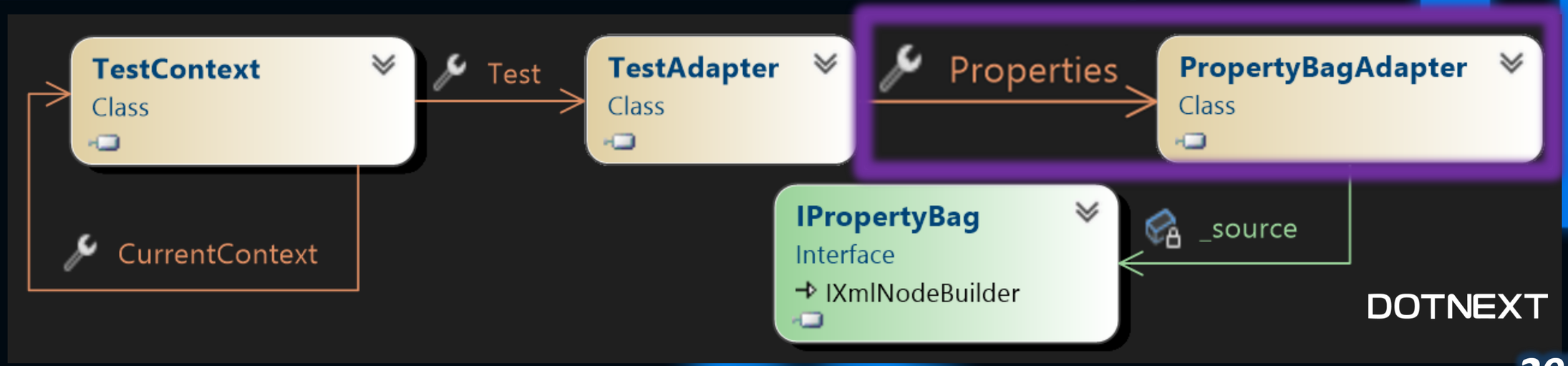
```
/// <summary>
/// TestAdapter adapts a Test for consumption by the user test
/// </summary>
public class TestAdapter
{
    private readonly Test _test;

    // ...
}
```





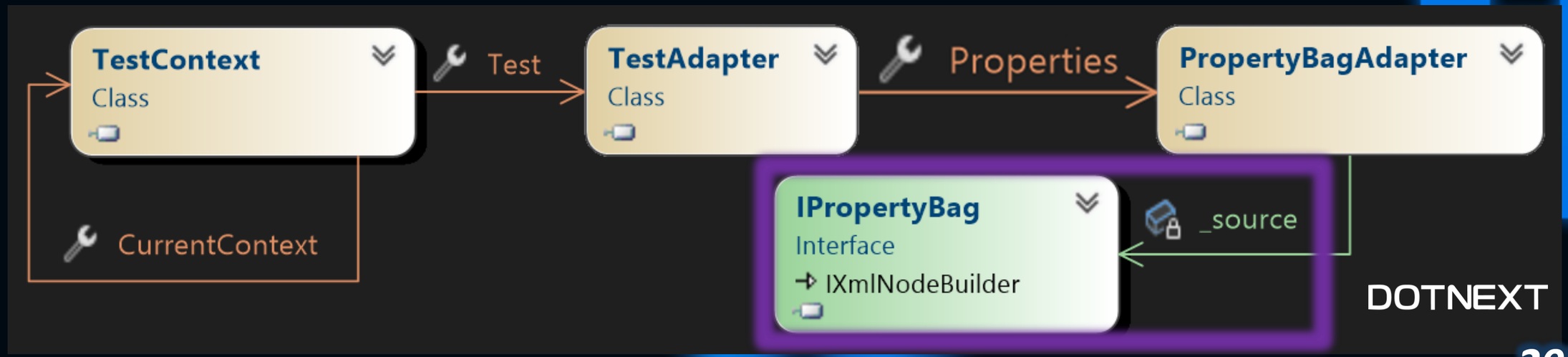
# Следите за руками

```
/// <summary>
/// TestAdapter adapts a Test for consumption by the user test code.
/// </summary>
public class TestAdapter
{
    // ...

    /// <summary>
    /// A shallow copy of the properties of the test.
    /// </summary>
    public PropertyBagAdapter Properties => /* ... */
}
```

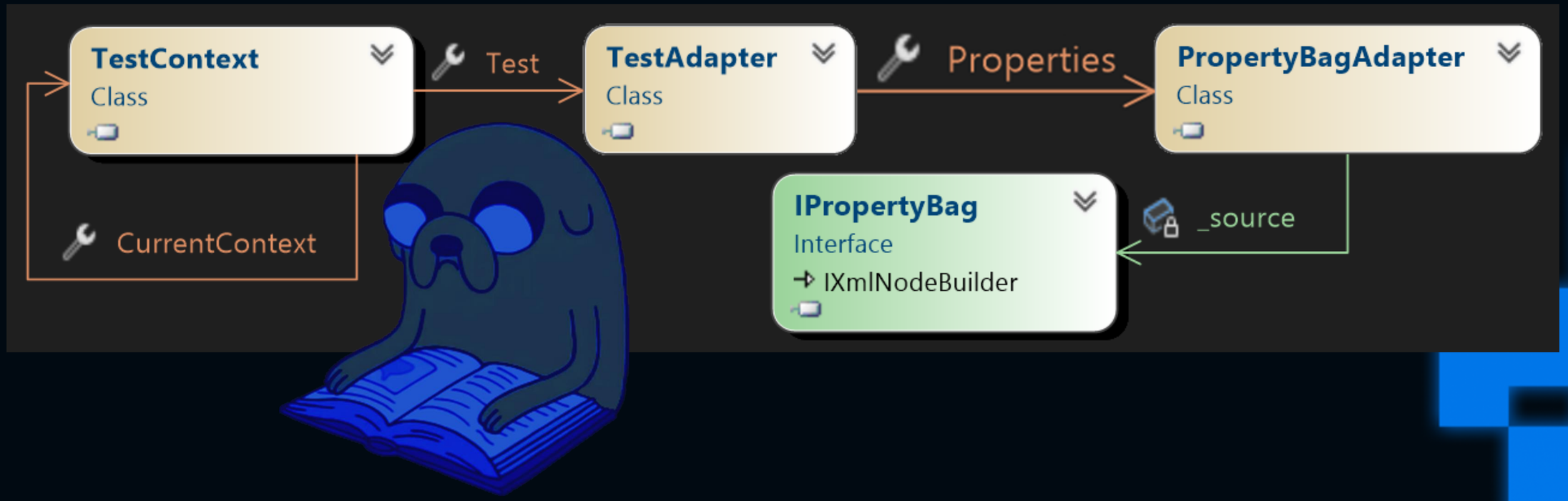


# Следите за руками

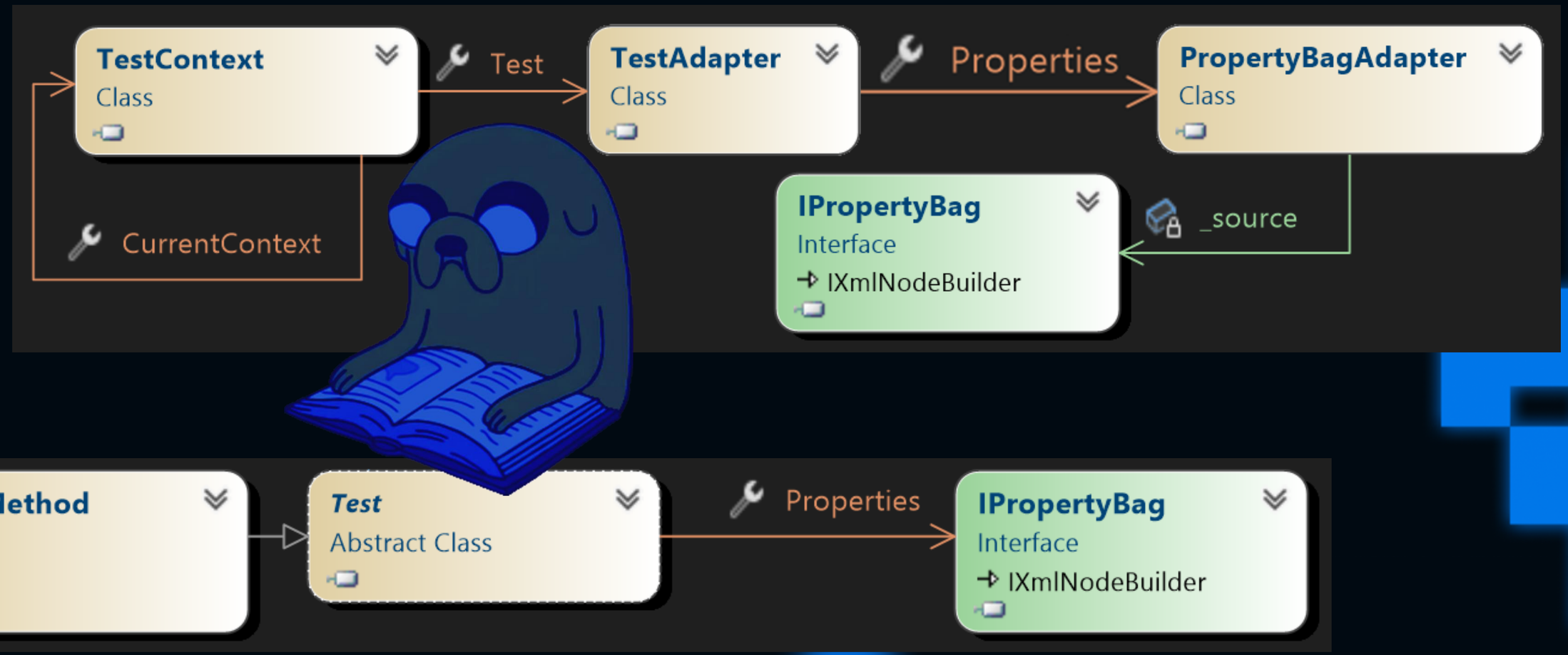
```
/// <summary>
/// TestAdapter adapts a Test for consumption by the user test code.
/// </summary>
public class TestAdapter
{
    // ...

    /// <summary>
    /// A shallow copy of the properties of the test.
    /// </summary>
    public PropertyBagAdapter Properties => /* ... */
}
```



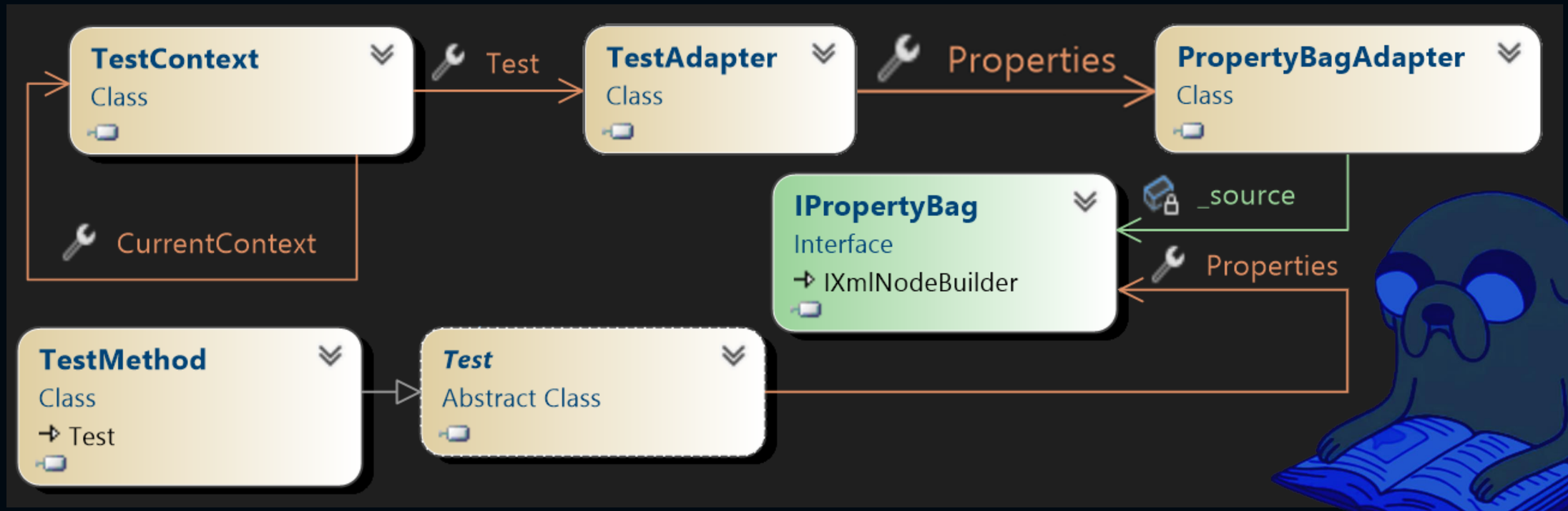
# Следите за руками



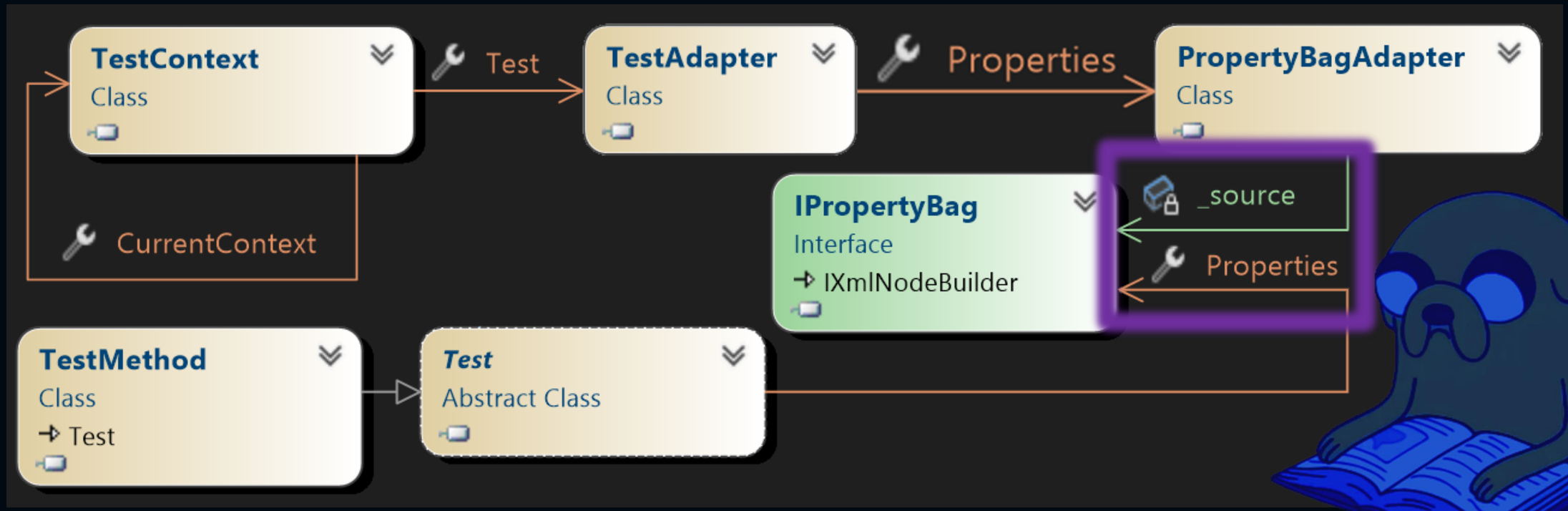
# Следите за руками



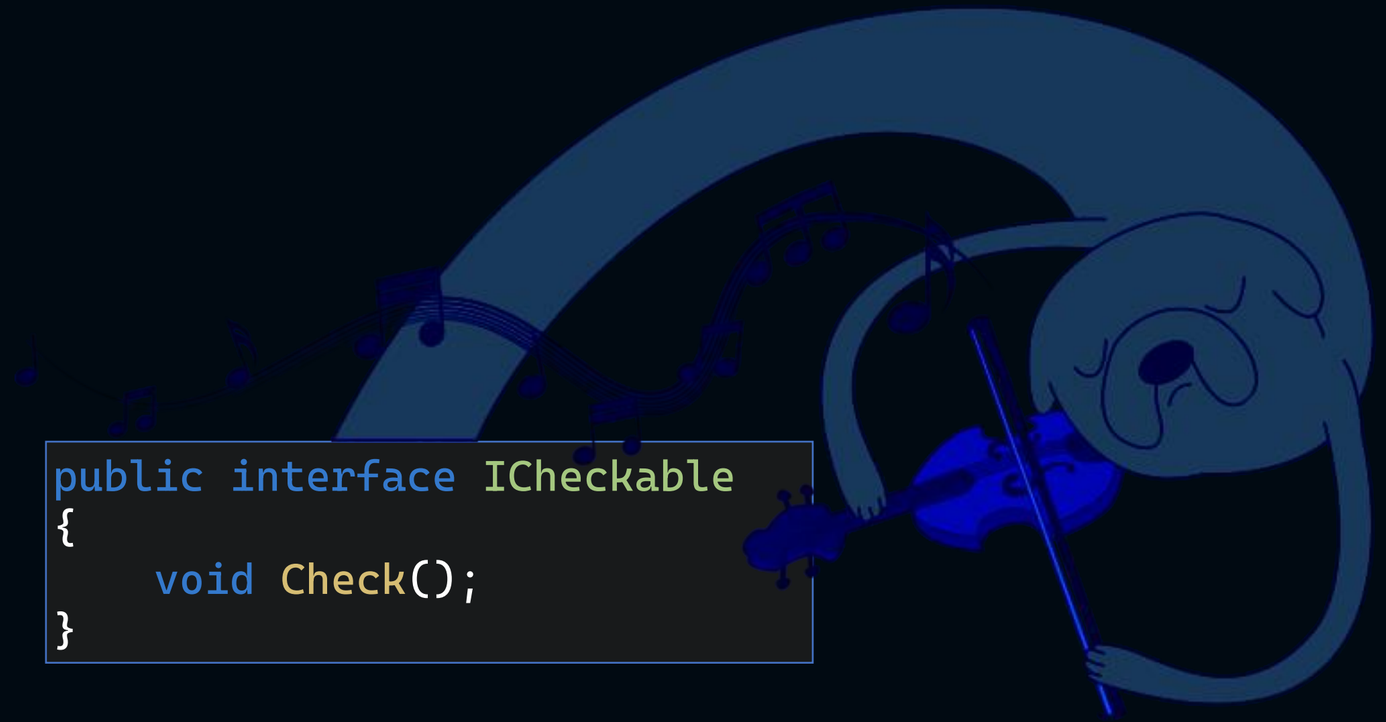
# Следите за руками



# Следите за руками



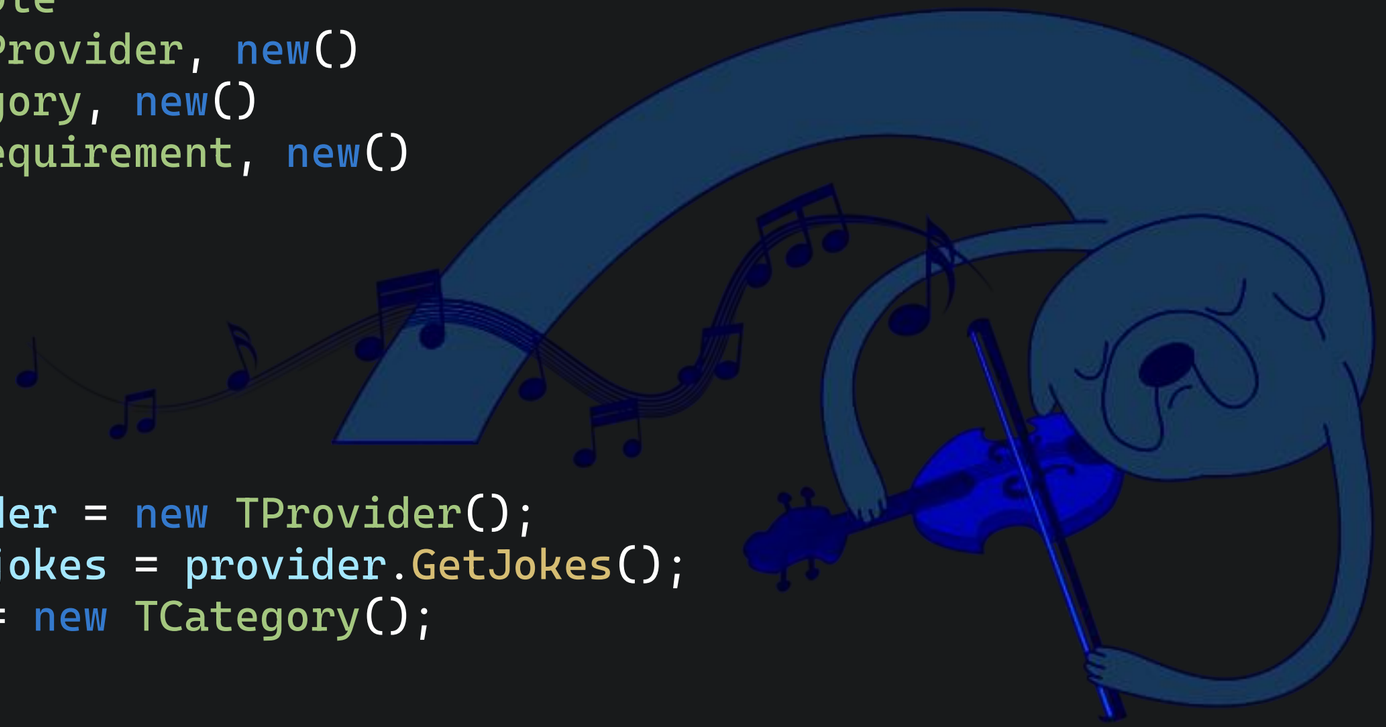
# Подготовка



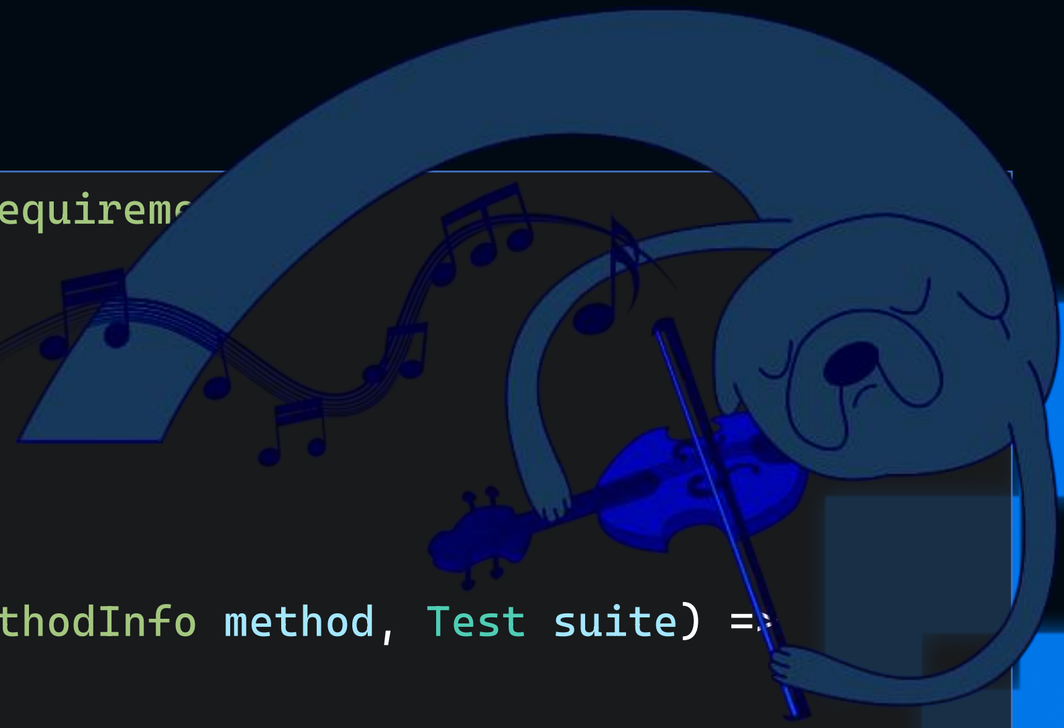
```
public interface ICheckable
{
    void Check();
}
```

# Подготовка

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    GCaseAttribute, ICheckable
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void Check()
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```



# Подготовка



```
public class CheckAttribute<TProvider, TCategory, TRequireme
    GCaseAttribute, Icheckable, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite) =>
        base.BuildFrom(method, suite).Select(test =>
        {
            test.Properties.Add(
                typeof(CheckAttribute<TProvider, TCategory, TRequirement>).FullName,
                this);

            return test;
        });
}
```

# Решение

```
public class CheckableAttribute : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs arg)
    {
        // ???
    }
}
```

# Решение

```
public class CheckableAttribute : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs arg)
    {
        var test = TestContext.CurrentContext.Test;

        test.Properties.Keys.SelectMany(key => test.Properties[key])
            .OfType<ICheckable>().Single().Check();
    }
}
```

# Решение

```
public class CheckableAttribute : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs arg)
    {
        var test = TestContext.CurrentContext.Test;

        test.Properties.Keys.SelectMany(key => test.Properties[key])
            .OfType<ICheckable>().Single().Check();
    }
}
```

# Решение

```
public class CheckableAttribute : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs arg)
    {
        var test = TestContext.CurrentContext.Test;

        test.Properties.Keys.SelectMany(key => test.Properties[key])
            .OfType<ICheckable>().Single().Check();
    }
}
```

# Решение

Test

- Jokes.Tests (2)
  - Jokes.Tests (2)
    - IntegrationTests (2)
      - TestRussianJokes()
      - TestGibraltarJokes()

**Test Detail Summary**

TestRussianJokes()

Source: IntegrationTests.cs line 14

Test has multiple result outcomes

2 Passed

Results

1) TestRussianJokes()
   Duration: < 1 ms

2) TestRussianJokes()
   Duration: < 1 ms

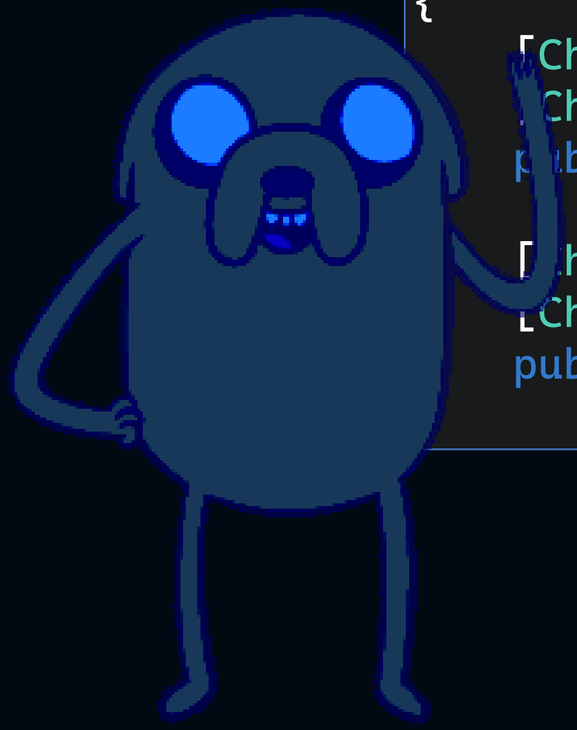
# Что-то смущает?

```
[TestFixture]
public class IntegrationTests
{
    [Checkable]
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestRussianJokes() { }

    [Checkable]
    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
}
```

# Что-то смущает?

```
[TestFixture]
public class IntegrationTests
{
    [Checkable]
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestRussianJokes() { }

    [Checkable]
    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
}
```

# Что-то смущает?



```
[TestFixture, Checkable]
public class IntegrationTests
{
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestRussianJokes() { }

    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
```

# Занимательный факт

TestFixtureAttribute не является необходимым для обнаружения тестов

# Check Point

- ✓ Наличие атрибута NUnit.Framework.TestFixtureAttribute не является необходимым для обнаружения теста
- ✓ Необходимо наличие реализации интерфейса-маркера NUnit.Framework.Interfaces.IImplyFixture
- ✓ NUnit.Framework.TestContext.CurrentContext.Test даёт доступ к текущему тесту NUnit.Framework.Internal.Test через адаптер NUnit.Framework.TestContext.TestAdapter
- ✓ TestAdapter через PropertyBagAdapter Properties даёт доступ к IPropertyBag Properties теста Test, в котором можно хранить полезные данные

# Check Point

- ✓ Наличие атрибута NUnit.Framework.TestFixtureAttribute не является необходимым для обнаружения теста
- ✓ Необходимо наличие реализации интерфейса-маркера NUnit.Framework.Interfaces.IImplyFixture
- ✓ NUnit.Framework.TestContext.CurrentContext.Test даёт доступ к текущему тесту NUnit.Framework.Internal.Test через адаптер NUnit.Framework.TestContext.TestAdapter
- ✓ TestAdapter через PropertyBagAdapter Properties даёт доступ к IPropertyBag Properties теста Test, в котором можно хранить полезные данные

# Check Point

- ✓ Наличие атрибута NUnit.Framework.TestFixtureAttribute не является необходимым для обнаружения теста
- ✓ Необходимо наличие реализации интерфейса-маркера NUnit.Framework.Interfaces.IImplyFixture
- ✓ NUnit.Framework.TestContext.CurrentContext.Test даёт доступ к текущему тесту NUnit.Framework.Internal.Test через адаптер NUnit.Framework.TestContext.TestAdapter
- ✓ TestAdapter через PropertyBagAdapter Properties даёт доступ к IPropertyBag Properties теста Test, в котором можно хранить полезные данные

# Check Point

- ✓ Наличие атрибута NUnit.Framework.TestFixtureAttribute не является необходимым для обнаружения теста
- ✓ Необходимо наличие реализации интерфейса-маркера NUnit.Framework.Interfaces.IImplyFixture
- ✓ NUnit.Framework.TestContext.CurrentContext.Test даёт доступ к текущему тесту NUnit.Framework.Internal.Test через адаптер NUnit.Framework.TestContext.TestAdapter
- ✓ TestAdapter через PropertyBagAdapter Properties даёт доступ к IPropertyBag Properties теста Test, в котором можно хранить полезные данные

# Check Point

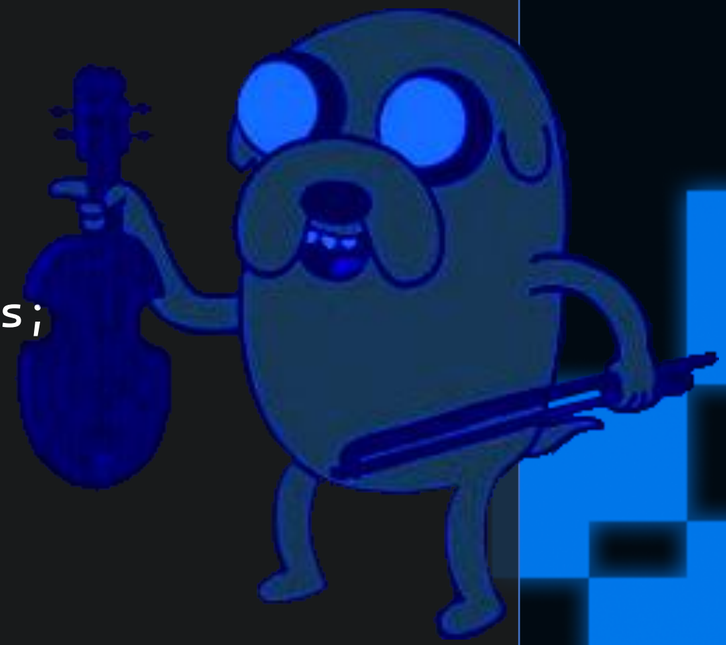
- ✓ Наличие атрибута NUnit.Framework.TestFixtureAttribute не является необходимым для обнаружения теста
- ✓ Необходимо наличие реализации интерфейса-маркера NUnit.Framework.Interfaces.IImplyFixture
- ✓ NUnit.Framework.TestContext.CurrentContext.Test даёт доступ к текущему тесту NUnit.Framework.Internal.Test через адаптер NUnit.Framework.TestContext.TestAdapter
- ✓ TestAdapter через PropertyBagAdapter Properties даёт доступ к IPropertyBag Properties теста Test, в котором можно хранить полезные данные

# Занимательный факт

```
/// <summary>
/// When implemented by an attribute, this interface implemented
/// to provide actions to execute before and after tests.
/// </summary>
public interface ITestAction
{
    /// <summary>
    /// Executed before each test is run
    /// </summary>
    void BeforeTest(ITest test);

    /// <summary>
    /// Executed after each test is run
    /// </summary>
    void AfterTest(ITest test);

    /// <summary>
    /// Provides the target for the action attribute
    /// </summary>
    ActionTargets Targets { get; }
}
```

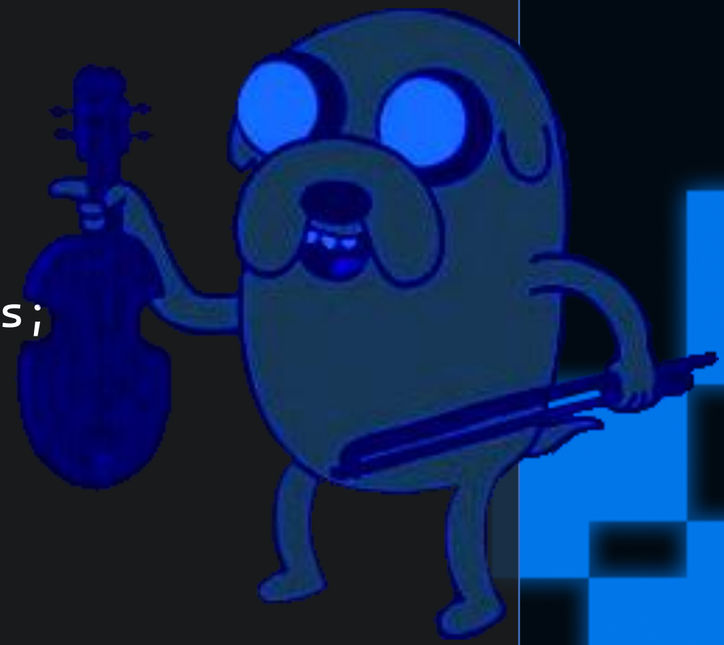
# Новая надежда

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;
    private readonly IReadOnlyCollection<Type> typeArguments;

    public CheckAttribute(int value)
    {
        this.value = value;
        typeArguments = GetType().GetGenericArguments();
    }

    public ActionTargets Targets => ActionTargets.Test;

    public void BeforeTest(ITest test) { }

    // ...
}
```



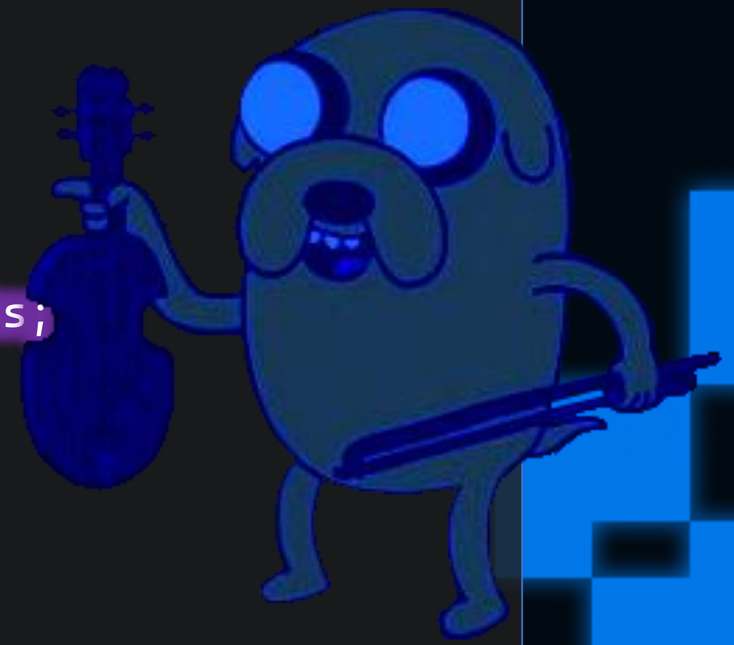
# Новая надежда

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;
    private readonly IReadOnlyCollection<Type> typeArguments;

    public CheckAttribute(int value)
    {
        this.value = value;
        typeArguments = GetType().GetGenericArguments();
    }

    public ActionTargets Targets => ActionTargets.Test;

    public void BeforeTest(ITest test) { }

    // ...
}
```



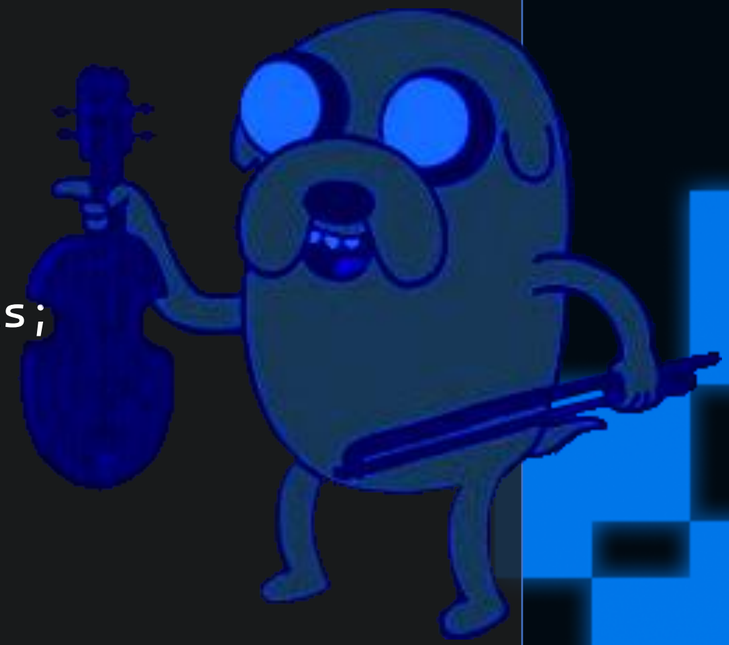
# Новая надежда

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;
    private readonly IReadOnlyCollection<Type> typeArguments;

    public CheckAttribute(int value)
    {
        this.value = value;
        typeArguments = GetType().GetGenericArguments();
    }

    public ActionTargets Targets => ActionTargets.Test;

    public void BeforeTest(ITest test) { }

    // ...
}
```



# Новая надежда

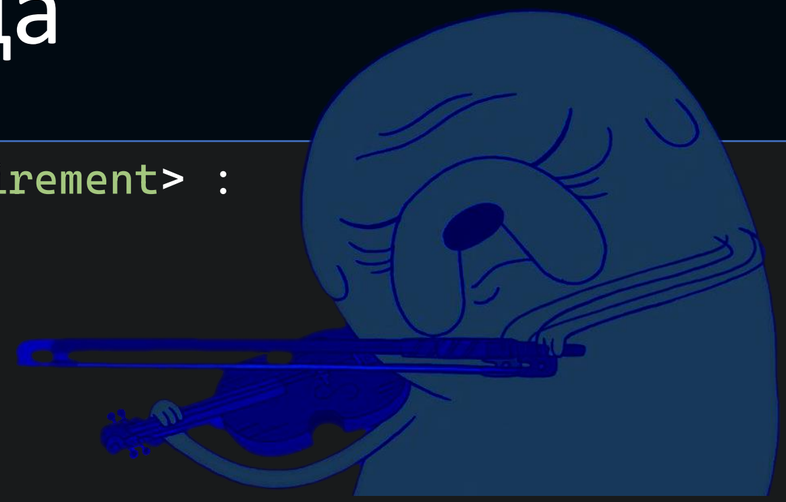
```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;
    private readonly IReadOnlyCollection<Type> typeArguments;

    public CheckAttribute(int value)
    {
        this.value = value;
        typeArguments = GetType().GetGenericArguments();
    }

    public ActionTargets Targets => ActionTargets.Test;

    public void BeforeTest(ITest test) { }

    // ...
}
```



# Новая надежда

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void AfterTest(ITest test)
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.IsTrue(actual);
    }
}
```

# Новая надежда



```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite) =>
        base.BuildFrom(method, suite).Select(test =>
        {
            test.Name = $"{method.Name}<{
                string.Join(',', typeArguments.Select(type => type.Name))}>({value})";
            test.FullName = $"{method.MethodInfo.DeclaringType.FullName}.{
                test.Name}";

            return test;
        });
}
```

# Изменим тест

```
[Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[Check<RussianJokes, ForParty, AreOfMinimalFun>(76)]
public void TestRussianJokes() { }

[Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void TestGibraltarJokes() { }
```

Сколько тестов упадёт?

0

1

2

4

# Изменим тест

Test

- Jokes.Tests (4)
  - Jokes.Tests (4)
    - IntegrationTests (4)
      - TestRussianJokes (2)
        - TestRussianJokes<RussianJokes,ForParty,AreOfMinimalFun>(76)
        - TestRussianJokes<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
      - TestGibraltarJokes (2)
        - *GibraltarJokes<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - raltarJokes<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)

**Group Summary**

Jokes.Tests

Tests in group: 4

Total Duration: 42 ms

Outcomes

2 Passed

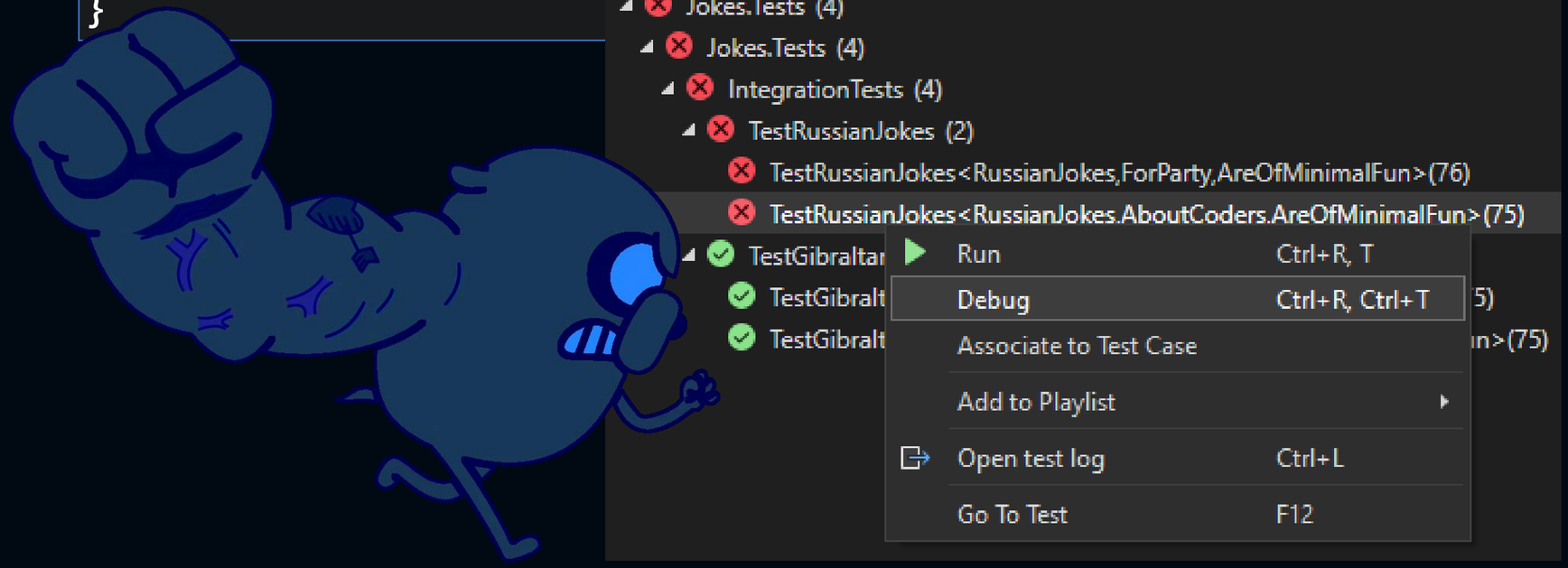
2 Failed

Сколько тестов упадёт?

- 0
- 1
- 2
- 4

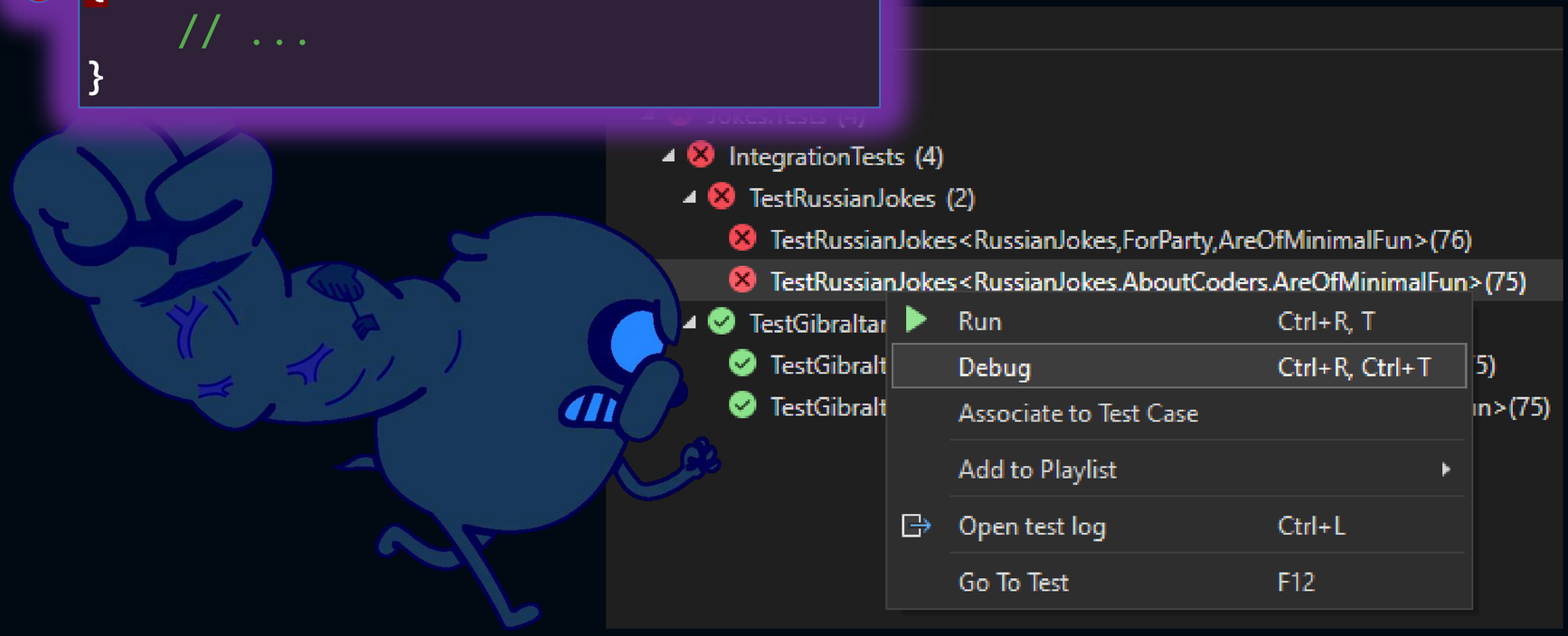
# Дебажим!!!

```
public void AfterTest(ITest test)
{
    // ...
}
```



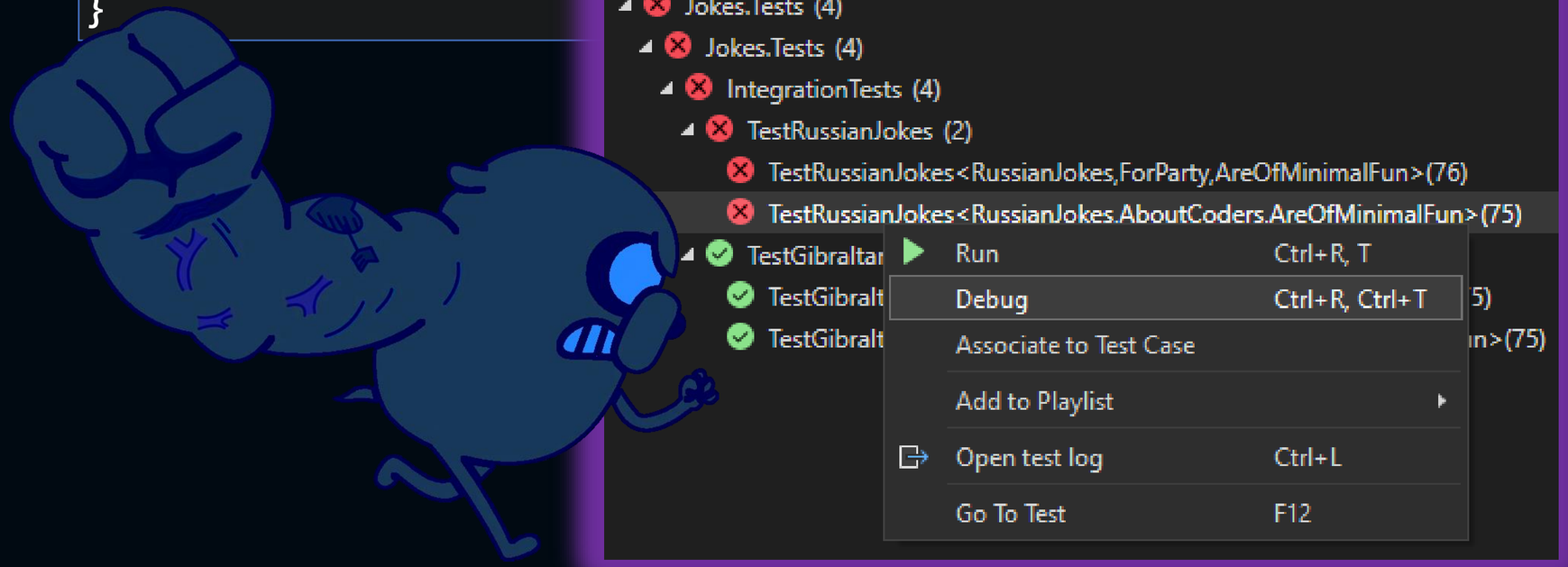
# Дебажим!!!

```
public void AfterTest(ITest test)
{
    // ...
}
```



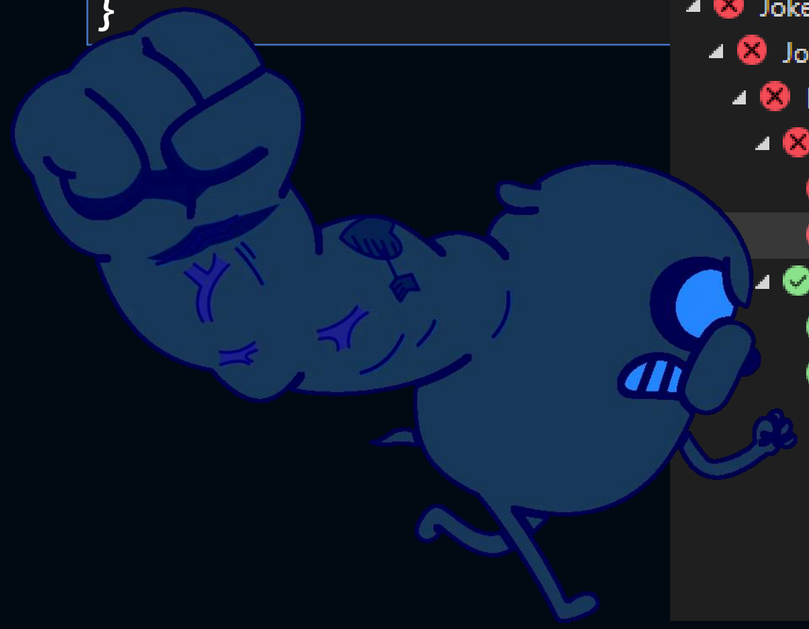
# Дебажим!!!

```
public void AfterTest(ITest test)
{
    // ...
}
```



# Дебажим!!!

```
public void AfterTest(ITest test)
{
    // ...
}
```



Test

- Jokes.Tests (4)
  - Jokes.Tests (4)
    - IntegrationTests (4)
      - TestRussianJokes (2)
        - TestRussianJokes<RussianJokes,ForParty,AreOfMinimalFun>(76)
        - TestRussianJokes<RussianJokes.AboutCoders.AreOfMinimalFun>(75)
      - TestGibraltar
        - TestGibralt
        - TestGibralt

Run Ctrl+R, T
Debug Ctrl+R, Ctrl+T
Associate to Test Case
Add to Playlist
Open test log Ctrl+L
Go To Test F12

# Дебажим!!!

```
public void AfterTest(ITest test)
{
    // ...
}
```

value 75

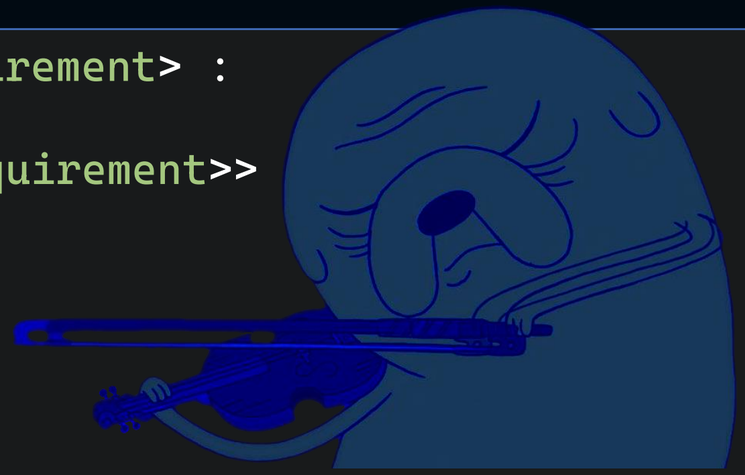
# Дебажим!!!

```
public void AfterTest(ITest test)
{
    // ...
}
```

value 76

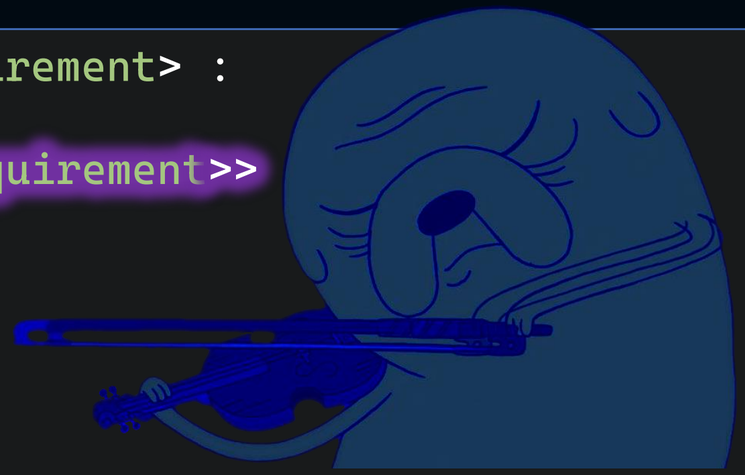
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder,
    IEquatable<CheckAttribute<TProvider, TCategory, TRequirement>>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public bool Equals(CheckAttribute<TProvider, TCategory, TRequirement> other) =>
        value == other.value;

    public override bool Equals(object obj) =>
        obj is CheckAttribute<TProvider, TCategory, TRequirement> other &&
        Equals(other);
}
```



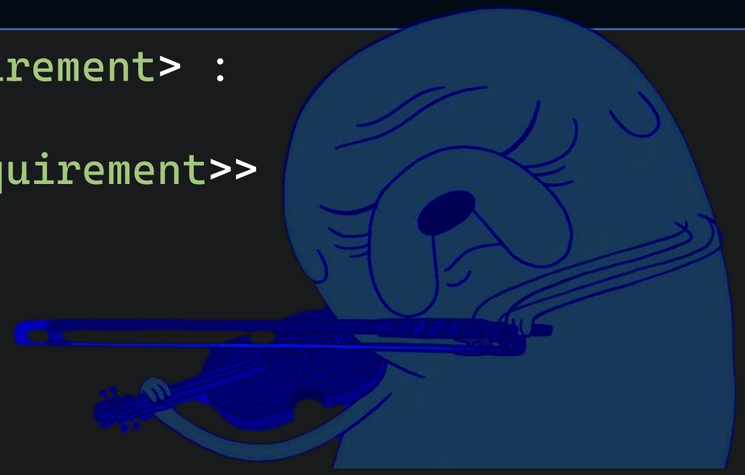
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder,
    IEquatable<CheckAttribute<TProvider, TCategory, TRequirement>>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public bool Equals(CheckAttribute<TProvider, TCategory, TRequirement> other) =>
        value == other.value;

    public override bool Equals(object obj) =>
        obj is CheckAttribute<TProvider, TCategory, TRequirement> other &&
        Equals(other);
}
```



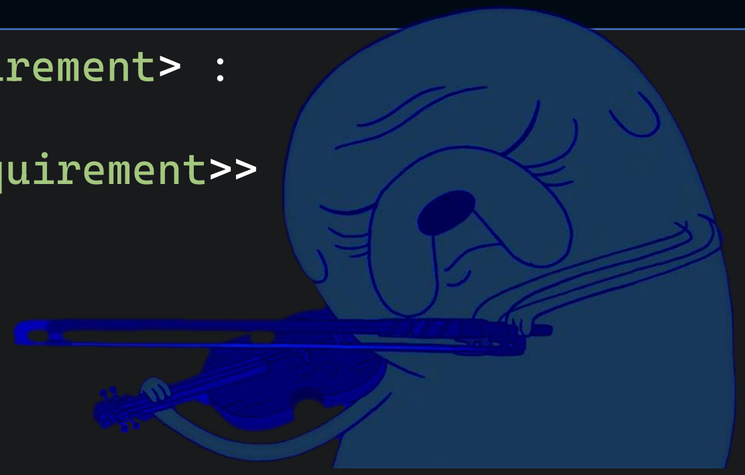
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder,
    IEquatable<CheckAttribute<TProvider, TCategory, TRequirement>>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public bool Equals(CheckAttribute<TProvider, TCategory, TRequirement> other) =>
        value == other.value;

    public override bool Equals(object obj) =>
        obj is CheckAttribute<TProvider, TCategory, TRequirement> other &&
        Equals(other);
}
```



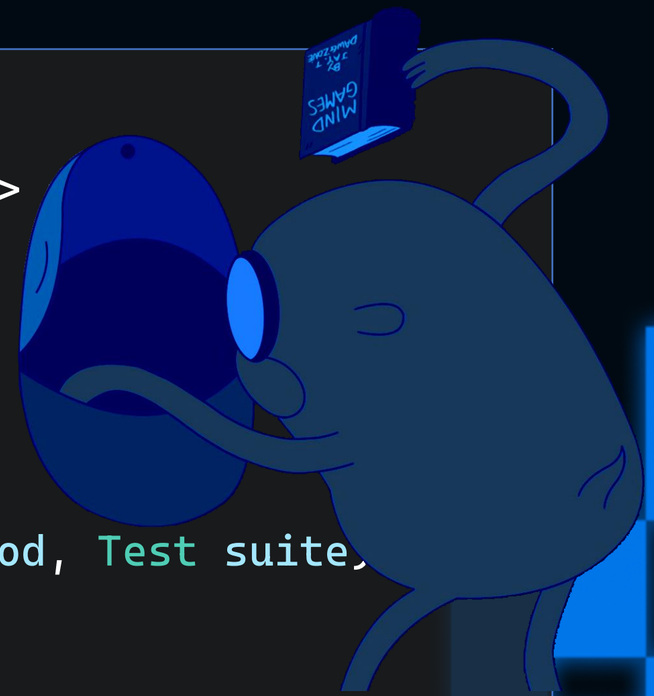
# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder,
    IEquatable<CheckAttribute<TProvider, TCategory, TRequirement>>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public bool Equals(CheckAttribute<TProvider, TCategory, TRequirement> other) =>
        value == other.value;

    public override bool Equals(object obj) =>
        obj is CheckAttribute<TProvider, TCategory, TRequirement> other &&
        Equals(other);
}
```



# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder,
    IEquatable<CheckAttribute<TProvider, TCategory, TRequirement>>
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public new IEnumerable<TestMethod> BuildFrom(IMethodInfo method, Test suite)
        base.BuildFrom(method, suite).Select(test =>
        {
            test.Properties.Add(
                typeof(CheckAttribute<TProvider, TCategory, TRequirement>).FullName,
                this);

            // ...
        });
}
```



# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void AfterTest(ITest test)
    {
        var attribute = test.Properties.Keys
            .SelectMany(key => test.Properties[key].Cast<object>())
            .OfType<TestCaseAttribute>()
            .Single();

        if (!Equals(attribute))
            return;

        // ...
    }
}
```

# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void AfterTest(ITest test)
    {
        var attribute = test.Properties.Keys
            .SelectMany(key => test.Properties[key].Cast<object>())
            .OfType<TestCaseAttribute>()
            .Single();

        if (!Equals(attribute))
            return;

        // ...
    }
}
```

# Решение

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    TestCaseAttribute, ITestAction, ITestBuilder
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public void AfterTest(ITest test)
    {
        var attribute = test.Properties.Keys
            .SelectMany(key => test.Properties[key].Cast<object>())
            .OfType<TestCaseAttribute>()
            .Single();

        if (!Equals(attribute))
            return;

        // ...
    }
}
```

# Решение

Test

- Jokes.Tests (4)
  - Jokes.Tests (4)
    - IntegrationTests (4)
      - TestRussianJokes (2)
        - TestRussianJokes<RussianJokes,ForParty,AreOfMinimalFun>(76)
        - TestRussianJokes<RussianJokes,AboutCoders,AreOfMinimalFun>(75)
      - TestGibraltarJokes (2)
        - TestGibraltarJokes<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
        - TestGibraltarJokes<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)

roup Summary

Jokes.Tests

Tests in group: 4

Total Duration: 39 ms

Outcomes

3 Passed

1 Failed

# Проблемка

```
[Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[Check<RussianJokes, ForParty, AreOfMinimalFun>(76)]
[Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
// {~100 cultures} x {~50 categories} x {~20 requirements} = {+100500 other cases}
public void Test() { }
```

# Занимательный факт

```
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    internal ITestAction[] Actions
    {
        get
        {
            if (_actions == null)
            {
                if (Method == null && TypeInfo != null)
                    _actions = GetActionsForType(TypeInfo.Type);
                else if (Method != null)
                    _actions = MethodInfoCache.Get(Method).TestActionAttributes;
                else
                    _actions = GetCustomAttributes<ITestAction>(false);
            }

            return _actions;
        }
    }

    // ...
}
```

# Занимательный факт

```
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    internal ITestAction[] Actions
    {
        get
        {
            if (_actions == null)
            {
                if (Method == null && TypeInfo != null)
                    _actions = GetActionsForType(TypeInfo.Type);
                else if (Method != null)
                    _actions = MethodInfoCache.Get(Method).TestActionAttributes;
                else
                    _actions = GetCustomAttributes<ITestAction>(false);
            }

            return _actions;
        }
    }

    // ...
}
```

DOTNEXT

# Занимательный факт

```
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    internal ITestAction[] Actions
    {
        get
        {
            if (_actions == null)
            {
                if (Method == null && TypeInfo != null)
                    _actions = GetActionsForType(TypeInfo.Type);
                else if (Method != null)
                    _actions = MethodInfoCache.Get(Method).TestActionAttributes;
                else
                    _actions = GetCustomAttributes<ITestAction>(false);
            }

            return _actions;
        }
    }

    // ...
}
```

# Занимательный факт

```
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    internal ITestAction[] Actions
    {
        get
        {
            if (_actions == null)
            {
                if (Method == null && TypeInfo != null)
                    _actions = GetActionsForType(TypeInfo.Type);
                else if (Method != null)
                    _actions = MethodInfoCache.Get(Method).TestActionAttributes;
                else
                    _actions = GetCustomAttributes<ITestAction>(false);
            }

            return _actions;
        }
    }

    // ...
}
```

# Занимательный факт

```
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    /// <summary>
    /// Get custom attributes applied to a test
    /// </summary>
    public virtual TAttr[] GetCustomAttributes<TAttr>(bool inherit)
        where TAttr : class
    {
        if (Method != null)
            return Method.GetCustomAttributes<TAttr>(inherit);

        if (TypeInfo != null)
            return TypeInfo.GetCustomAttributes<TAttr>(inherit);

        return ArrayHelper.Empty<TAttr>();
    }

    // ...
}
```

# Занимательный факт

```
public abstract class Test : ITest, IComparable, IComparable<Test>
{
    // ...

    /// <summary>
    /// Get custom attributes applied to a test
    /// </summary>
    public virtual TAttr[] GetCustomAttributes<TAttr>(bool inherit)
        where TAttr : class
    {
        if (Method != null)
            return Method.GetCustomAttributes<TAttr>(inherit);

        if (TypeInfo != null)
            return TypeInfo.GetCustomAttributes<TAttr>(inherit);

        return ArrayHelper.Empty<TAttr>();
    }

    // ...
}
```

# Check Point

- ✓ NUnit.Framework.ITestAction — элемент АОП в Nunit
- ✓ В рамках одного тестового метода каждая реализация ITestAction применяется к каждой реализации IImplyFixture

# Check Point

- ✓ NUnit.Framework.ITestAction — элемент АОП в Nunit
- ✓ В рамках одного тестового метода каждая реализация ITestAction применяется к каждой реализации IImplyFixture

# Check Point

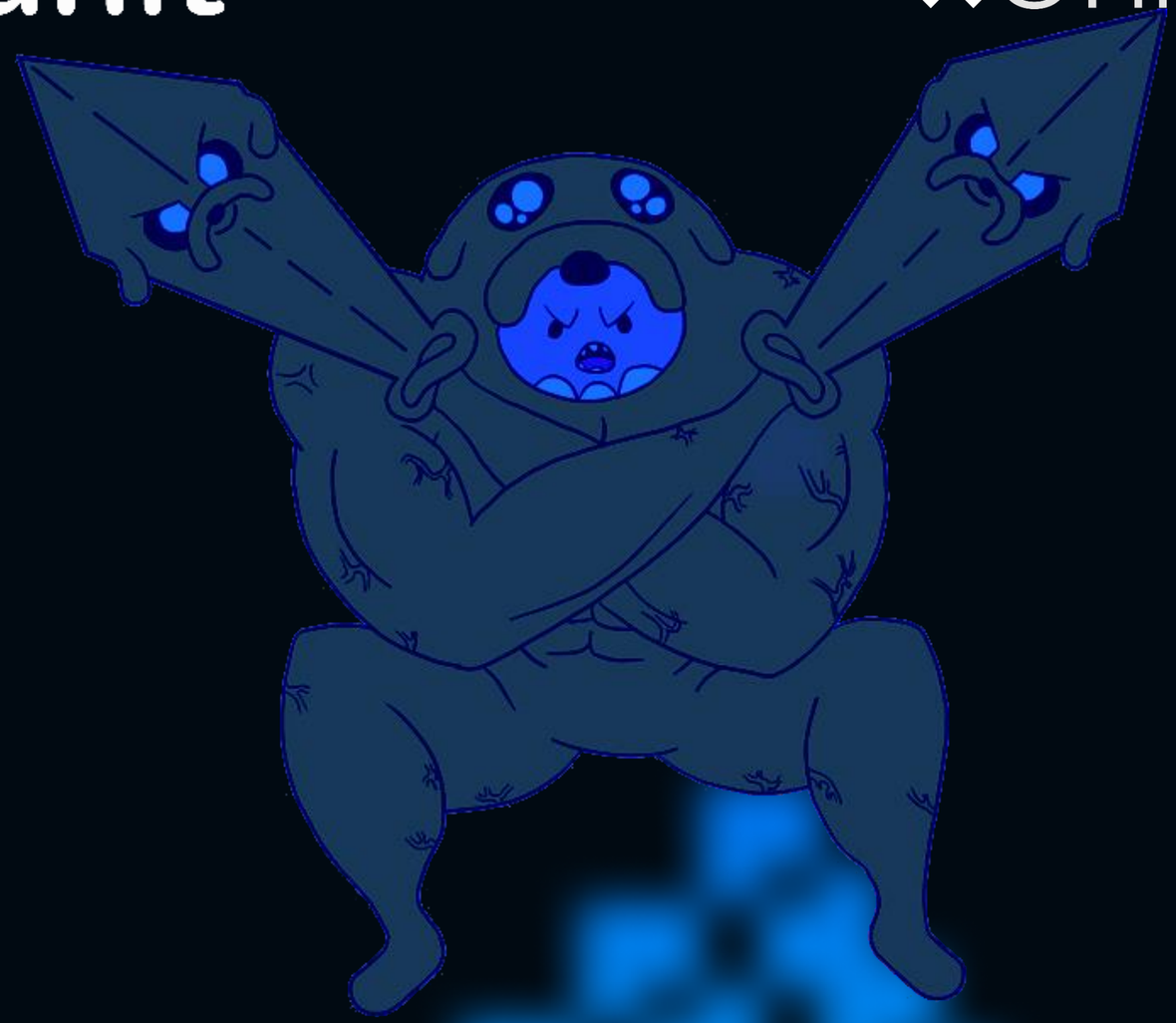
- ✓ NUnit.Framework.ITestAction — элемент АОП в Nunit
- ✓ В рамках одного тестового метода каждая реализация ITestAction применяется к каждой реализации IImplyFixture

# С NUnit - ВСЁ

# Не NUnit единым

nunit

xUnit.net



# Лезем в Google

Google xunit generic test

Все Картинки Покупки Видео Новости Ещё Инструменты

Результатов: примерно 104 000 (0,38 сек.)

https://stackoverflow.com › xu... Перевести эту страницу

## xUnit theory test using generics - Stack Overflow

19 сент. 2016 г. — In **xUnit** I can have a Theory **test** that uses **generics** in this form: [Theory] [MemberData(SomeScenario)] public void TestMethod<T>(T myType) ...

1 ответ · Лучший ответ: You can simply include Type as an input parameter instead. E.g.: [T…

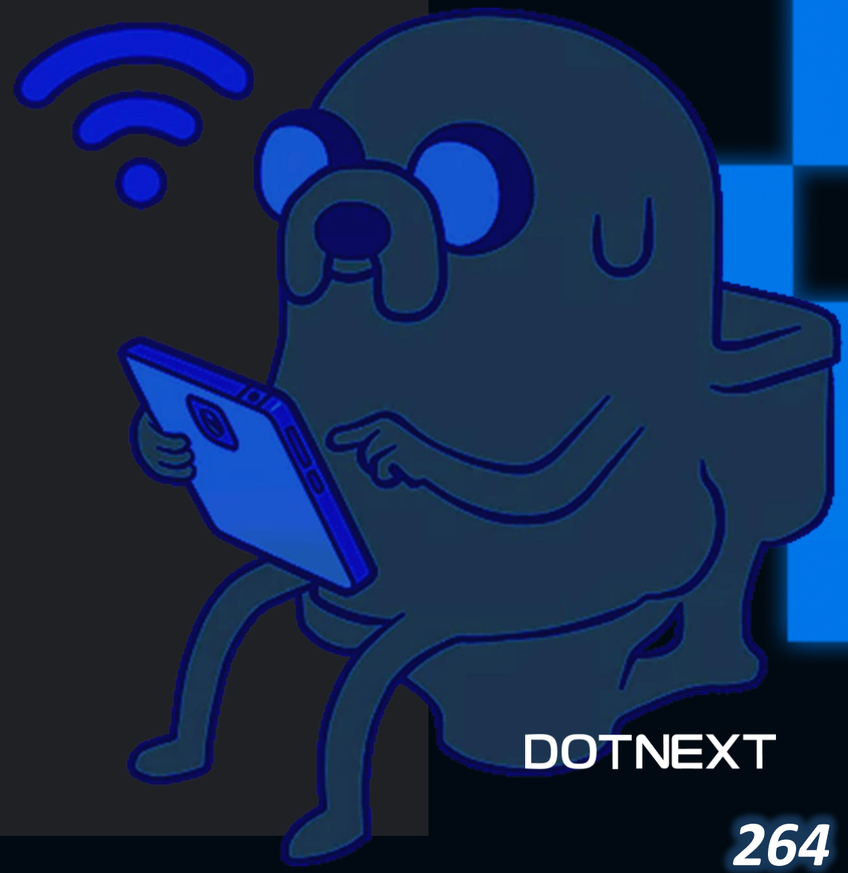
**Test** class simplification without **generics** - Stack Overflow 24 дек. 2017 г.

Unit **testing** a **generic** method - Stack Overflow 10 сент. 2013 г.

**xUnit generic** theories generated by introspection 14 апр. 2021 г.

Create an Unit **Test** for **Generic** Method - Stack Overflow 24 июн. 2016 г.

Другие результаты с сайта stackoverflow.com



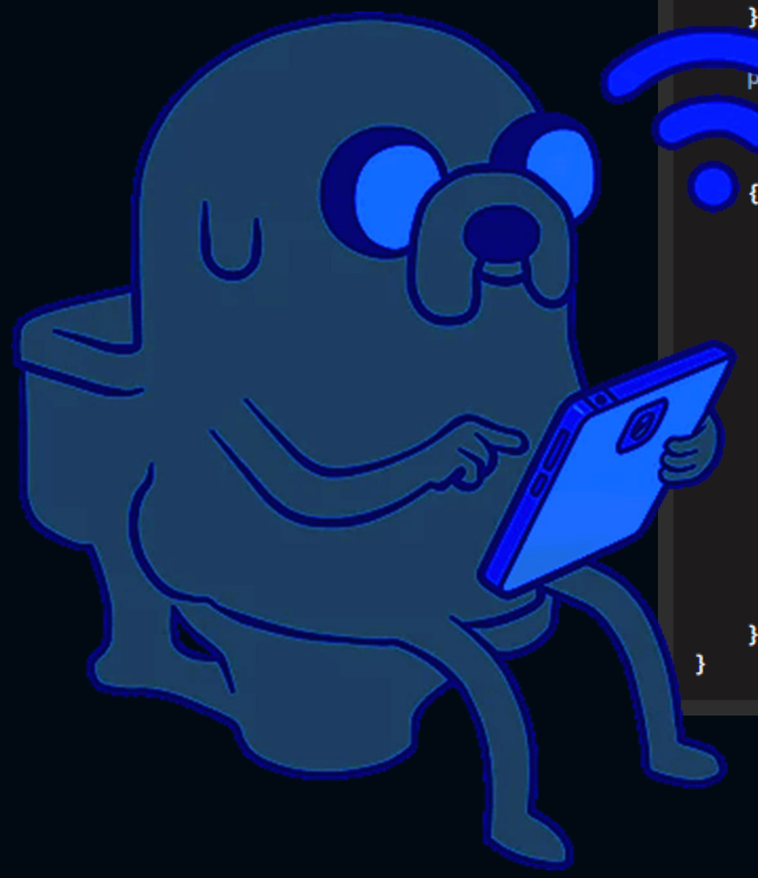
# Лезем в Google

**Edit (if you really need generics)**

1) You need to subclass `ITestMethod` to persist generic method info, it also has to implement `IXunitSerializable`

```
// assuming namespace Contosco
public class GenericTestMethod : MarshalByRefObject, ITestMethod, IXunitSerializable
{
    public IMethodInfo Method { get; set; }
    public ITestClass TestClass { get; set; }
    public ITypeInfo GenericArgument { get; set; }

    /// <summary />
    [Obsolete("Called by the de-serializer; should only be called by deriving classes for de-serialization purposes")]
    public GenericTestMethod()
    {
    }

    public GenericTestMethod(ITestClass @class, IMethodInfo method, ITypeInfo genericArgument)
    {
        this.Method = method;
        this.TestClass = @class;
        this.GenericArgument = genericArgument;
    }

    public void Serialize(IXunitSerializationInfo info)
    {
        info.AddValue("MethodName", (object) this.Method.Name, (Type) null);
        info.AddValue("TestClass", (object) this.TestClass, (Type) null);
        info.AddValue("GenericArgumentAssemblyName", GenericArgument.Assembly.Name);
        info.AddValue("GenericArgumentTypeName", GenericArgument.Name);
    }

    public void Deserialize(IXunitSerializationInfo info)
    {
        this.TestClass = info.GetValue<ITestClass>("TestClass");
        string assemblyName = info.GetValue<string>("GenericArgumentAssemblyName");
        string typeName = info.GetValue<string>("GenericArgumentTypeName");
        this.GenericArgument = Reflector.Wrap(GetType(assemblyName, typeName));
        this.Method = this.TestClass.Class.GetMethod(info.GetValue<string>("MethodName"), true).MakeGenericMethod(GenericArgument);
    }
}
```

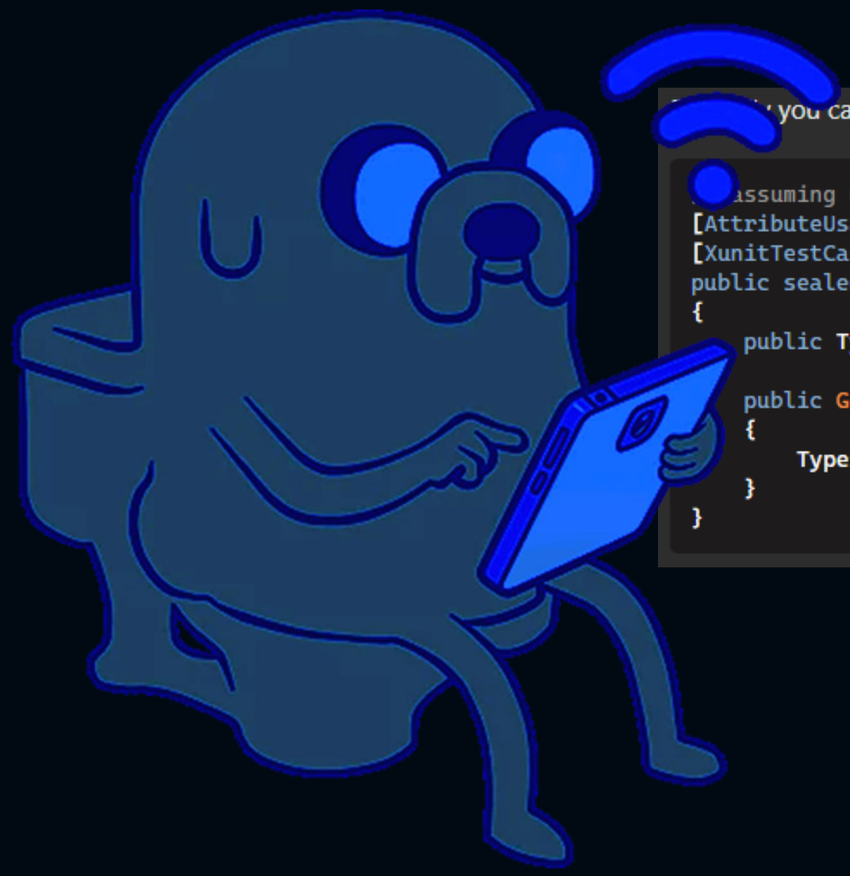
# Лезем в Google



2) You need to write your own discoverer for generic methods, it has to be subclass of `IXunitTestCaseDiscoverer`

```
// assuming namespace Contosco
public class GenericMethodDiscoverer : IXunitTestCaseDiscoverer
{
    public GenericMethodDiscoverer(IMessageSink diagnosticMessageSink)
    {
        DiagnosticMessageSink = diagnosticMessageSink;
    }

    pr    ed IMessageSink DiagnosticMessageSink { get; }

    pu lic IEnumerable<IXunitTestCase> Discover(ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod, IAttributeInfo factAttribute)
    {
        var result = new List<IXunitTestCase>();
        var types = factAttribute.GetNamedArgument<Type[]>("Types");
        foreach (var type in types)
        {
            var typeInfo = new ReflectionTypeInfo(type);
            var genericMethodInfo = testMethod.Method.MakeGenericMethod(typeInfo);
            var genericTestMethod = new GenericTestMethod(testMethod.TestClass, genericMethodInfo, typeInfo);

            result.Add(
                new XunitTestCase(DiagnosticMessageSink, discoveryOptions.MethodDisplayOrDefault(),
                    genericTestMethod));
        }

        return result;
    }
}
```

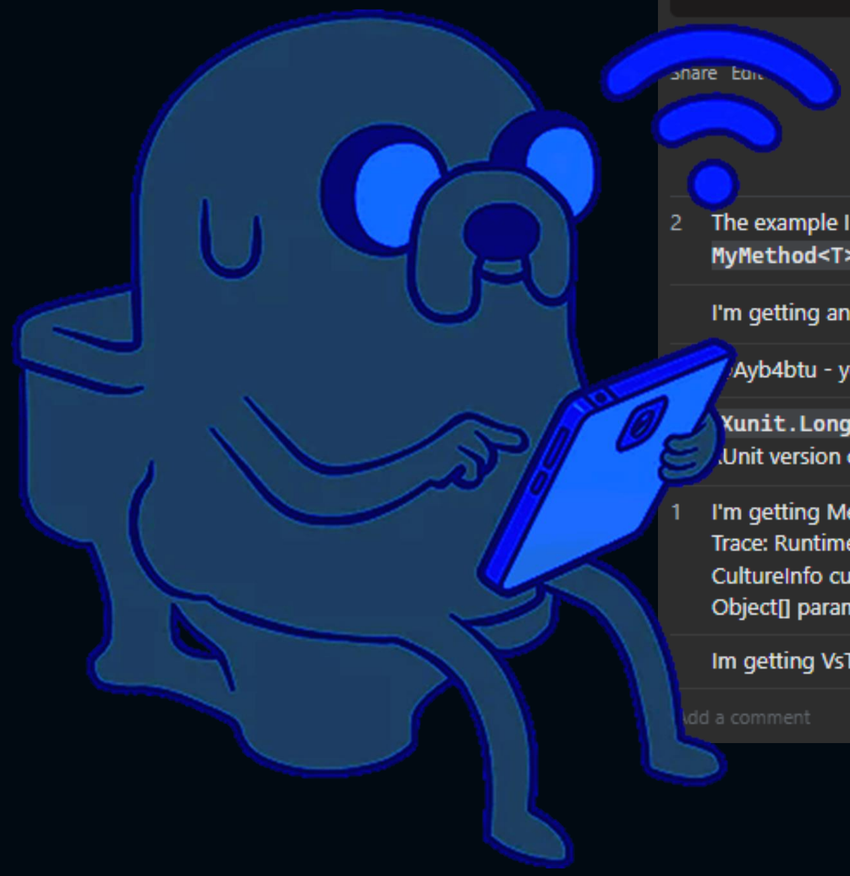
# Лезем в Google



you can make your attribute for generic methods and hook it to your custom discoverer by `XunitTestCaseDiscoverer` attribute

```
// assuming namespace Contosco
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
[XunitTestCaseDiscoverer("Contosco.GenericMethodDiscoverer", "Contosco")]
public sealed class GenericMethodAttribute : FactAttribute
{
    public Type[] Types { get; private set; }

    public GenericMethodAttribute(Type[] types)
    {
        Types = types;
    }
}
```

# Лезем в Google



Usage:

```
[GenericMethod(new Type[] { typeof(double), typeof(int) })]
public void TestGeneric<T>()
{
  Assert.Equal(typeof(T), typeof(double));
}
```

Share Edit

edited Jul 20, 2017 at 12:06

answered Jul 18, 2017 at 22:35

Ondrej Svejdar
20.7k ● 5 ● 54 ● 88

2 The example I gave in the question was a simplistic case. What if I wanted to test a method with generics that only takes in a generic parameter, e.g.: `public int MyMethod<T>(string someInput)` ? I could call this method passing in a `Type` using reflection, but this gets messy. – Ayb4btu Jul 19, 2017 at 1:07

I'm getting an assembly reference error on `Xunit.Sdk.LongLivedMarshalByRefObject` . Am I missing something? – Ayb4btu Jul 20, 2017 at 9:41

@Ayb4btu - you can substitue for MarshalByRefObject instead – Ondrej Svejdar Jul 20, 2017 at 10:07

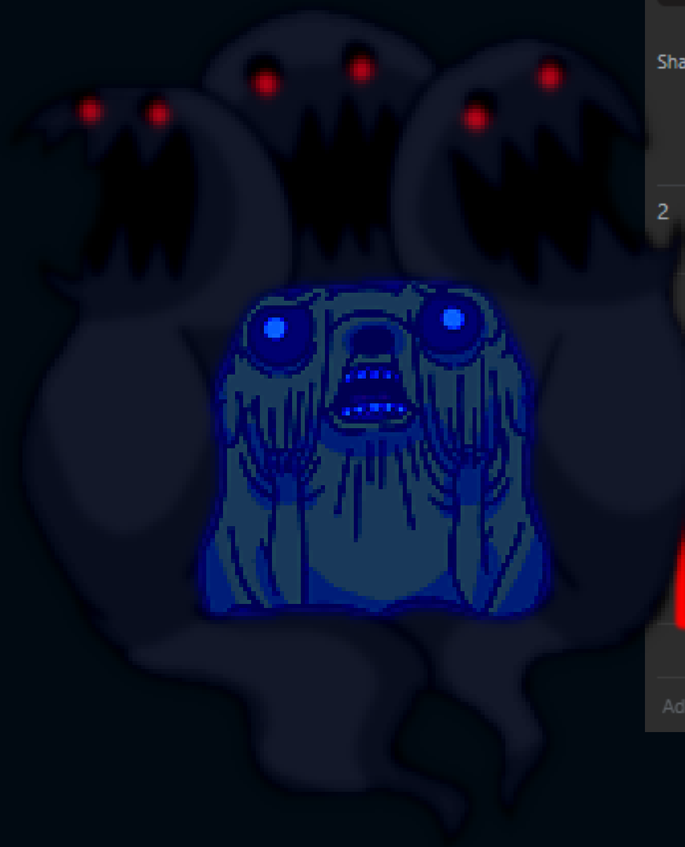
`Xunit.LongLivedMarshalByRefObject` is available, but visual studio Test Explorer can't seem to find the test. Everything has Contosco as the namespace. Could it be an xUnit version or dotnet core (which I'm using) issue? – Ayb4btu Jul 23, 2017 at 2:31

1 I'm getting Message: System.InvalidOperationException : Late bound operations cannot be performed on types or methods for which ContainsGenericParameters is true. Stack Trace: RuntimeMethodInfo.ThrowNoInvokeException() RuntimeMethodInfo.InvokeArgumentsCheck(Object obj, BindingFlags invokeAttr, Binder binder, Object[] parameters, CultureInfo culture) RuntimeMethodInfo.Invoke(Object obj, BindingFlags invokeAttr, Binder binder, Object[] parameters, CultureInfo culture) MethodBase.Invoke(Object obj, Object[] parameters) – Antao Almada Oct 16, 2019 at 17:12

Im getting VsTest test-case is missing. Rebuild the project and try again. Rebuild don't help ofc – Paweł Kanarek Jun 4, 2020 at 22:44

Add a comment

# Лезем в Google



Usage:

```
[GenericMethod(new Type[] { typeof(double), typeof(int) })]
public void TestGeneric<T>()
{
  Assert.Equal(typeof(T), typeof(double));
}
```

Share Edit Follow

edited Jul 20, 2017 at 12:06

answered Jul 18, 2017 at 22:35

Ondrej Svejdar
20.7k ● 5 ● 54 ● 88

2 The example I gave in the question was a simplistic case. What if I wanted to test a method with generics that only takes in a generic parameter, e.g.: `public int MyMethod<T>(string someInput)` ? I could call this method passing in a `Type` using reflection, but this gets messy. – Ayb4btu Jul 19, 2017 at 1:07

I'm getting an assembly reference error on `Xunit.Sdk.LongLivedMarshalByRefObject` . Am I missing something? – Ayb4btu Jul 20, 2017 at 9:41

@Ayb4btu - you can substitue for MarshalByRefObject instead – Ondrej Svejdar Jul 20, 2017 at 10:07

`Xunit.LongLivedMarshalByRefObject` is available, but visual studio Test Explorer can't seem to find the test. Everything has Contosco as the namespace. Could it be an xUnit version or dotnet core (which I'm using) issue? – Ayb4btu Jul 23, 2017 at 2:31

I'm getting Message: System.InvalidOperationException : Late bound operations cannot be performed on types or methods for which ContainsGenericParameters is true. Stack Trace: RuntimeMethodInfo.ThrowNoInvokeException() RuntimeMethodInfo.InvokeArgumentsCheck(Object obj, BindingFlags invokeAttr, Binder binder, Object[] parameters, CultureInfo culture) RuntimeMethodInfo.Invoke(Object obj, BindingFlags invokeAttr, Binder binder, Object[] parameters, CultureInfo culture) MethodBase.Invoke(Object obj, Object[] parameters) – Antao Almada Oct 16, 2019 at 17:12

Im getting VsTest test-case is missing. Rebuild the project and try again. Rebuild don't help ofc – Paweł Kanarek Jun 4, 2020 at 22:44

Add a comment

# Подготовка

```
public interface IGeneric
{
    Type[] TypeArguments { get; }
}
```

```
public static class AttributeInfoExtensions
{
    public static Type[] GetTypeArguments(this IAttributeInfo attribute) =>
        attribute.GetNamedArgument<Type[]>(nameof(IGeneric.TypeArguments));
}
```

# Подготовка

```
public interface IGeneric
{
    Type[] TypeArguments { get; }
}
```

```
public static class AttributeInfoExtensions
{
    public static Type[] GetTypeArguments(this IAttributeInfo attribute) =>
        attribute.GetNamedArgument<Type[]>(nameof(IGeneric.TypeArguments));
}
```

# Подготовка

```
public interface IGeneric
{
    Type[] TypeArguments { get; }
}
```

```
public static class AttributeInfoExtensions
{
    public static Type[] GetTypeArguments(this IAttributeInfo attribute) =>
        attribute.GetNamedArgument<Type[]>(nameof(IGeneric.TypeArguments));
}
```

# GDataAttribute

```
public class GDataDiscoverer : InlineDataDiscoverer
{
    public override IEnumerable<object[]> GetData(
        IAttributeInfo dataAttribute, IMethodInfo testMethod)
    {
        var typeArguments = dataAttribute.GetTypeArguments();

        foreach (var data in base.GetData(dataAttribute, testMethod))
            yield return data.Prepend(typeArguments).ToArray();
    };
}
```

# GDataAttribute

```
public class GDataDiscoverer : InlineDataDiscoverer
{
    public override IEnumerable<object[]> GetData(
        IAttributeInfo dataAttribute, IMethodInfo testMethod)
    {
        var typeArguments = dataAttribute.GetTypeArguments();

        foreach (var data in base.GetData(dataAttribute, testMethod))
            yield return data.Prepend(typeArguments).ToArray();
    };
}
```

# GDataAttribute

```
public class GDataDiscoverer : InlineDataDiscoverer
{
    public override IEnumerable<object[]> GetData(
        IAttributeInfo dataAttribute, IMethodInfo testMethod)
    {
        var typeArguments = dataAttribute.GetTypeArguments();

        foreach (var data in base.GetData(dataAttribute, testMethod))
            yield return data.Prepend(typeArguments).ToArray();
    };
}
```

# GDataAttribute

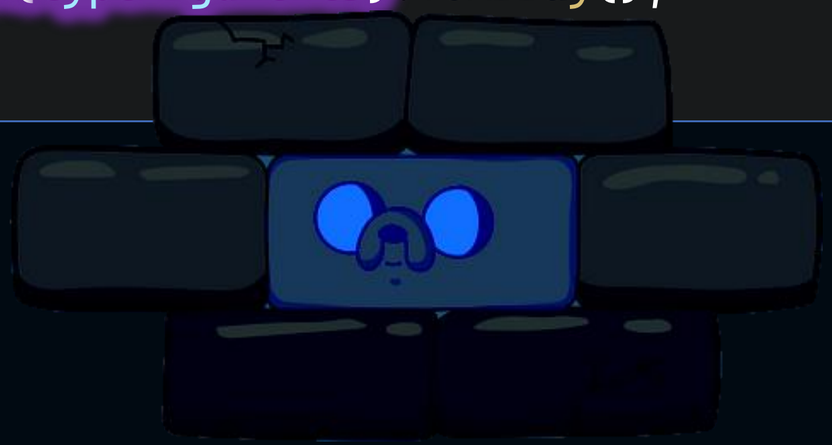
```
public class GDataDiscoverer : InlineDataDiscoverer
{
    public override IEnumerable<object[]> GetData(
        IAttributeInfo dataAttribute, IMethodInfo testMethod)
    {
        var typeArguments = dataAttribute.GetTypeArguments();

        foreach (var data in base.GetData(dataAttribute, testMethod))
            yield return data.Prepend(typeArguments).ToArray();
    };
}
```

# GDataAttribute

```
public class GDataDiscoverer : InlineDataDiscoverer
{
    public override IEnumerable<object[]> GetData(
        IAttributeInfo dataAttribute, IMethodInfo testMethod)
    {
        var typeArguments = dataAttribute.GetTypeArguments();

        foreach (var data in base.GetData(dataAttribute, testMethod))
            yield return data.Prepend(typeArguments).ToArray();
    };
}
```

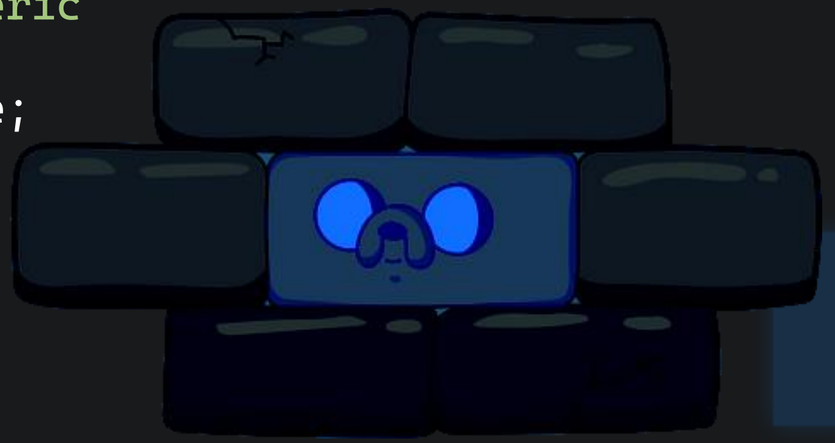
# GDataAttribute

```
public class GDataDiscoverer : InlineDataDiscoverer
{
    public override IEnumerable<object[]> GetData(
        IAttributeInfo dataAttribute, IMethodInfo testMethod)
    {
        var typeArguments = dataAttribute.GetTypeArguments();

        foreach (var data in base.GetData(dataAttribute, testMethod))
            yield return data.Prepend(typeArguments).ToArray();
    };
}
```



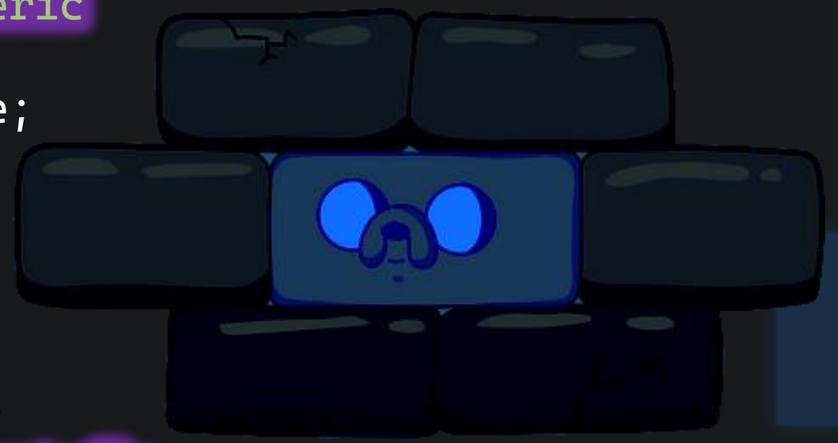
# GDataAttribute

```
[DataDiscoverer(
        $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(GDataDiscoverer)}",
        $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = true)]
public class GDataAttribute : DataAttribute, IGeneric
{
    private readonly InlineDataAttribute attribute;

    public Type[] TypeArguments { get; }

    public GDataAttribute(params object[] data)
    {
        attribute = new InlineDataAttribute(data);
        TypeArguments = GetType().GetGenericArguments();
    }

    public override IEnumerable<object[]> GetData(MethodInfo testMethod) =>
        attribute.GetData(testMethod);
}
```



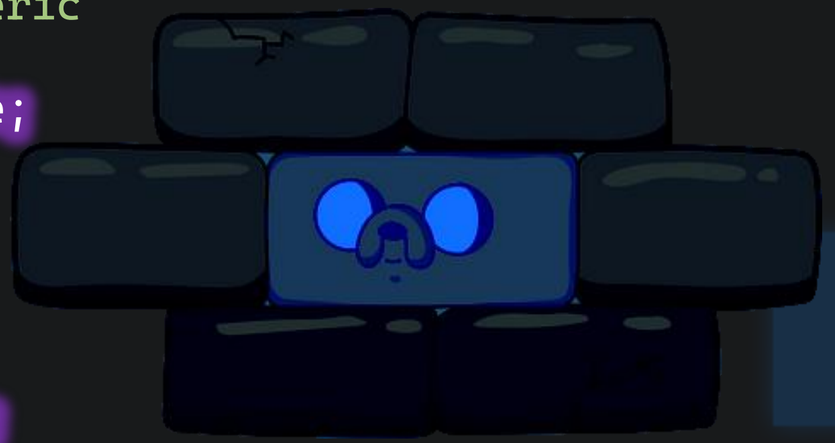
# GDataAttribute

```
[DataDiscoverer(
        $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(GDataDiscoverer)}",
        $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = true)]
public class GDataAttribute : DataAttribute, IGeneric
{
    private readonly InlineDataAttribute attribute;

    public Type[] TypeArguments { get; }

    public GDataAttribute(params object[] data)
    {
        attribute = new InlineDataAttribute(data);
        TypeArguments = GetType().GetGenericArguments();
    }

    public override IEnumerable<object[]> GetData(MethodInfo testMethod) =>
        attribute.GetData(testMethod);
}
```



# GDataAttribute

```
[DataDiscoverer(
        $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(GDataDiscoverer)}",
        $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = true)]
public class GDataAttribute : DataAttribute, IGeneric
{
    private readonly InlineDataAttribute attribute;

    public Type[] TypeArguments { get; }

    public GDataAttribute(params object[] data)
    {
        attribute = new InlineDataAttribute(data);
        TypeArguments = GetType().GetGenericArguments();
    }

    public override IEnumerable<object[]> GetData(MethodInfo testMethod) =>
        attribute.GetData(testMethod);
}
```



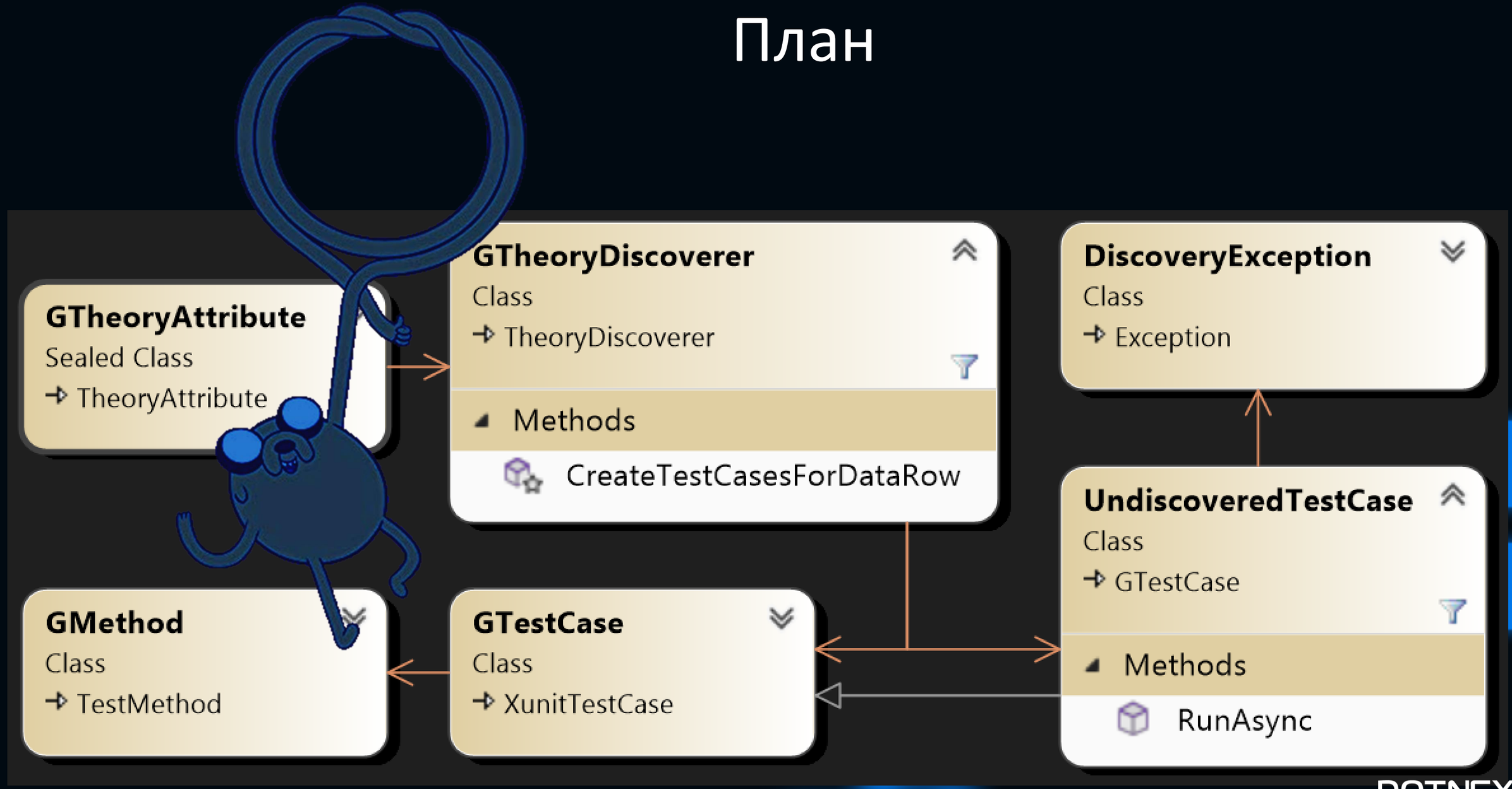
# GDataAttribute

```
[DataDiscoverer(
        $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(GDataDiscoverer)}",
        $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true, Inherited = true)]
public class GDataAttribute : DataAttribute, IGeneric
{
    private readonly InlineDataAttribute attribute;

    public Type[] TypeArguments { get; }

    public GDataAttribute(params object[] data)
    {
        attribute = new InlineDataAttribute(data);
        TypeArguments = GetType().GetGenericArguments();
    }

    public override IEnumerable<object[]> GetData(MethodInfo testMethod) =>
        attribute.GetData(testMethod);
}
```

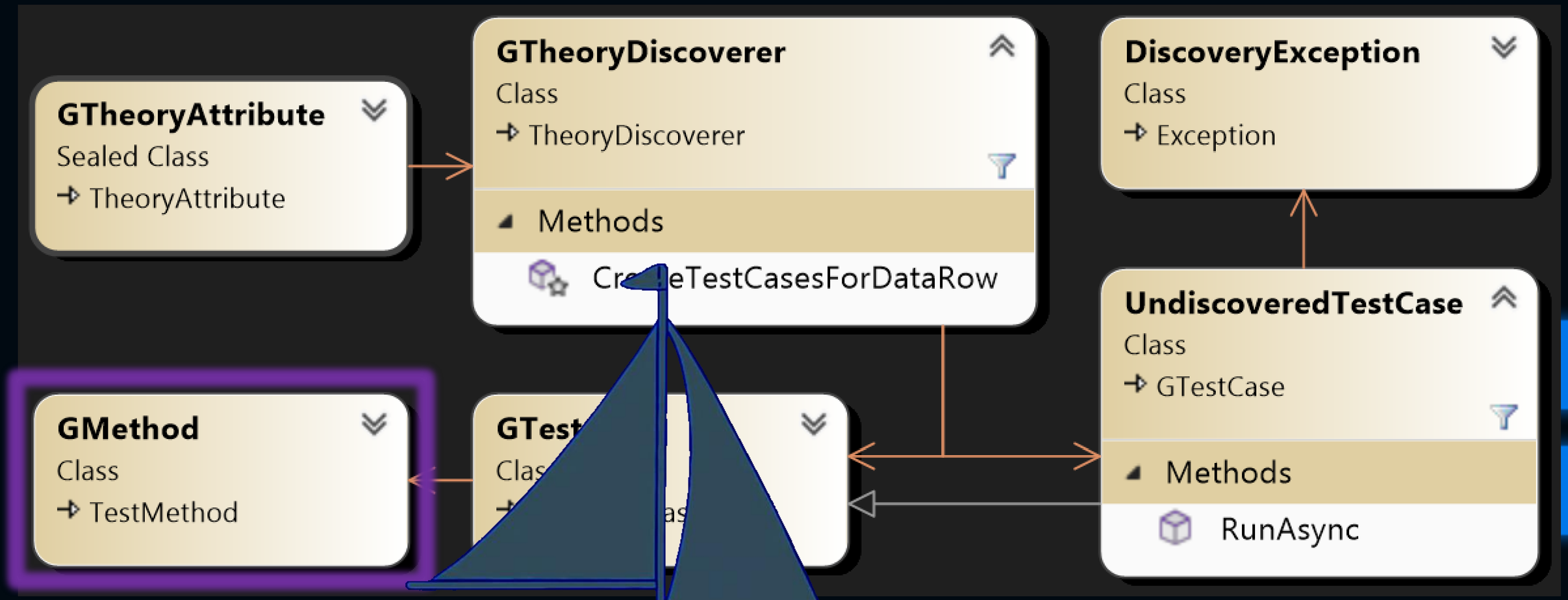
# GDataAttribute

```
public class GDataAttribute<T1, T2, T3> : GDataAttribute
{
    public GDataAttribute(params object[] data) : base(data) { }
}
```

# План


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync

# GMethod


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
DiscoveryException
Class
Exception
GMethod
Class
TestMethod
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync

# GMethod

```
public class GMethod : TestMethod, IGeneric
{
    private Type[] typeArguments;

    public Type[] TypeArguments
    {
        get => typeArguments;
        private set =>
            Method = Method.MakeGenericMethod((typeArguments = value)
                .Select(Reflector.Wrap).ToArray());
    }

    public GMethod(ITestMethod testMethod, Type[] typeArguments)
        : base(testMethod.TestClass, testMethod.Method) =>
        TypeArguments = typeArguments;
}
```

# GMethod

```
public class GMethod : TestMethod, IGeneric
{
    private Type[] typeArguments;

    public Type[] TypeArguments
    {
        get => typeArguments;
        private set =>
            Method = Method.MakeGenericMethod((typeArguments = value)
                .Select(Reflector.Wrap).ToArray());
    }

    public GMethod(ITestMethod testMethod, Type[] typeArguments)
        : base(testMethod.TestClass, testMethod.Method) =>
        TypeArguments = typeArguments;
}
```

# GMethod

```
public class GMethod : TestMethod, IGeneric
{
    private Type[] typeArguments;

    public Type[] TypeArguments
    {
        get => typeArguments;
        private set =>
            Method = Method.MakeGenericMethod((typeArguments = value)
                .Select(Reflector.Wrap).ToArray());
    }

    public GMethod(ITestMethod testMethod, Type[] typeArguments)
        : base(testMethod.TestClass, testMethod.Method) =>
        TypeArguments = typeArguments;
}
```

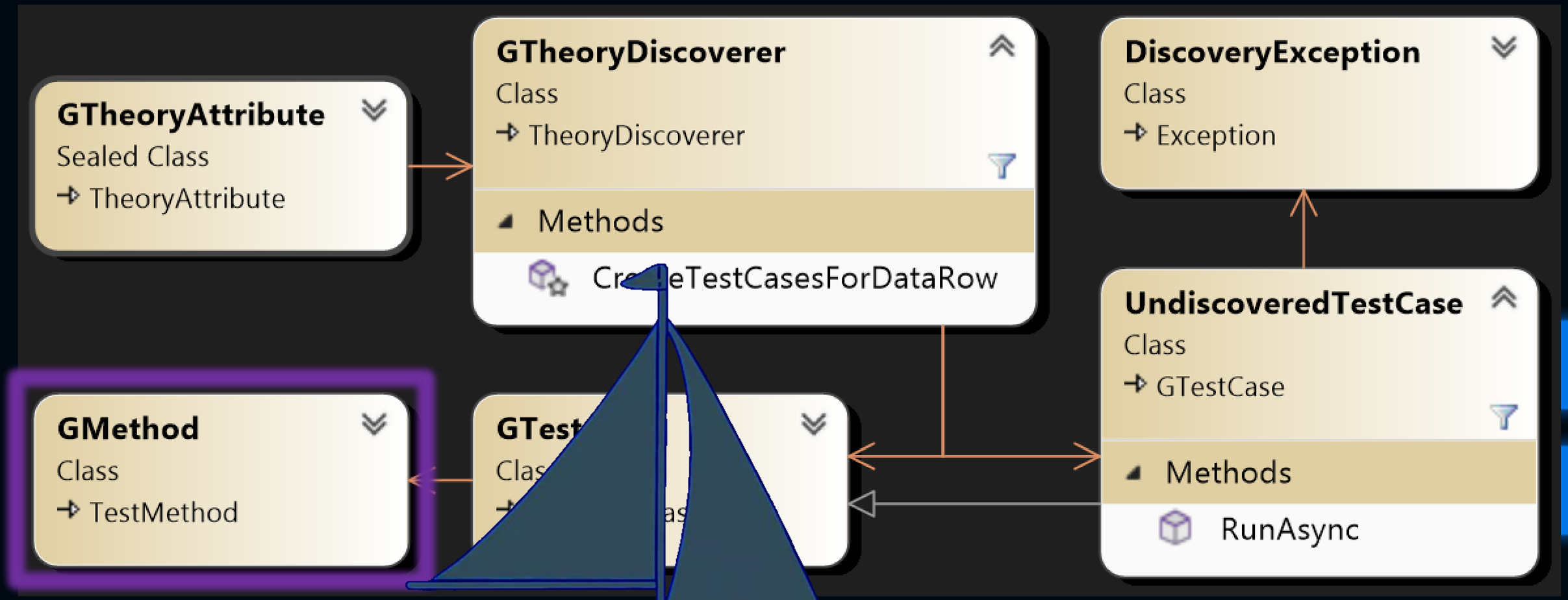
# GMethod

```
public class GMethod : TestMethod, IGeneric
{
    private Type[] typeArguments;

    public Type[] TypeArguments
    {
        get => typeArguments;
        private set =>
            Method = Method.MakeGenericMethod((typeArguments = value)
                .Select(Reflector.Wrap).ToArray());
    }

    public GMethod(ITestMethod testMethod, Type[] typeArguments)
        : base(testMethod.TestClass, testMethod.Method) =>
        TypeArguments = typeArguments;
}
```

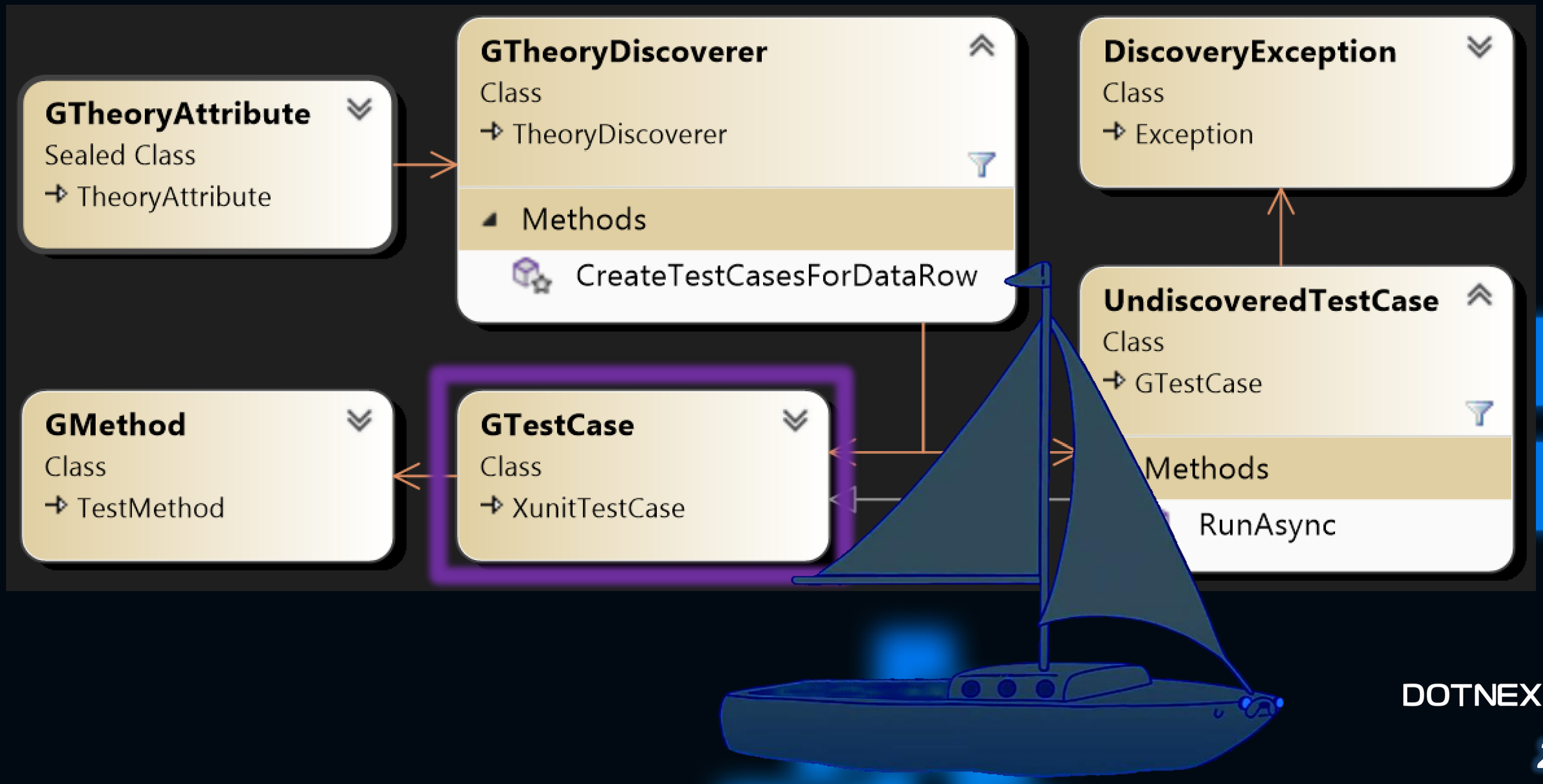
# GMethod

```
public class GMethod : TestMethod, IGeneric, IXunitSerializable
{
    // ...

    [Obsolete("Called by the de-serializer ...")]
    public GMethod() { }

    public new void Serialize(IXunitSerializationInfo info)
    {
        base.Serialize(info);
        info.AddValue(nameof(TypeArguments), TypeArguments);
    }

    public new void Deserialize(IXunitSerializationInfo info)
    {
        base.Deserialize(info);
        TypeArguments = info.GetValue<Type[]>(nameof(TypeArguments));
    }
}
```

# GMethod


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
GMethod
Class
TestMethod
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync

# GTestCase


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# GTestCase

```
public class GTestCase : XunitTestCase, IGeneric
{
    public Type[] TypeArguments { get; private set; }

    public GTestCase(IMessageSink diagnosticMessageSink,
                     TestMethodDisplay defaultMethodDisplay,
                     TestMethodDisplayOptions defaultMethodDisplayOptions,
                     ITestMethod testMethod,
                     Type[] typeArguments,
                     object[] testMethodArguments = null)
        : base(diagnosticMessageSink, defaultMethodDisplay,
               defaultMethodDisplayOptions, testMethod, testMethodArguments) =>
        TypeArguments = typeArguments;

    // ...
}
```

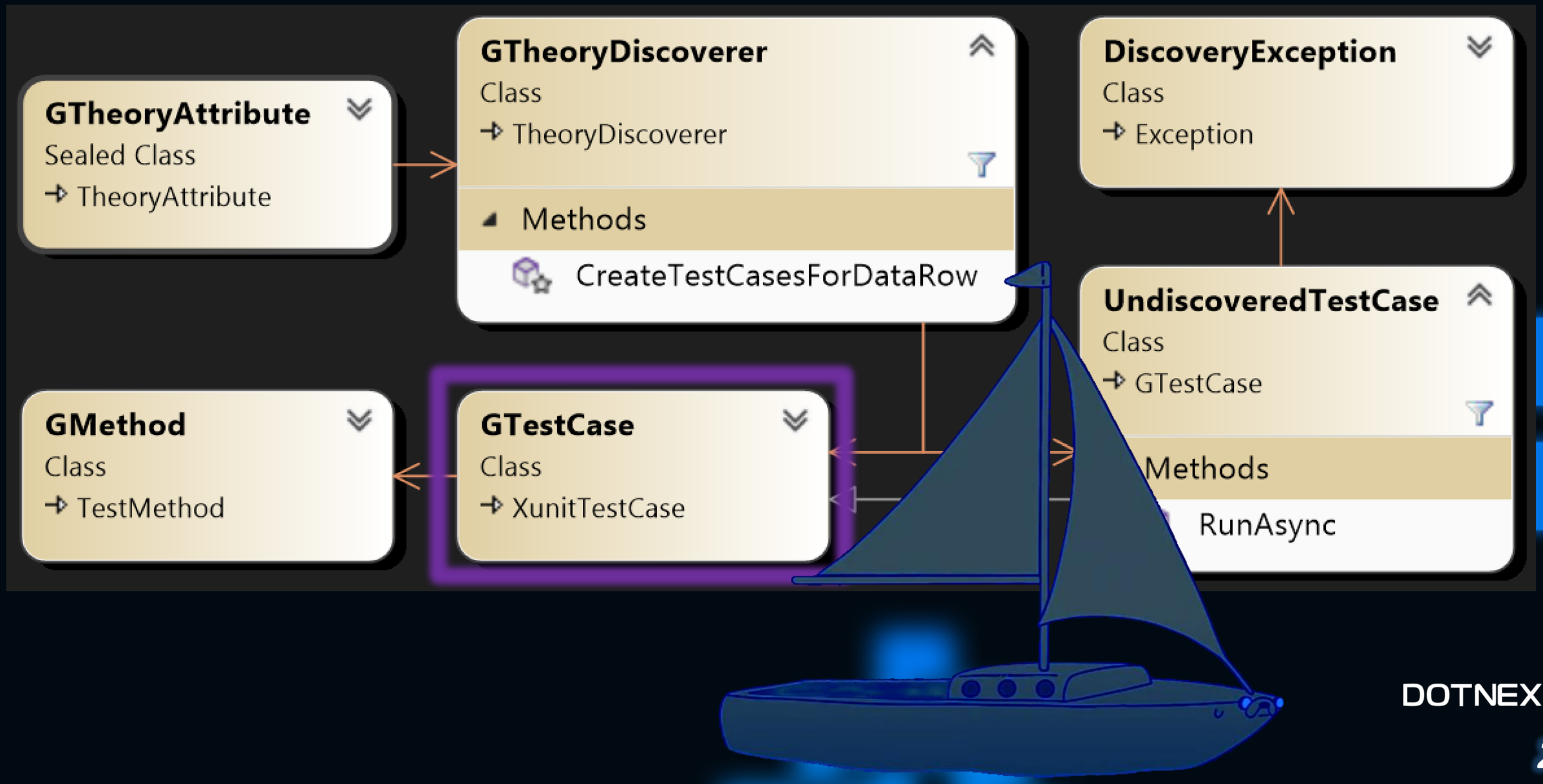
# GTestCase

```
public class GTestCase : XunitTestCase, IGeneric
{
    public Type[] TypeArguments { get; private set; }

    public GTestCase(IMessageSink diagnosticMessageSink,
                     TestMethodDisplay defaultMethodDisplay,
                     TestMethodDisplayOptions defaultMethodDisplayOptions,
                     ITestMethod testMethod,
                     Type[] typeArguments,
                     object[] testMethodArguments = null)
        : base(diagnosticMessageSink, defaultMethodDisplay,
               defaultMethodDisplayOptions, testMethod, testMethodArguments) =>
        TypeArguments = typeArguments;

    // ...
}
```

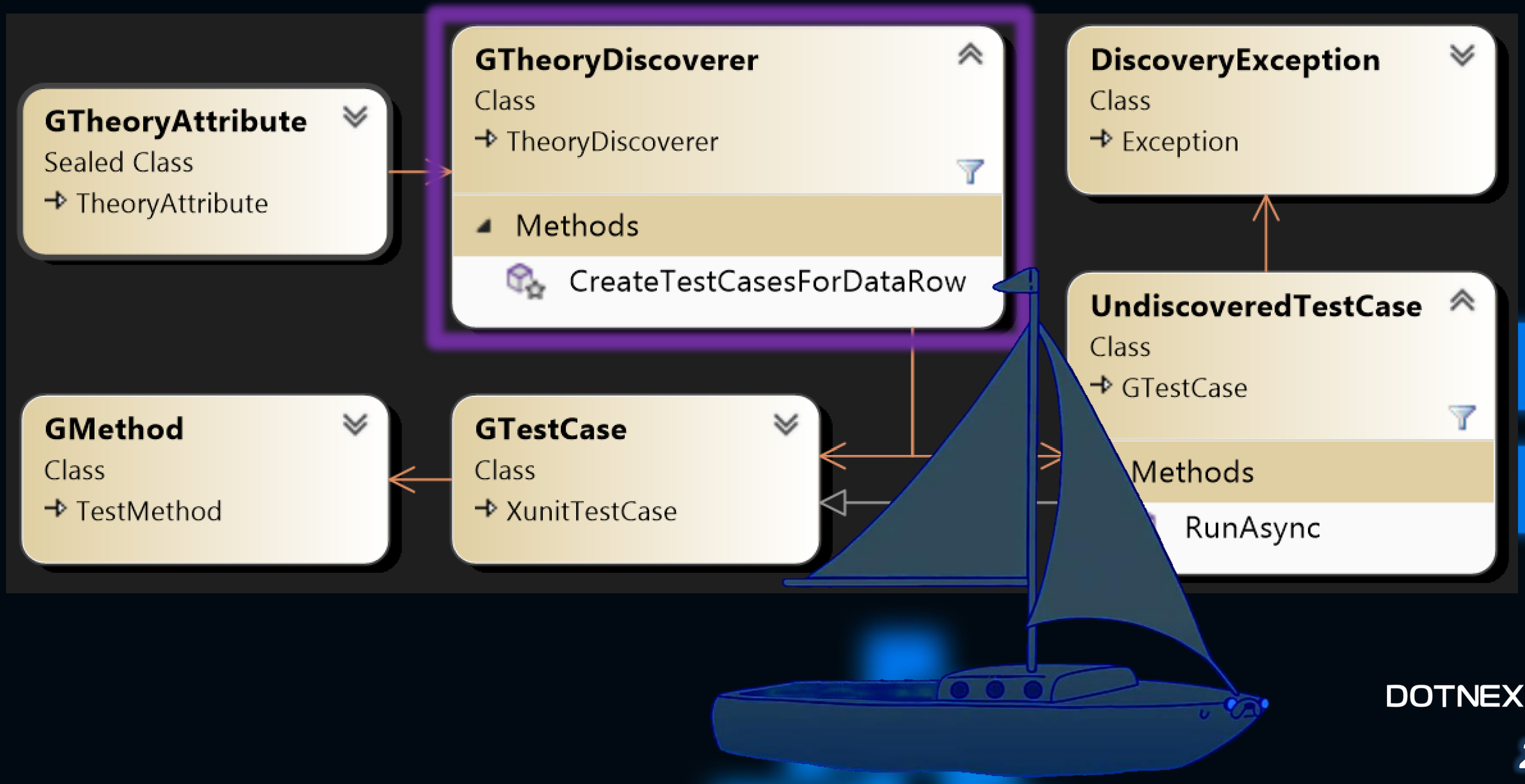
# GTestCase

```
public class GTestCase : XunitTestCase, IGeneric
{
    // ...

    [Obsolete("Called by the de-serializer ...")]
    public GTestCase() { }

    public override void Serialize(IXunitSerializationInfo data)
    {
        base.Serialize(data);
        data.AddValue(nameof(TypeArguments), TypeArguments);
    }

    public override void Deserialize(IXunitSerializationInfo data)
    {
        base.Deserialize(data);
        TypeArguments = data.GetValue<Type[]>(nameof(TypeArguments));
    }
}
```

# GTestCase


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# GTheoryDiscoverer



# GTheoryDiscoverer

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();
        var test = new GMethod(testMethod, typeArguments);
        var testCase = new GTestCase(
                DiagnosticMessageSink,
                discoveryOptions.MethodDisplayOrDefault(),
                discoveryOptions.MethodDisplayOptionsOrDefault(),
                test,
                typeArguments,
                testMethodArguments);

        return new[] { testCase };
    }
}
```

# GTheoryDiscoverer

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();
        var test = new GMethod(testMethod, typeArguments);
        var testCase = new GTestCase(
                DiagnosticMessageSink,
                discoveryOptions.MethodDisplayOrDefault(),
                discoveryOptions.MethodDisplayOptionsOrDefault(),
                test,
                typeArguments,
                testMethodArguments);

        return new[] { testCase };
    }
}
```

# GTheoryDiscoverer

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();
        var test = new GMethod(testMethod, typeArguments);
        var testCase = new GTestCase(
                DiagnosticMessageSink,
                discoveryOptions.MethodDisplayOrDefault(),
                discoveryOptions.MethodDisplayOptionsOrDefault(),
                test,
                typeArguments,
                testMethodArguments);

        return new[] { testCase };
    }
}
```

# GTheoryDiscoverer

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();
        var test = new GMethod(testMethod, typeArguments);
        var testCase = new GTestCase(
                DiagnosticMessageSink,
                discoveryOptions.MethodDisplayOrDefault(),
                discoveryOptions.MethodDisplayOptionsOrDefault(),
                test,
                typeArguments,
                testMethodArguments);

        return new[] { testCase };
    }
}
```

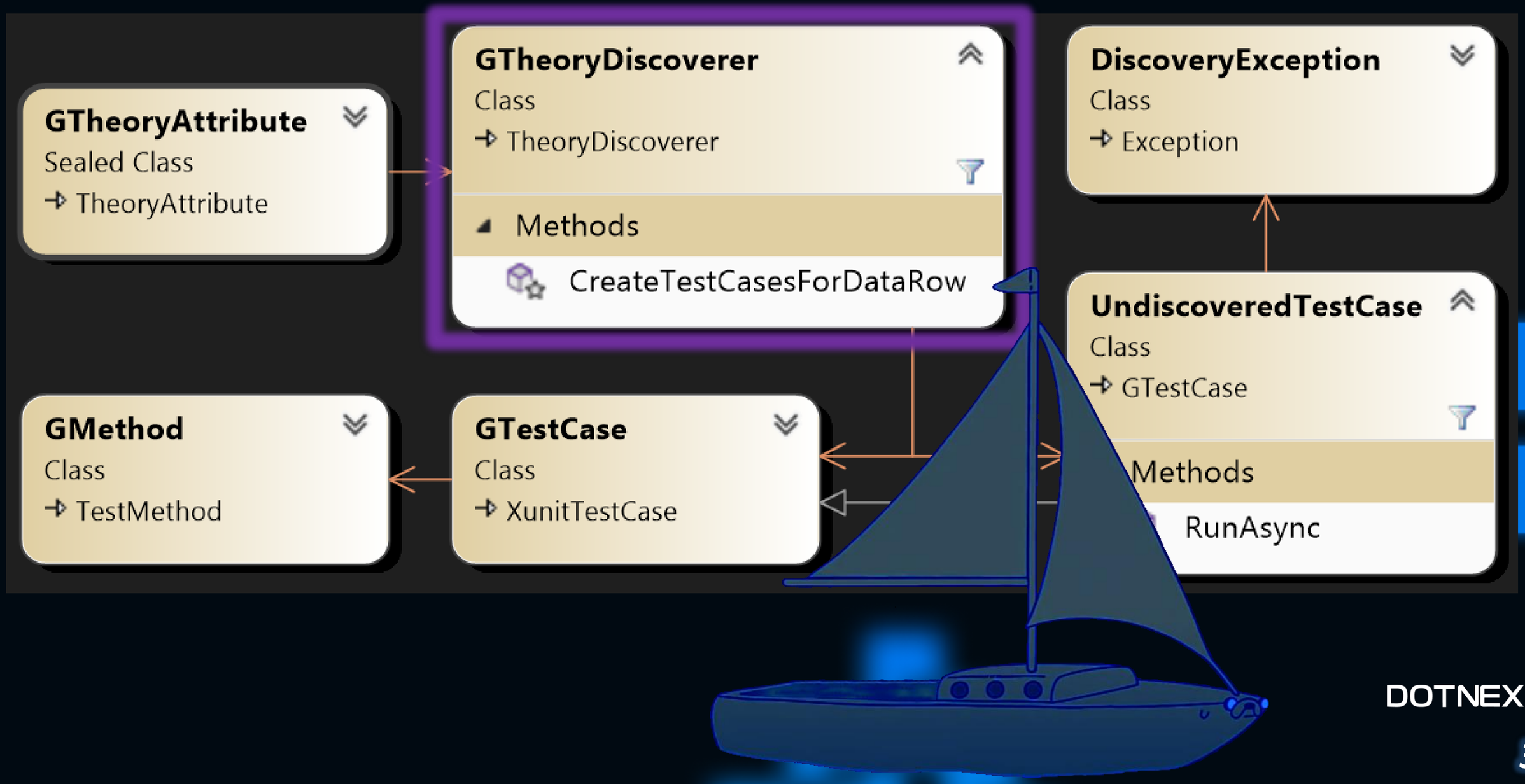
# GTheoryDiscoverer

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();
        var test = new GMethod(testMethod, typeArguments);
        var testCase = new GTestCase(
                DiagnosticMessageSink,
                discoveryOptions.MethodDisplayOrDefault(),
                discoveryOptions.MethodDisplayOptionsOrDefault(),
                test,
                typeArguments,
                testMethodArguments);

        return new[] { testCase };
    }
}
```

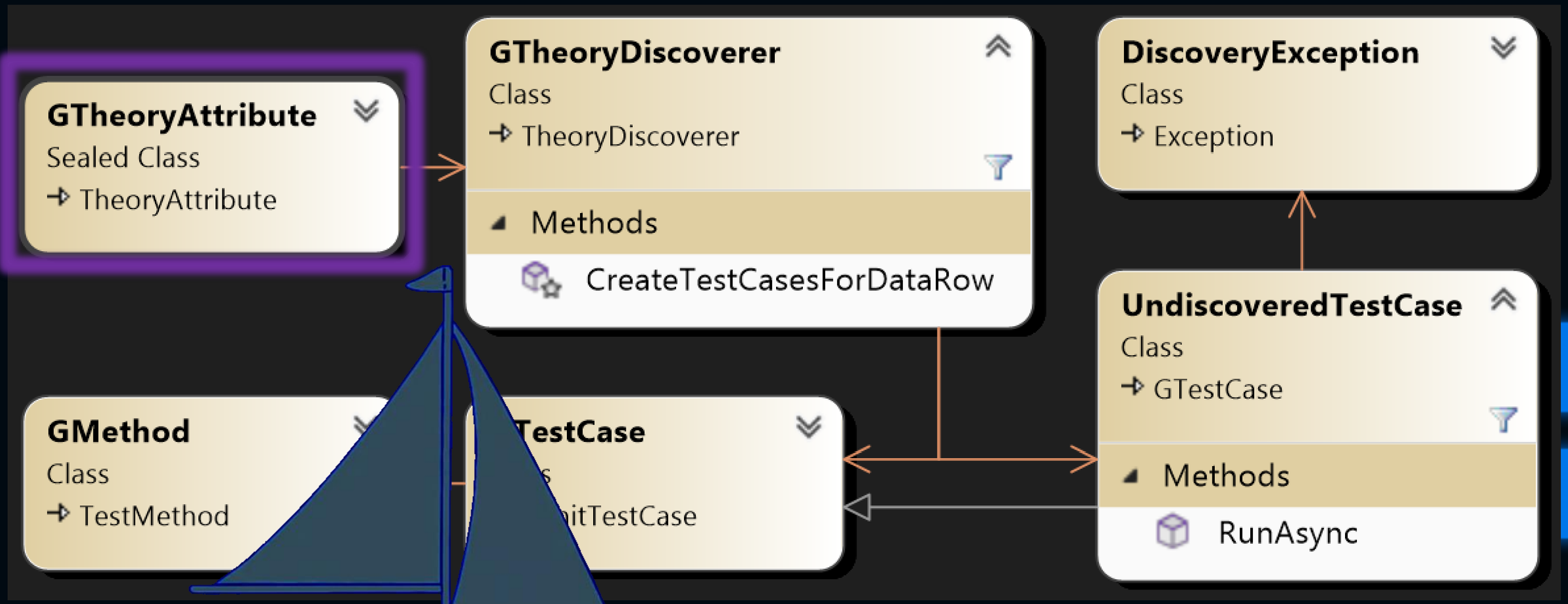
# GTheoryDiscoverer

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();
        var test = new GMethod(testMethod, typeArguments);
        var testCase = new GTestCase(
                DiagnosticMessageSink,
                discoveryOptions.MethodDisplayOrDefault(),
                discoveryOptions.MethodDisplayOptionsOrDefault(),
                test,
                typeArguments,
                testMethodArguments);

        return new[] { testCase };
    }
}
```

# GTheoryDiscoverer



# GTheory


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod

# GTheory

```
[XunitTestCaseDiscoverer(
    $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(GTheoryDiscoverer)}",
    $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
public sealed class GTheoryAttribute : TheoryAttribute { }
```

# GTheory

```
[XunitTestCaseDiscoverer(
    $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(GTheoryDiscoverer)}",
    $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
public sealed class GTheoryAttribute : TheoryAttribute { }
```

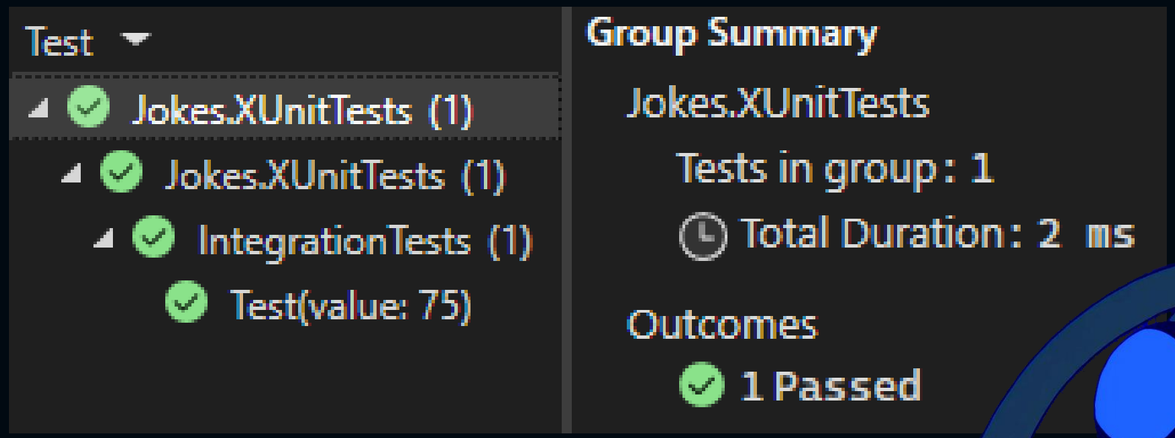
# Тесты

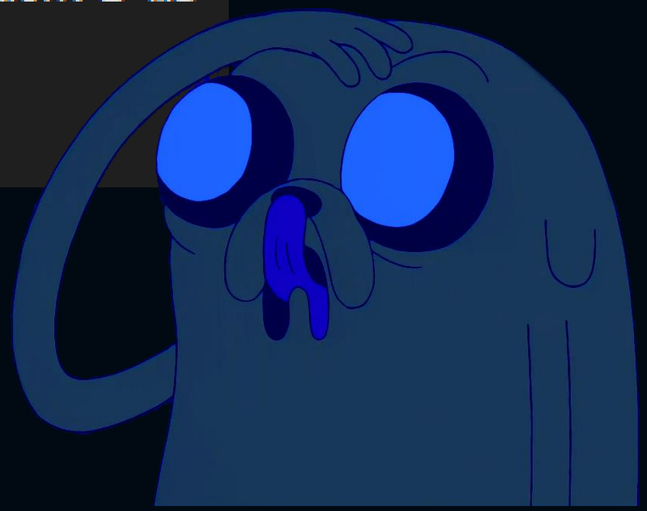
```
[GTheory]
[GData<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GData<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GData<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GData<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.True(actual);
}
```

# Просто сравните

```
[GCase<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<RussianJokes, ForParty, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
[GCase<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Имя, сестра!


Test
Jokes.XUnitTests (1)
Jokes.XUnitTests (1)
IntegrationTests (1)
Test(value: 75)
Group Summary
Jokes.XUnitTests
Tests in group: 1
Total Duration: 2 ms
Outcomes
1 Passed



# Имя, сестра!

```
public class GTestCase : XunitTestCase, IGeneric
{
    // ...

    protected override string GetDisplayName(IAttributeInfo factAttribute, string displayName) =>
        Method.GetDisplayNameWithArguments(Method.Name, TestMethodArguments ?? new object[0],
            Method.GetGenericArguments().ToArray());
}
```

# Имя, сестра!

Test

- Jokes.XUnitTests (1)
  - Jokes.XUnitTests (1)
    - IntegrationTests (1)
      - Test<GibraltarJokes, ForParty, AreOfMinimalFun>(value: 75)

**Group Summary**

Jokes.XUnitTests

Tests in group: 1

Total Duration: 3 ms

Outcomes

1 Passed

# Имя, сестра!

https://xunit.net/xunit.analyzers/rules/xUnit1025

xUnit.net

## xUnit1025 Warning

InlineData should be unique within the Theory it belongs to

### Cause

Test data provided with `InlineDataAttribute` is duplicated in other `InlineDataAttribute` occurence(s).

### Reason for rule

Having test data duplicated leads to duplication of test ID which may result in incorrect or unexpected behavior. This usually comes from:

- typos
- test data copying and not updating
- not taking into account default values or `params` defined parameters

### How to fix violations

Remove duplicated `InlineDataAttribute` occurrences.

# Имя, сестра!

https://xunit.net/xunit.analyzers/rules/xUnit1025

xUnit.net

## xUnit1025 Warning

InlineData should be unique within the Theory it belongs to

### Cause

Test data provided with `InlineDataAttribute` is duplicated in other `InlineDataAttribute` occurence(s).

### Reason for rule

Having test data duplicated leads to duplication of test ID which may result in incorrect or unexpected behavior. This usually comes from:

- typos
- test data copying and not updating
- not taking into account default values or `params` defined parameters

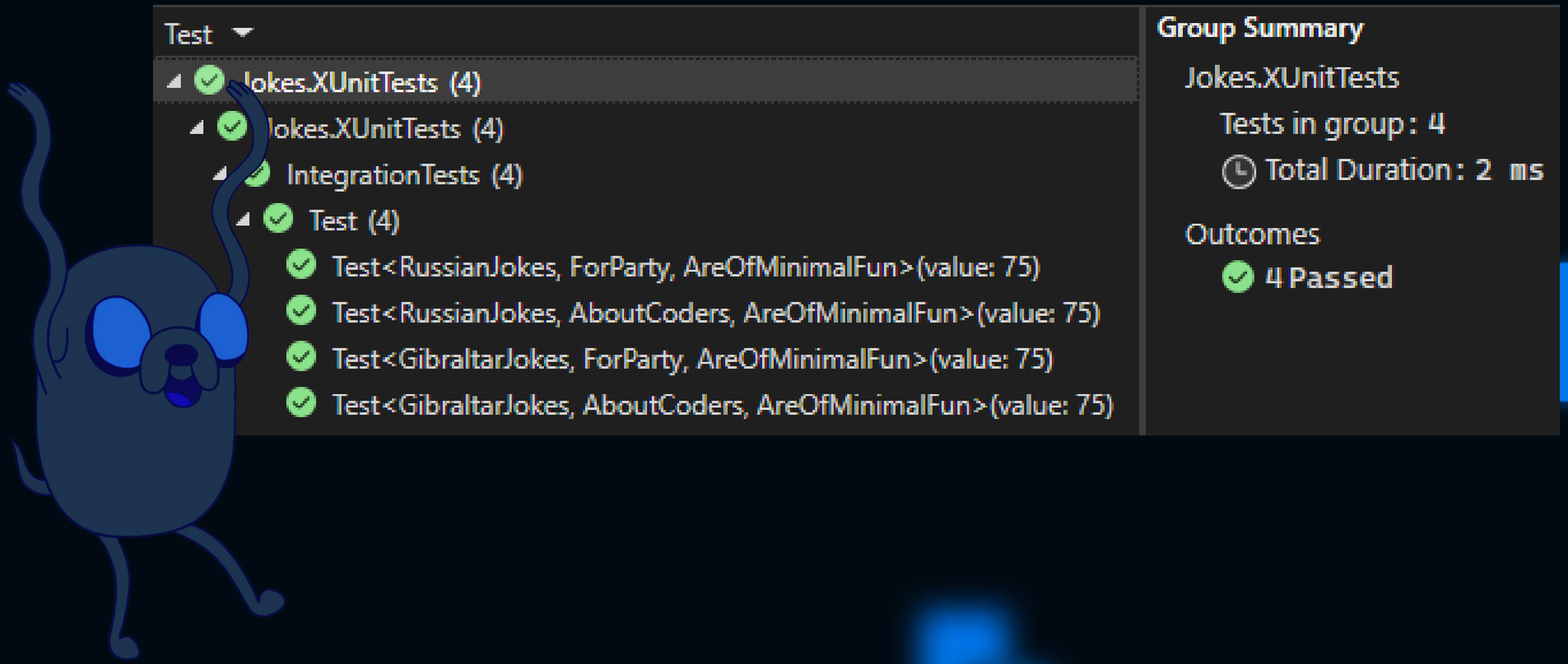
### How to fix violations

Remove duplicated `InlineDataAttribute` occurrences.

# Имя, сестра!

```
public class GTestCase : XunitTestCase, IGeneric
{
    // ...

    protected override string GetUniqueID() => $"{DisplayName}{{{base.GetUniqueID()}}}";
}
```

# Имя, сестра!


Test
lokes.XUnitTests (4)
lokes.XUnitTests (4)
IntegrationTests (4)
Test (4)
Test<RussianJokes, ForParty, AreOfMinimalFun>(value: 75)
Test<RussianJokes, AboutCoders, AreOfMinimalFun>(value: 75)
Test<GibraltarJokes, ForParty, AreOfMinimalFun>(value: 75)
Test<GibraltarJokes, AboutCoders, AreOfMinimalFun>(value: 75)
Group Summary
Jokes.XUnitTests
Tests in group : 4
Total Duration : 2 ms
Outcomes
4 Passed

# Плохой тест

```
// ...
[GData<object, object, object>(75)]
public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...
}
```

# Плохой тест

Test

- Jokes.XUnitTests (1)
  - Jokes.XUnitTests (1)
    - IntegrationTests (1)
      - Test

**Test Detail Summary**

Jokes.XUnitTests.IntegrationTests.Test

Source: IntegrationTests.cs line 18

Duration: 1 ms

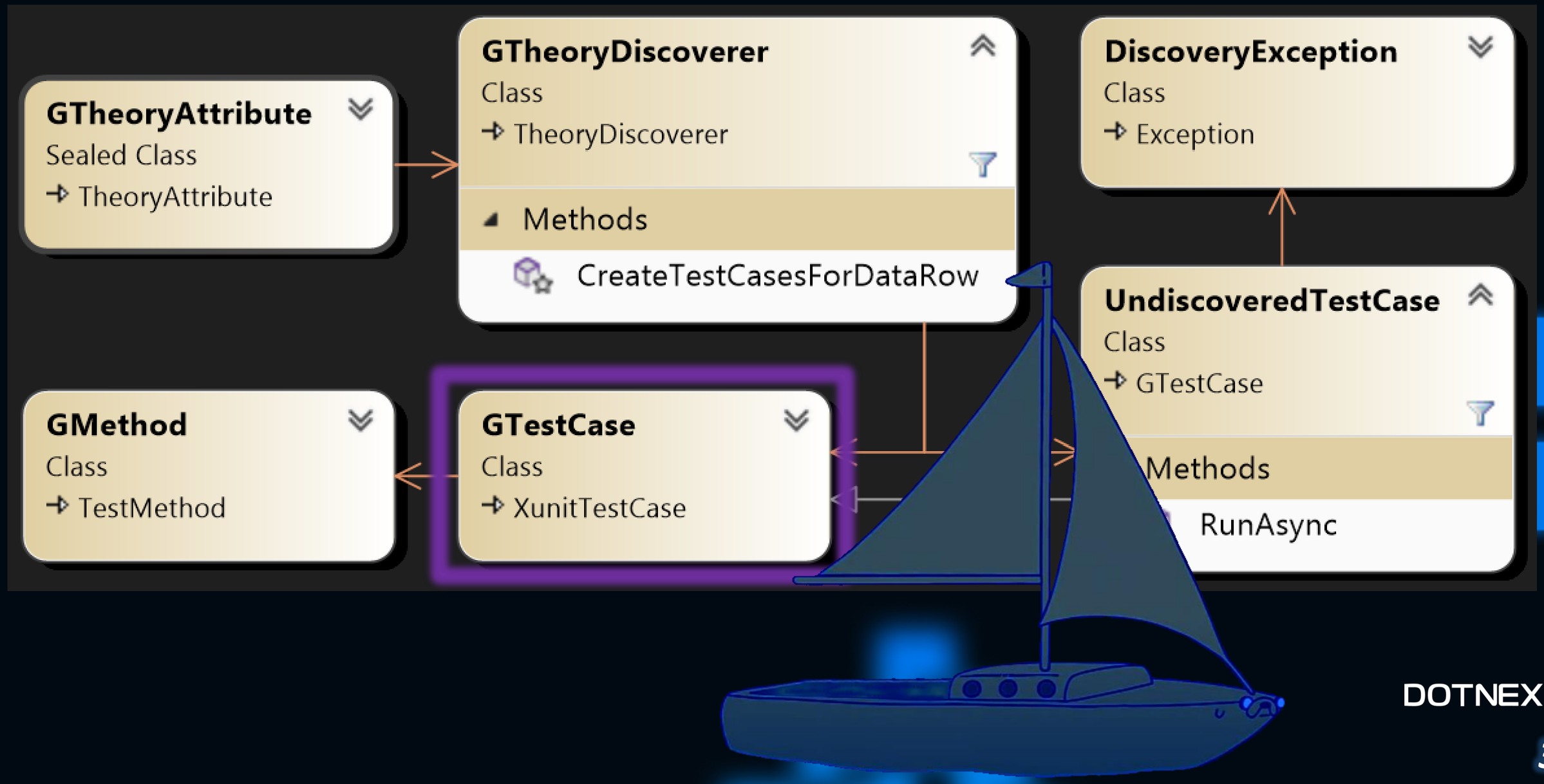
Message:

```
System.ArgumentException : GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCategory,TRequirement](Int32)' violates the
---- System.Security.VerificationException : Method Jokes.XUnitTests.IntegrationTests.Test: type argument 'System.Object' violates the
```

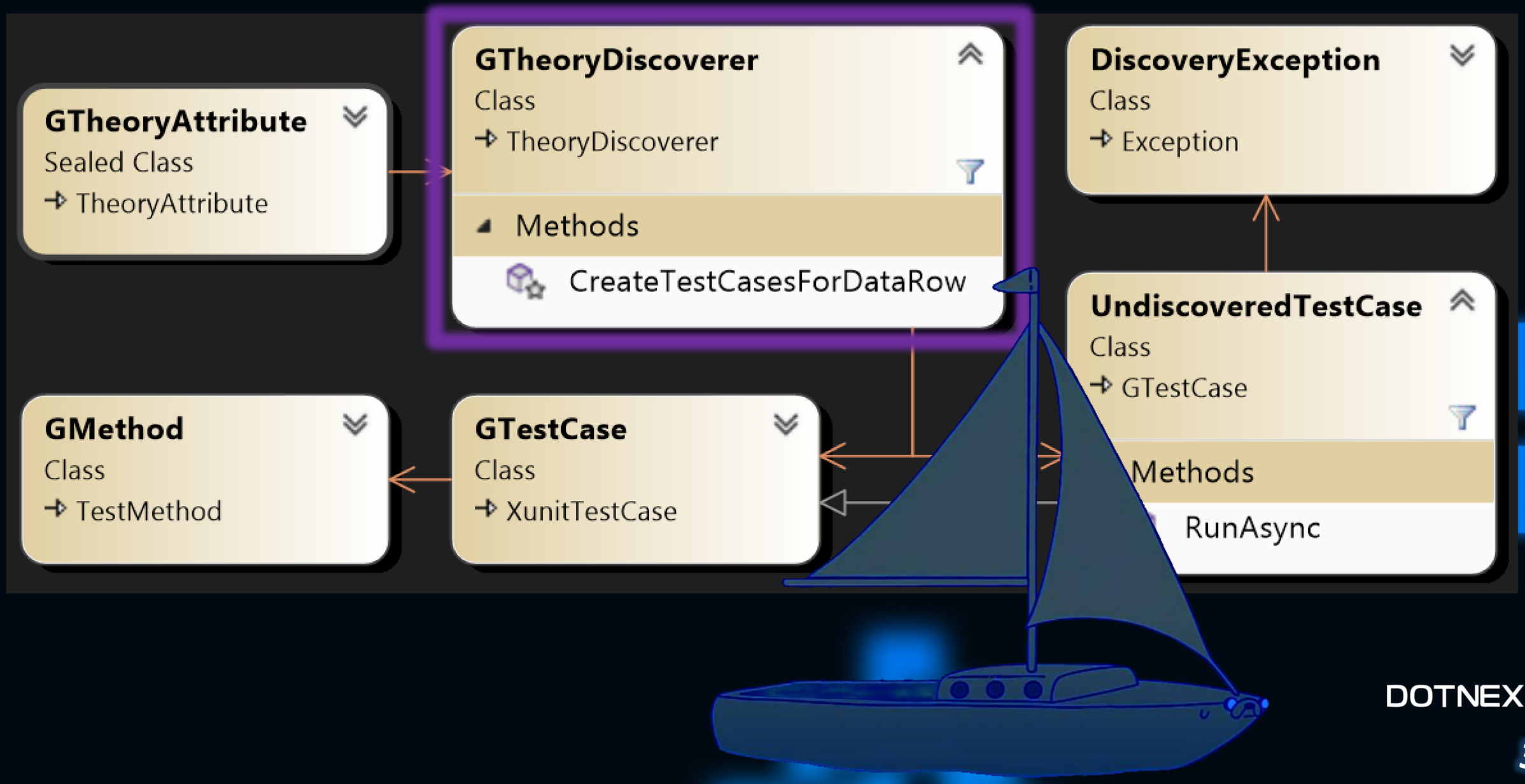
Stack Trace:

```
RuntimeType.ValidateGenericArguments(MemberInfo definition, RuntimeType[] genericArguments, Exception e)
RuntimeMethodInfo.MakeGenericMethod(Type[] methodInstantiation)
----- Inner Stack Trace -----
RuntimeMethodHandle.GetStubIfNeeded(RuntimeMethodHandleInternal method, RuntimeType declaringType, RuntimeType[] methodInstantiation)
RuntimeMethodInfo.MakeGenericMethod(Type[] methodInstantiation)
```

# Чиним

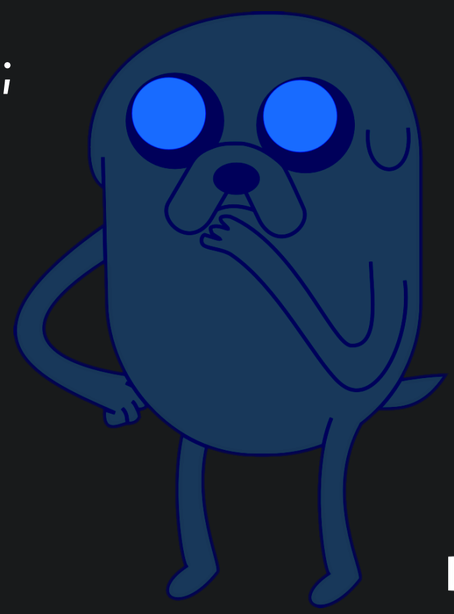

GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# Чиним

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();

        try
        {
            // ...
        }
        catch(Exception ex)
        {
            return Array.Empty<XunitTestCase>();
        }
    }
}
```



# Чиним

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();

        try
        {
            // ...
        }
        catch(Exception ex)
        {
            return Array.Empty<XunitTestCase>();
        }
    }
}
```

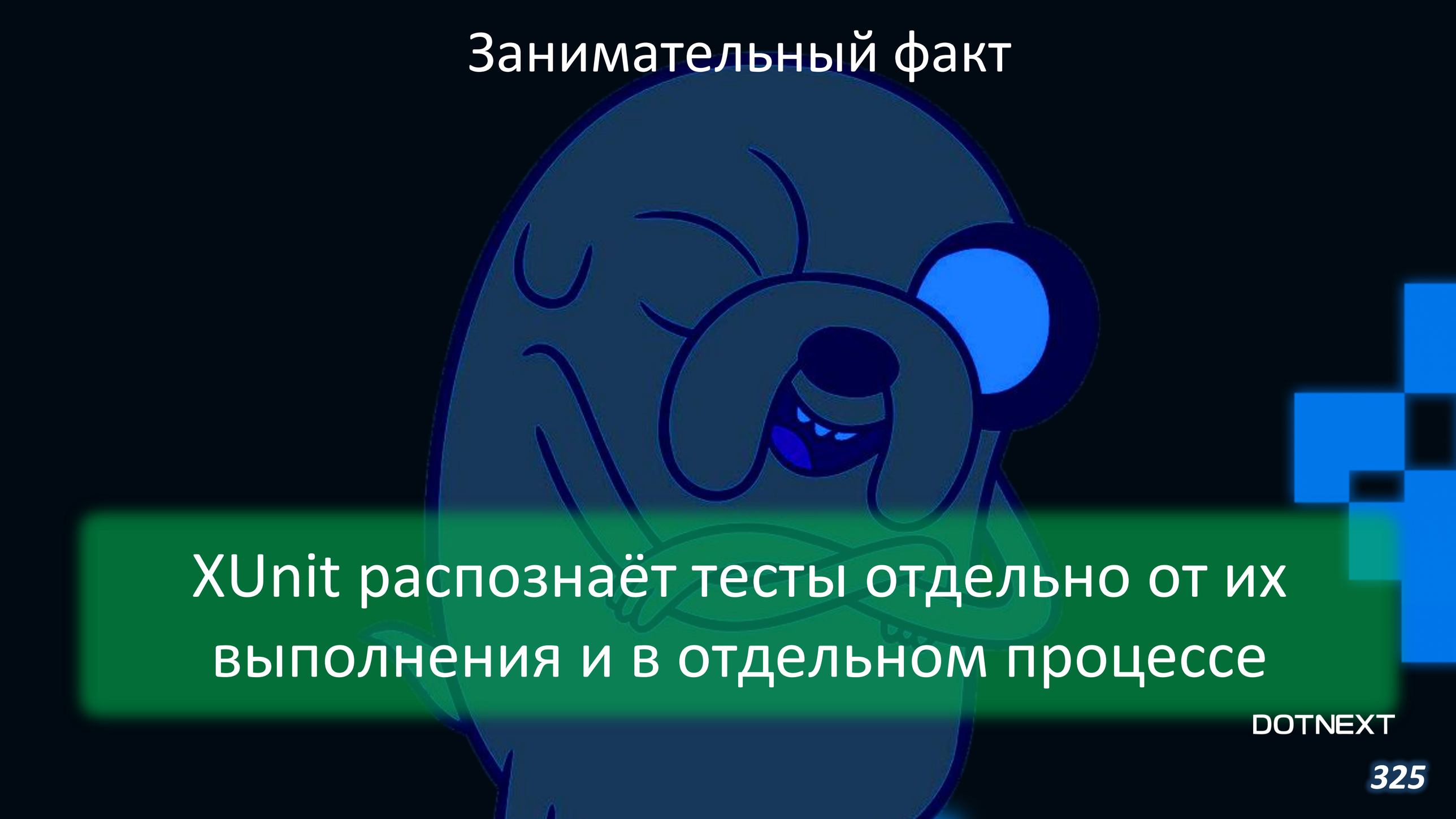
# Чиним

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();

        try
        {
            // ...
        }
        catch(Exception ex)
        {
            return Array.Empty<XunitTestCase>();
        }
    }
}
```

# Чиним

```
public class GTheoryDiscoverer : TheoryDiscoverer
{
    public GTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod testMethod,
        IAttributeInfo theoryAttribute, object[] dataRow)
    {
        var typeArguments = (Type[])dataRow[0];
        var testMethodArguments = dataRow.Skip(1).ToArray();

        try
        {
            // ...
        }
        catch(Exception ex)
        {
            return Array.Empty<XunitTestCase>();
        }
    }
}
```

# Занимательный факт



XUnit распознаёт тесты отдельно от их выполнения и в отдельном процессе

# Занимательный факт

```
try
{
    // ...
}
catch(Exception ex)
{
    System.Diagnostics.Debugger.Launch();

    // ...
}
```

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

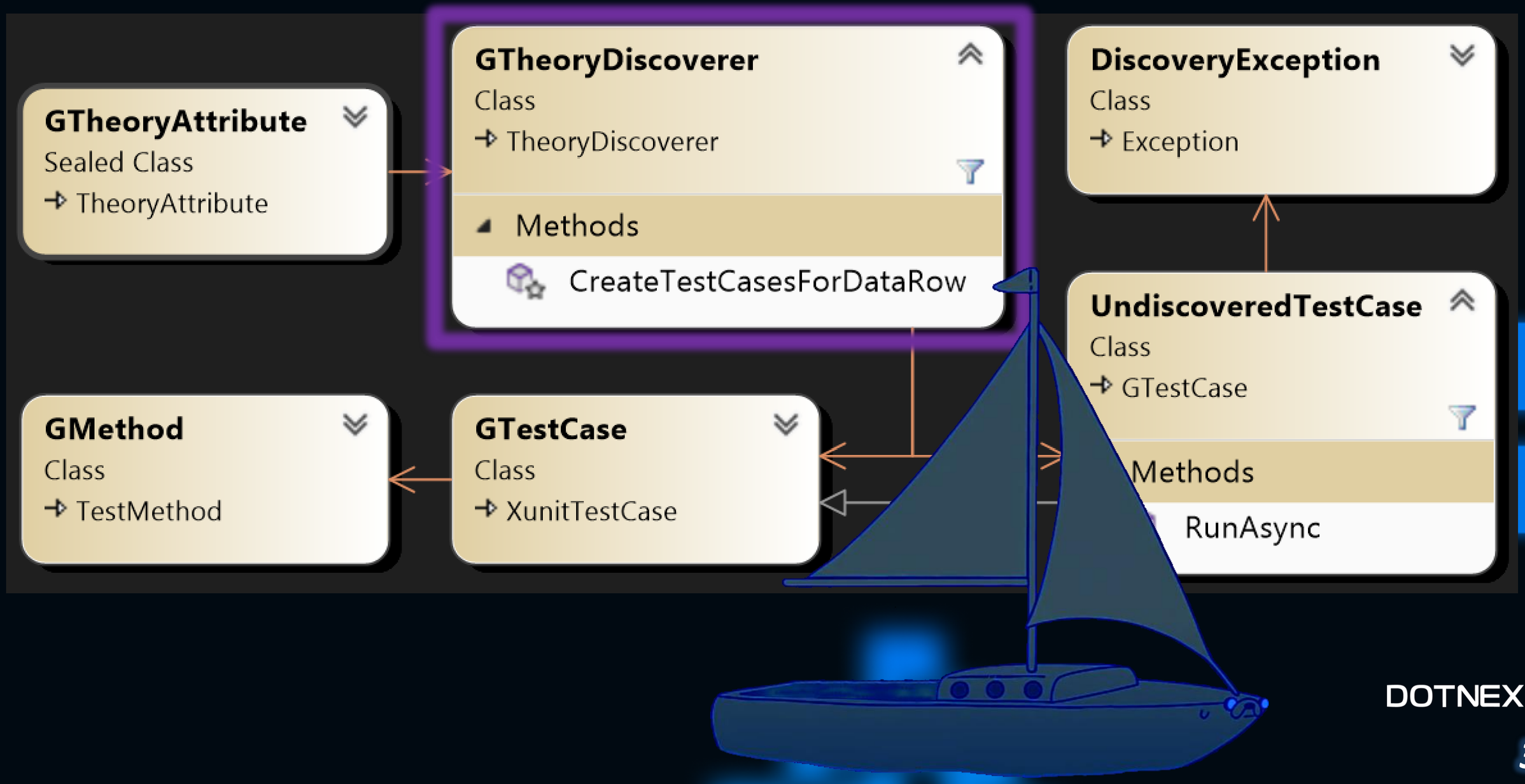
- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Check Point

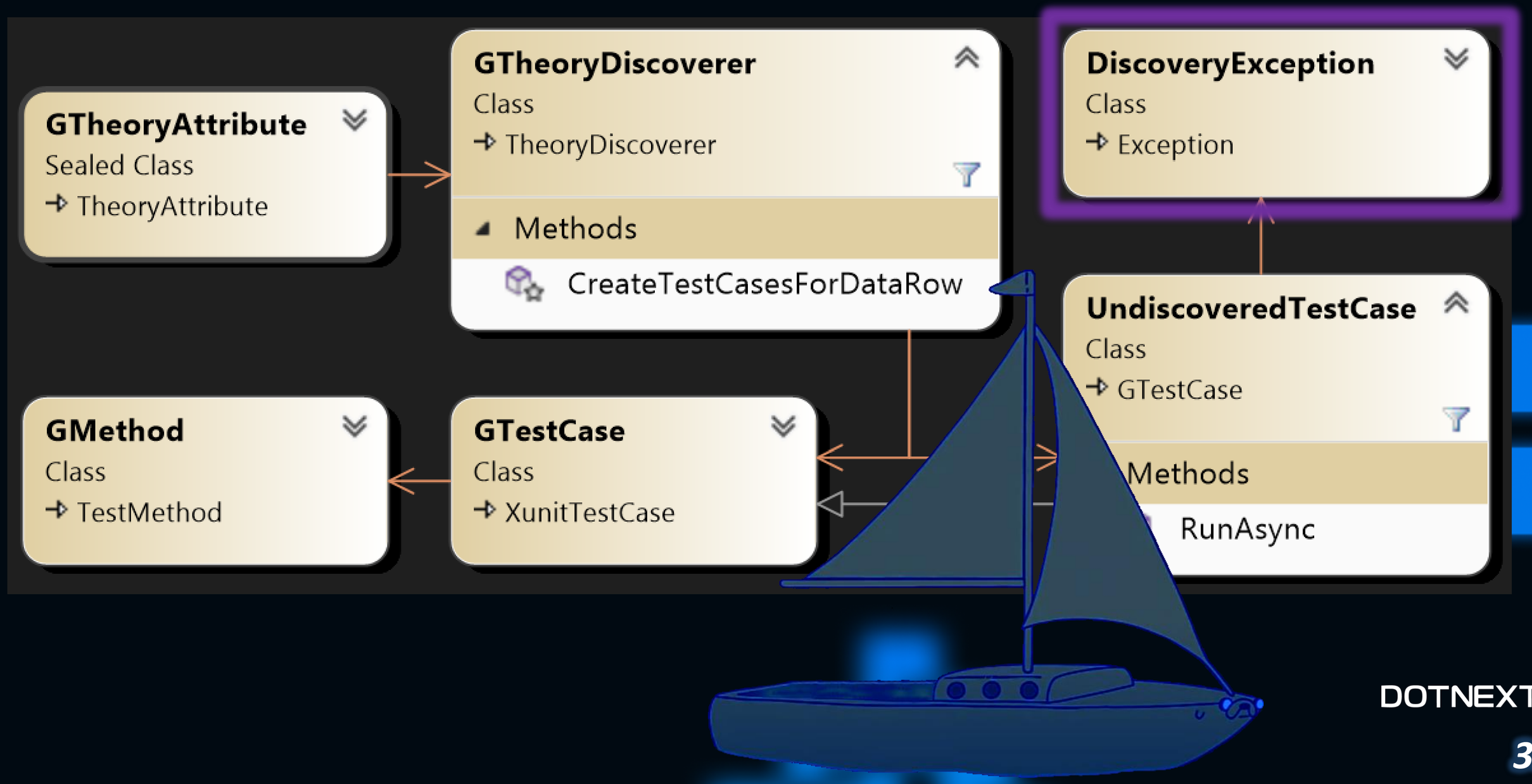
- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения тестов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# Чиним

```
public class DiscoveryException : Exception
{
    private string message;
    private string stackTrace;

    public override string Message => message;
    public override string StackTrace => stackTrace;

    public DiscoveryException(Exception exception)
    {
        message = exception.Message;
        stackTrace = exception.StackTrace;
    }
}
```

# Чиним

```
public class DiscoveryException : Exception
{
    private string message;
    private string stackTrace;

    public override string Message => message;
    public override string StackTrace => stackTrace;

    public DiscoveryException(Exception exception)
    {
        message = exception.Message;
        stackTrace = exception.StackTrace;
    }
}
```

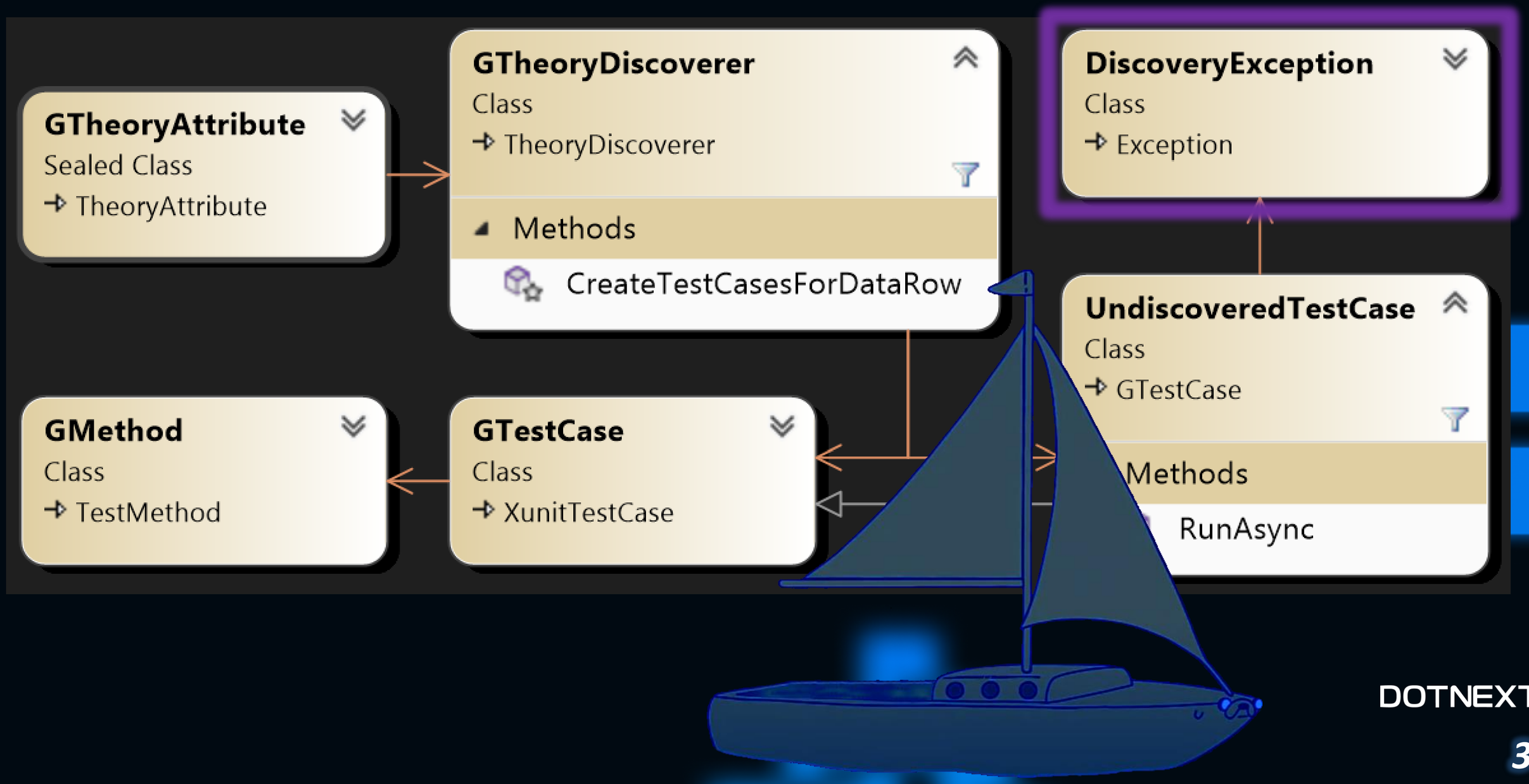
# Чиним

```
public class DiscoveryException : Exception
{
    private string message;
    private string stackTrace;

    public override string Message => message;
    public override string StackTrace => stackTrace;

    public DiscoveryException(Exception exception)
    {
        message = exception.Message;
        stackTrace = exception.StackTrace;
    }
}
```

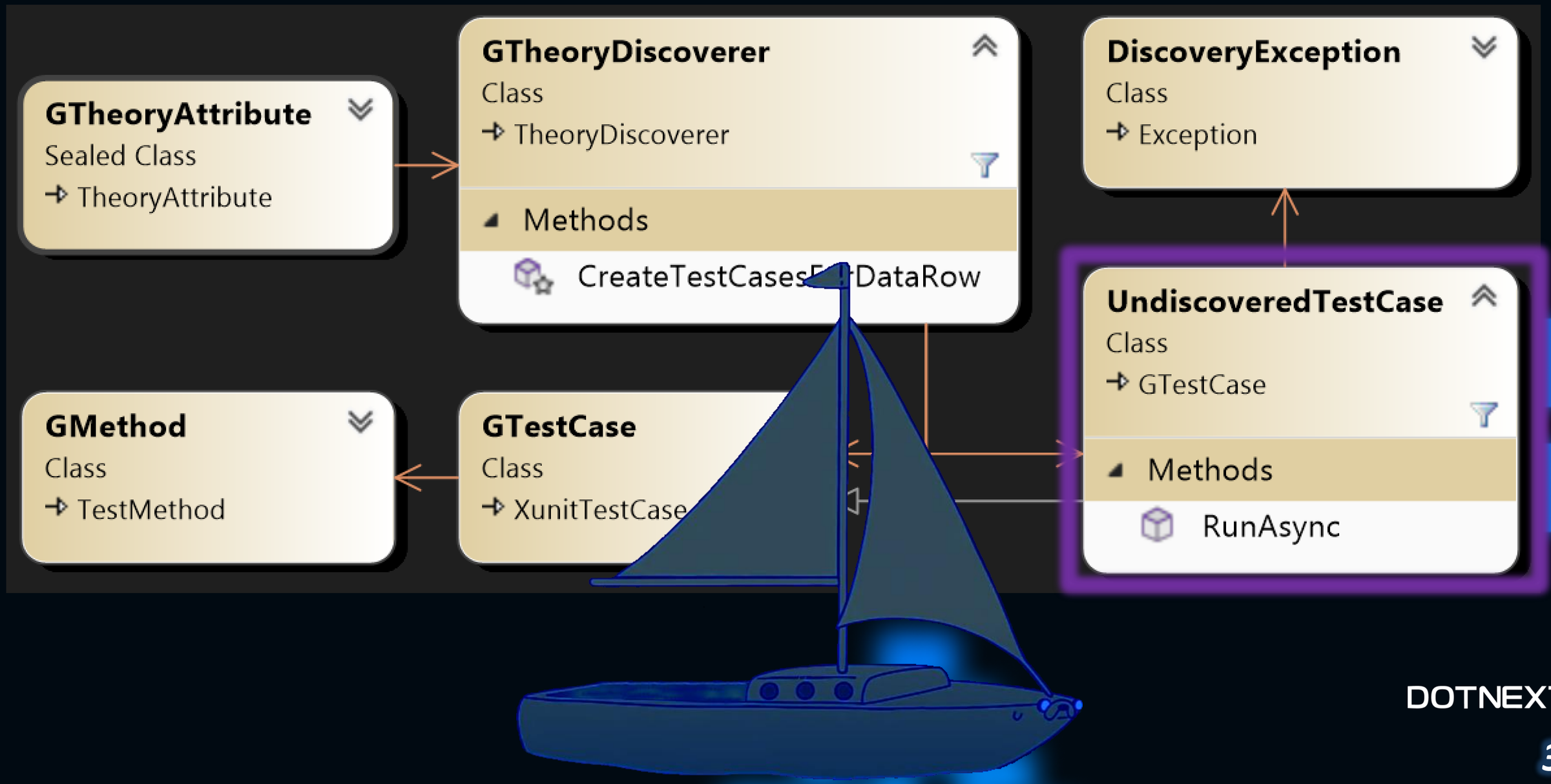
# Чиним

```
public class DiscoveryException : Exception, IXunitSerializable
{
    // ...

    [Obsolete("Called by the de-serializer ...")]
    public DiscoveryException() { }

    public void Serialize(IXunitSerializationInfo info)
    {
        info.AddValue(nameof(message), message);
        info.AddValue(nameof(stackTrace), stackTrace);
    }

    public void Deserialize(IXunitSerializationInfo info)
    {
        message = info.GetValue<string>(nameof(message));
        stackTrace = info.GetValue<string>(nameof(stackTrace));
    }
}
```

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
DiscoveryException
Class
Exception
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync

# Чиним

```
public class UndiscoveredTestCase : GTestCase
{
    public DiscoveryException Exception { get; private set; }

    public UndiscoveredTestCase(
        IMessageSink diagnosticMessageSink, TestMethodDisplay defaultMethodDisplay,
        TestMethodDisplayOptions defaultMethodDisplayOptions, ITestMethod testMethod,
        Type[] typeArguments, DiscoveryException exception, object[] testMethodArguments = null)
        : base(diagnosticMessageSink, defaultMethodDisplay, defaultMethodDisplayOptions,
               testMethod, typeArguments, testMethodArguments) =>
            Exception = exception;
}
```

# Чиним

```
public class UndiscoveredTestCase : GTestCase
{
    public DiscoveryException Exception { get; private set; }

    public UndiscoveredTestCase(
        IMessageSink diagnosticMessageSink, TestMethodDisplay defaultMethodDisplay,
        TestMethodDisplayOptions defaultMethodDisplayOptions, ITestMethod testMethod,
        Type[] typeArguments, DiscoveryException exception, object[] testMethodArguments = null)
        : base(diagnosticMessageSink, defaultMethodDisplay, defaultMethodDisplayOptions,
               testMethod, typeArguments, testMethodArguments) =>
            Exception = exception;
}
```

# Чиним

```
public class UndiscoveredTestCase : GTestCase
{
    // ...

    [Obsolete("Called by the de-serializer ...")]
    public UndiscoveredTestCase() { }

    public override void Serialize(IXunitSerializationInfo info)
    {
        base.Serialize(info);
        info.AddValue(nameof(Exception), Exception);
    }

    public override void Deserialize(IXunitSerializationInfo info)
    {
        base.Deserialize(info);
        Exception = info.GetValue<DiscoveryException>(nameof(Exception));
    }
}
```

# Чиним

```
public class UndiscoveredTestCase : GTestCase
{
    // ...

    protected override void Initialize() =>
        Method = TestMethod.Method;

    protected override string GetUniqueID()
    {
        DisplayName = Method.GetDisplayNameWithArguments(Method.Name,
            TestMethodArguments ?? new object[0], TypeArguments.Select(Reflector.Wrap).ToArray());

        return base.GetUniqueID();
    }
}
```

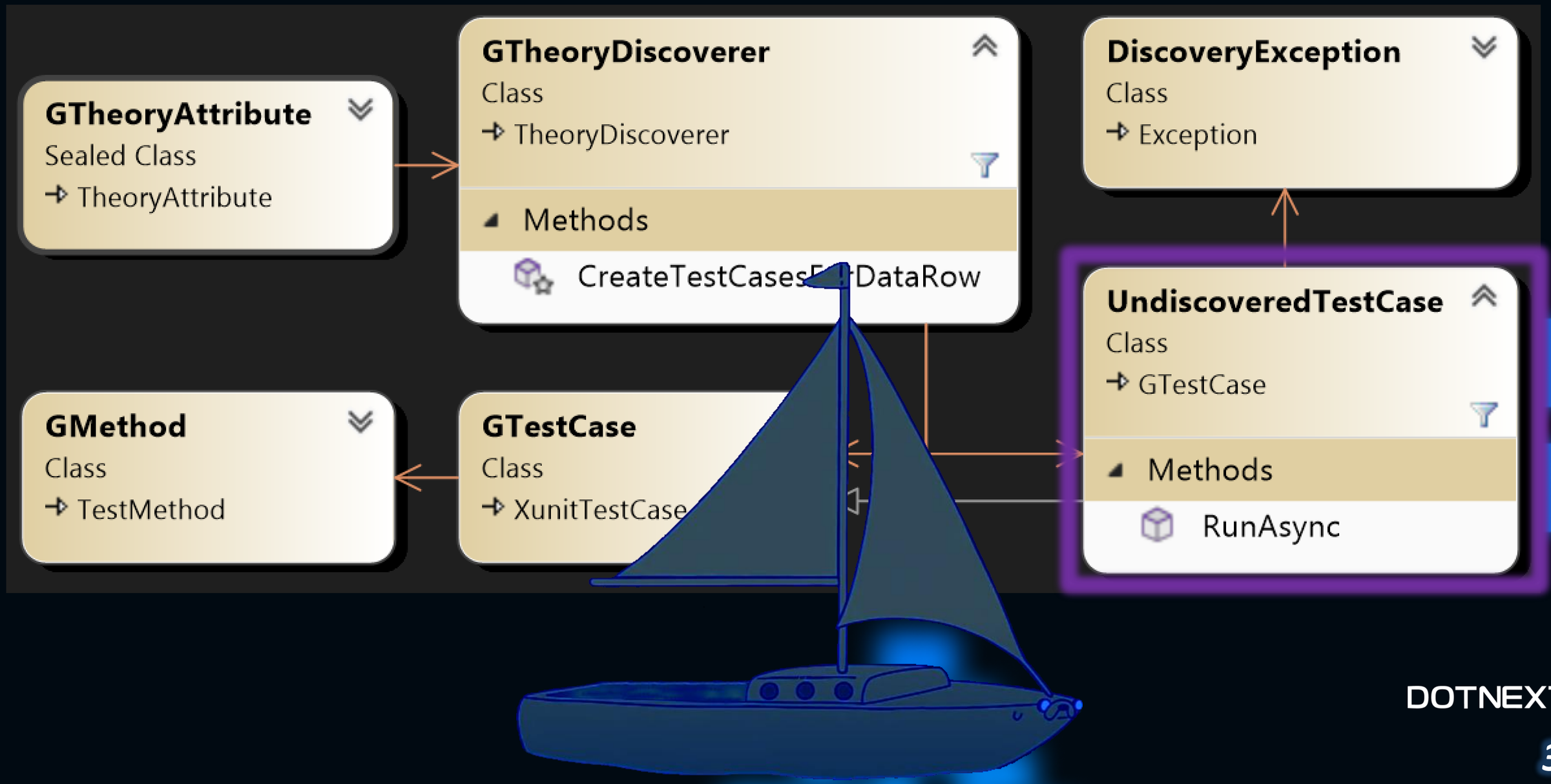
# Чиним

```
public class UndiscoveredTestCase : GTestCase
{
    // ...

    protected override void Initialize() =>
        Method = TestMethod.Method;

    protected override string GetUniqueID()
    {
        DisplayName = Method.GetDisplayNameWithArguments(Method.Name,
            TestMethodArguments ?? new object[0], TypeArguments.Select(Reflector.Wrap).ToArray());

        return base.GetUniqueID();
    }
}
```

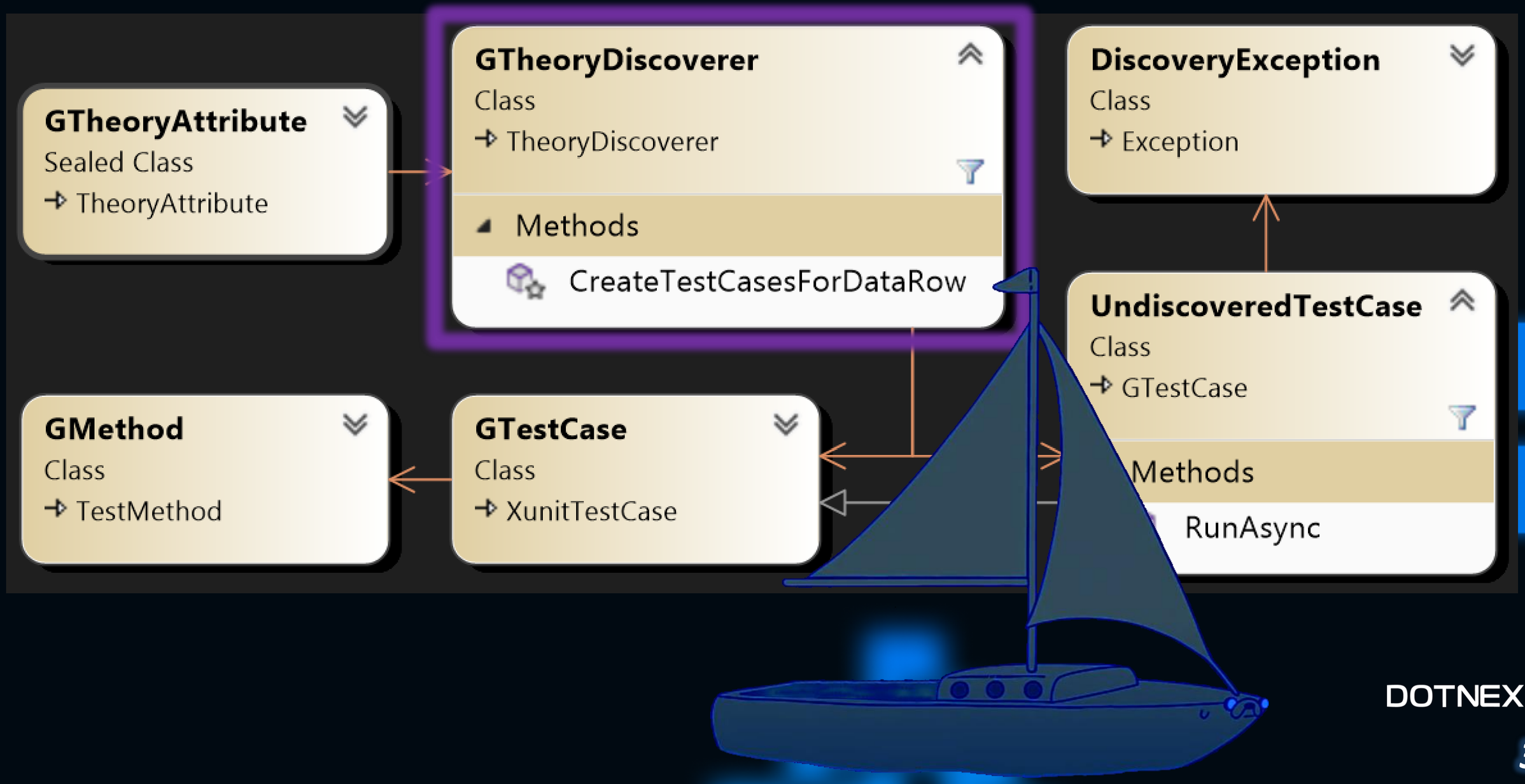
# Чиним

```
public class UndiscoveredTestCase : GTestCase
{
    // ...

    public override Task<RunSummary> RunAsync(IMessageSink diagnosticMessageSink,
        IMessageBus messageBus, object[] constructorArguments, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
    {
        aggregator.Add(Exception);

        return base.RunAsync(diagnosticMessageSink, messageBus, constructorArguments,
            aggregator, cancellationTokenSource);
    }
}
```

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
DiscoveryException
Class
Exception
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync

# Чиним


GTheoryAttribute
Sealed Class
TheoryAttribute
GTheoryDiscoverer
Class
TheoryDiscoverer
Methods
CreateTestCasesForDataRow
DiscoveryException
Class
Exception
UndiscoveredTestCase
Class
GTestCase
Methods
RunAsync
GMethod
Class
TestMethod
GTestCase
Class
XunitTestCase

# Чиним

```
var typeArguments = (Type[])dataRow[0];
var testMethodArguments = dataRow.Skip(1).ToArray();

try
{
    // ...
}
catch(Exception ex)
{
    return new[]
    {
        new UndiscoveredTestCase(
            DiagnosticMessageSink,
            discoveryOptions.MethodDisplayOrDefault(),
            discoveryOptions.MethodDisplayOptionsOrDefault(),
            testMethod,
            typeArguments,
            new DiscoveryException(ex),
            testMethodArguments)
    };
}
```

# Чиним

Test

- Jokes.XUnitTests (5)
  - Jokes.XUnitTests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<RussianJokes, ForParty, AreOfMinimalFun>(value: 75)
        - Test<RussianJokes, AboutCoders, AreOfMinimalFun>(value: 75)
        - Test<Object, Object, Object>(value: 75)
        - Test<GibraltarJokes, ForParty, AreOfMinimalFun>(value: 75)
        - Test<GibraltarJokes, AboutCoders, AreOfMinimalFun>(value: 75)

**Test Detail Summary**

Jokes.XUnitTests.IntegrationTests.Test

Source: IntegrationTests.cs line 18

Duration: 1 ms

Message:

```
Jokes.XUnitTests.DiscoveryException : GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCate
```

Stack Trace:

```
RuntimeType.ValidateGenericArguments(MemberInfo definition, RuntimeType[] genericArguments, Exception e)
RuntimeMethodInfo.MakeGenericMethod(Type[] methodInstantiation)
GMethod.set_TypeArguments(Type[] value) line 16
GMethod.ctor(ITestMethod testMethod, Type[] typeArguments) line 22
GTheoryDiscoverer.CreateTestCasesForDataRow(ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod
```

# Чиним

Test

- Jokes.XUnitTests (5)
  - Jokes.XUnitTests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<RussianJokes, ForParty, AreOfMinimalFun>(value: 75)
        - Test<RussianJokes, AboutCoders, AreOfMinimalFun>(value: 75)
        - Test<Object, Object, Object>(value: 75)
        - Test<GibraltarJokes, ForParty, AreOfMinimalFun>(value: 75)
        - Test<GibraltarJokes, AboutCoders, AreOfMinimalFun>(value: 75)

**Test Detail Summary**

Jokes.XUnitTests.IntegrationTests.Test

Source: IntegrationTests.cs line 18

Duration: 1 ms

Message:

```
Jokes.XUnitTests.DiscoveryException : GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCate
```

Stack Trace:

```
RuntimeType.ValidateGenericArguments(MemberInfo definition, RuntimeType[] genericArguments, Exception e)
RuntimeMethodInfo.MakeGenericMethod(Type[] methodInstantiation)
GMethod.set_TypeArguments(Type[] value) line 16
GMethod.ctor(ITestMethod testMethod, Type[] typeArguments) line 22
GTheoryDiscoverer.CreateTestCasesForDataRow(ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod
```

# Чиним

Test

- Jokes.XUnitTests (5)
  - Jokes.XUnitTests (5)
    - IntegrationTests (5)
      - Test (5)
        - Test<RussianJokes, ForParty, AreOfMinimalFun>(value: 75)
        - Test<RussianJokes, AboutCoders, AreOfMinimalFun>(value: 75)
        - Test<Object, Object, Object>(value: 75)
        - Test<GibraltarJokes, ForParty, AreOfMinimalFun>(value: 75)
        - Test<GibraltarJokes, AboutCoders, AreOfMinimalFun>(value: 75)

**Test Detail Summary**

Jokes.XUnitTests.IntegrationTests.Test

Source: IntegrationTests.cs line 18

Duration: 1 ms

Message:

```
Jokes.XUnitTests.DiscoveryException : GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCate
```

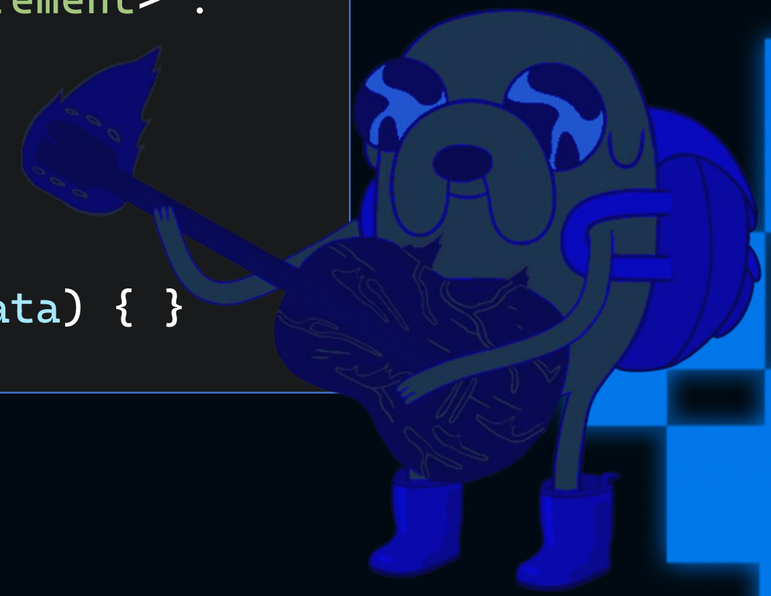
Stack Trace:

```
RuntimeType.ValidateGenericArguments(MemberInfo definition, RuntimeType[] genericArguments, Exception e)
RuntimeMethodInfo.MakeGenericMethod(Type[] methodInstantiation)
GMethod.set_TypeArguments(Type[] value) line 16
GMethod.ctor(ITestMethod testMethod, Type[] typeArguments) line 22
GTheoryDiscoverer.CreateTestCasesForDataRow(ITestFrameworkDiscoveryOptions discoveryOptions, ITestMethod
```

# Безумная идея

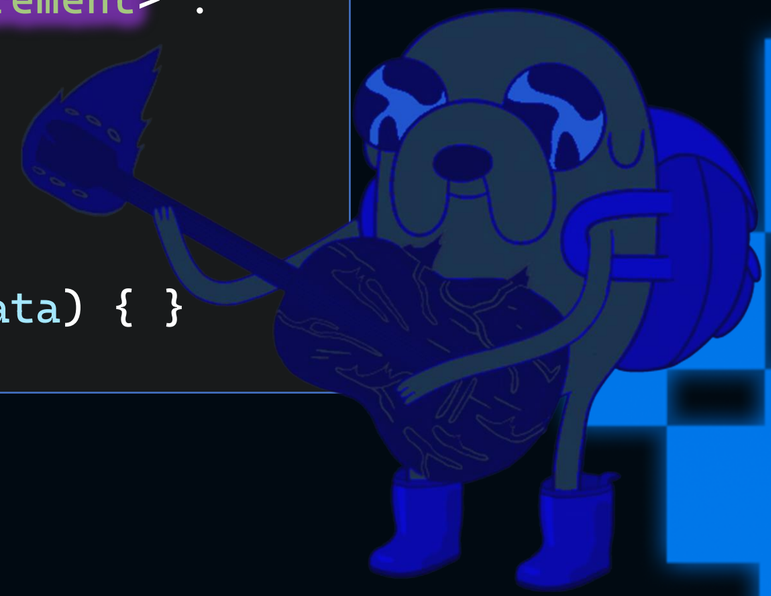
# Подготовка

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    GDataAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    public CheckAttribute(params object[] data) : base(data) { }
}
```



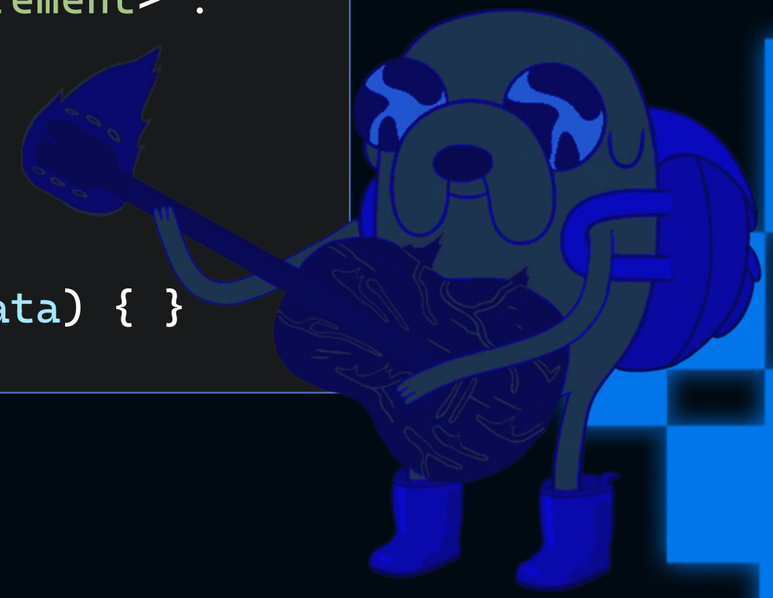
# Подготовка

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    GDataAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    public CheckAttribute(params object[] data) : base(data) { }
}
```



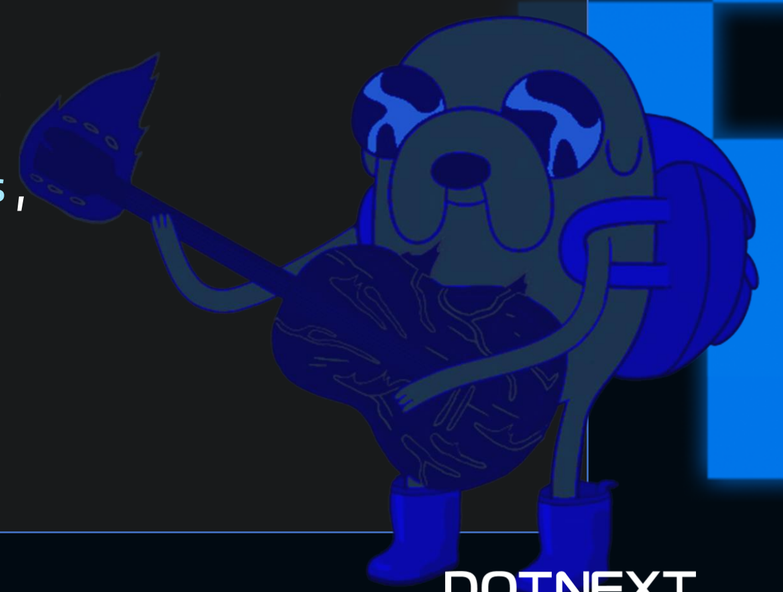
# Подготовка

```
public class CheckAttribute<TProvider, TCategory, TRequirement> :
    GDataAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    public CheckAttribute(params object[] data) : base(data) { }
}
```



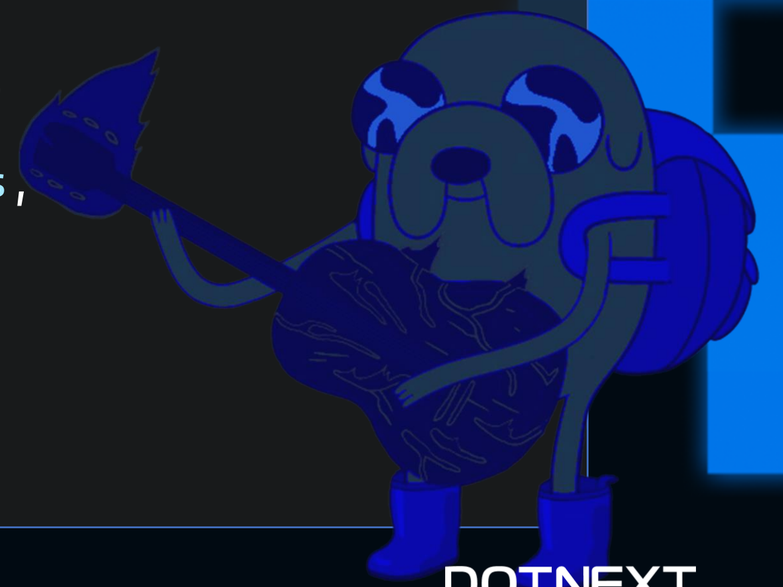
# Подготовка

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private int value;
    private readonly ITypeInfo[] typeArguments =
        Reflector.Wrap(typeof(CheckTestCase<TProvider, TCategory, TRequirement>))
        .GetGenericArguments().ToArray();

    public CheckTestCase(IMessageSink diagnosticMessageSink,
        TestMethodDisplay defaultMethodDisplay,
        TestMethodDisplayOptions defaultMethodDisplayOptions,
        ITestMethod testMethod,
        int value)
        : base(diagnosticMessageSink, defaultMethodDisplay,
               defaultMethodDisplayOptions, testMethod) =>
        this.value = value;
}
```



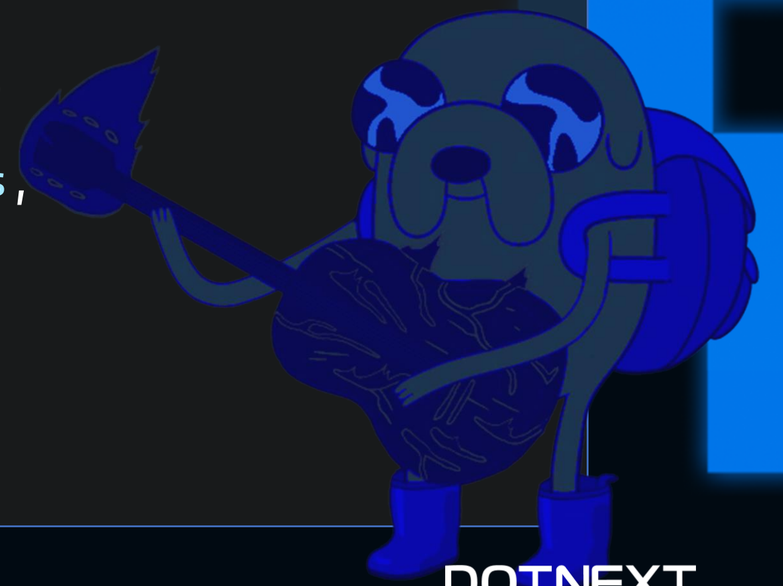
# Подготовка

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private int value;
    private readonly ITypeInfo[] typeArguments =
        Reflector.Wrap(typeof(CheckTestCase<TProvider, TCategory, TRequirement>))
        .GetGenericArguments().ToArray();

    public CheckTestCase(IMessageSink diagnosticMessageSink,
        TestMethodDisplay defaultMethodDisplay,
        TestMethodDisplayOptions defaultMethodDisplayOptions,
        ITestMethod testMethod,
        int value)
        : base(diagnosticMessageSink, defaultMethodDisplay,
               defaultMethodDisplayOptions, testMethod) =>
        this.value = value;
}
```



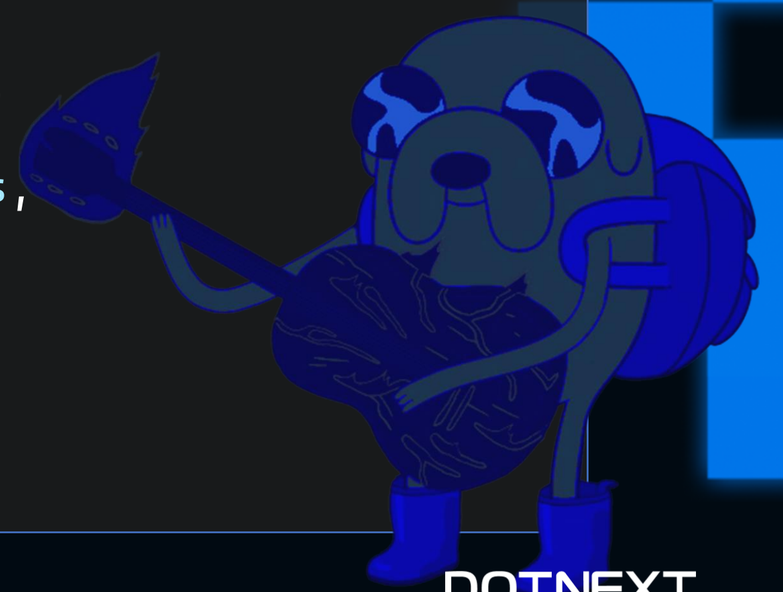
# Подготовка

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private int value;
    private readonly ITypeInfo[] typeArguments =
        Reflector.Wrap(typeof(CheckTestCase<TProvider, TCategory, TRequirement>))
        .GetGenericArguments().ToArray();

    public CheckTestCase(IMessageSink diagnosticMessageSink,
        TestMethodDisplay defaultMethodDisplay,
        TestMethodDisplayOptions defaultMethodDisplayOptions,
        ITestMethod testMethod,
        int value)
        : base(diagnosticMessageSink, defaultMethodDisplay,
               defaultMethodDisplayOptions, testMethod) =>
        this.value = value;
}
```



# Подготовка

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private int value;
    private readonly ITypeInfo[] typeArguments =
        Reflector.Wrap(typeof(CheckTestCase<TProvider, TCategory, TRequirement>))
        .GetGenericArguments().ToArray();

    public CheckTestCase(IMessageSink diagnosticMessageSink,
        TestMethodDisplay defaultMethodDisplay,
        TestMethodDisplayOptions defaultMethodDisplayOptions,
        ITestMethod testMethod,
        int value)
        : base(diagnosticMessageSink, defaultMethodDisplay,
               defaultMethodDisplayOptions, testMethod) =>
        this.value = value;
}
```



# Подготовка

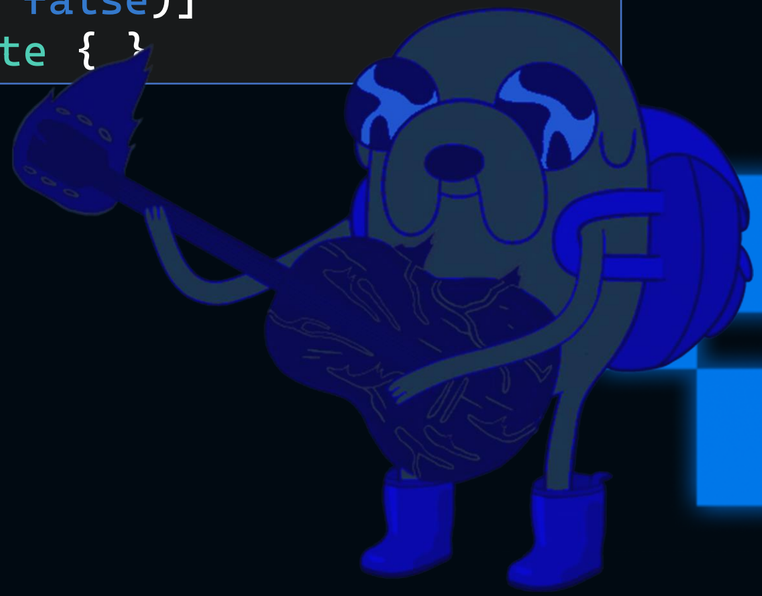
```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    [Obsolete("Called by the de-serializer ...")]
    public CheckTestCase() { }

    public override void Serialize(IXunitSerializationInfo data)
    {
        data.AddValue(nameof(value), value);
        base.Serialize(data);
    }

    public override void Deserialize(IXunitSerializationInfo data)
    {
        value = data.GetValue<int>(nameof(value));
        base.Deserialize(data);
    }
}
```

# Подготовка

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    protected override string GetDisplayName(
        IAttributeInfo factAttribute, string displayName) =>
        Method.GetDisplayNameWithArguments(Method.Name, null, typeArguments) +
        $"({nameof(value)}: {value})";

    protected override string GetUniqueID() =>
        $"{DisplayName}{{{base.GetUniqueID()}}}";
}
```

# Подготовка

```
[XunitTestCaseDiscoverer(
    $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(CheckTheoryDiscoverer)}",
    $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
public sealed class CheckTheoryAttribute : TheoryAttribute { }
```



# Подготовка

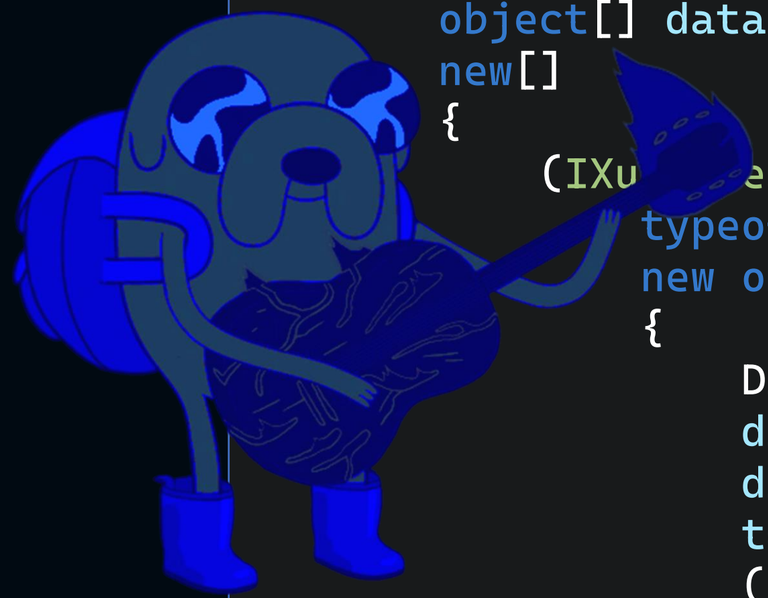
```
[XunitTestCaseDiscoverer(
    $"{nameof(Jokes)}.{nameof(XUnitTests)}.{nameof(CheckTheoryDiscoverer)}",
    $"{nameof(Jokes)}.{nameof(XUnitTests)}")]
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
public sealed class CheckTheoryAttribute : TheoryAttribute { }
```

# Подготовка

```
public class CheckTheoryDiscoverer : TheoryDiscoverer
{
    public CheckTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod,
        IAttributeInfo theoryAttribute,
        object[] dataRow) =>
        new[]
        {
            (IXu      estCase)Activator.CreateInstance(
                typeof(CheckTestCase<,,>).MakeGenericType((Type[])dataRow[0]),
                new object[]
                {
                    DiagnosticMessageSink,
                    discoveryOptions.MethodDisplayOrDefault(),
                    discoveryOptions.MethodDisplayOptionsOrDefault(),
                    testMethod,
                    (int)dataRow[1]
                })
        };
}
```

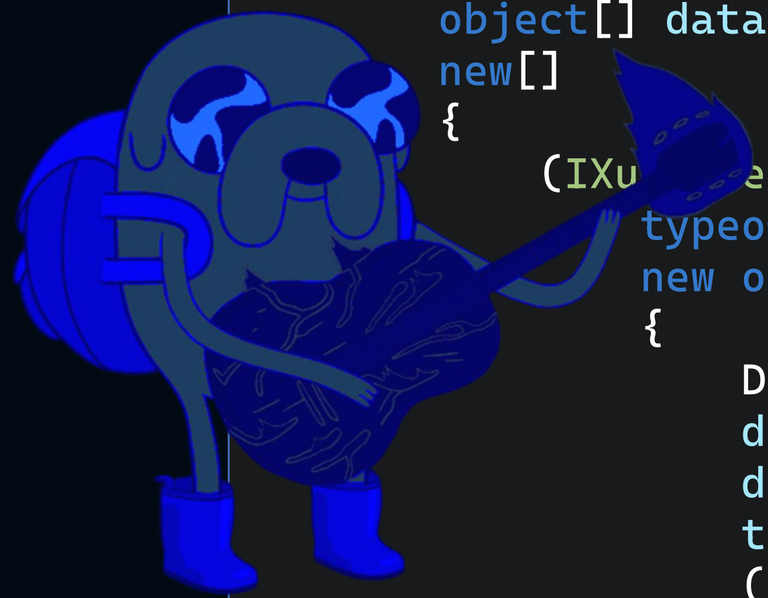
# Подготовка

```
public class CheckTheoryDiscoverer : TheoryDiscoverer
{
    public CheckTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod,
        IAttributeInfo theoryAttribute,
        object[] dataRow) =>
        new[]
        {
            (IXunitTestCase)Activator.CreateInstance(
                typeof(CheckTestCase<,,>).MakeGenericType((Type[])dataRow[0]),
                new object[]
                {
                    DiagnosticMessageSink,
                    discoveryOptions.MethodDisplayOrDefault(),
                    discoveryOptions.MethodDisplayOptionsOrDefault(),
                    testMethod,
                    (int)dataRow[1]
                })
        };
}
```



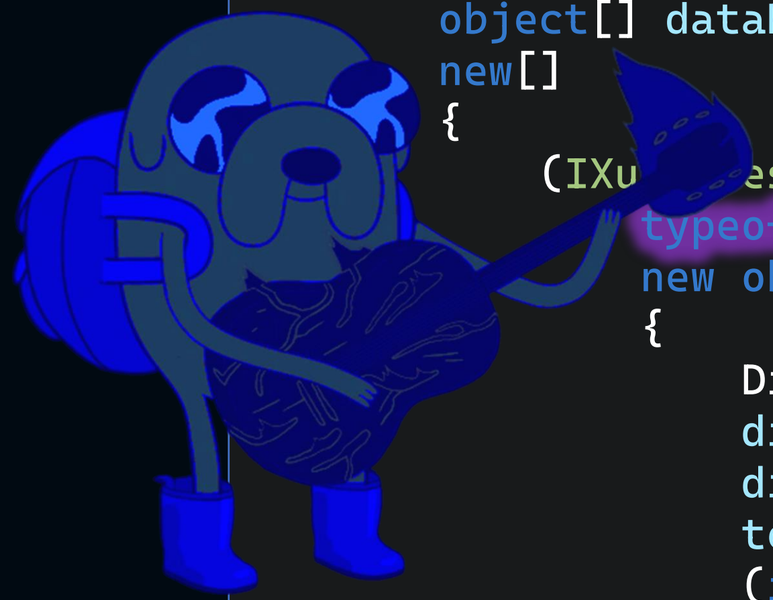
# Подготовка

```
public class CheckTheoryDiscoverer : TheoryDiscoverer
{
    public CheckTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod,
        IAttributeInfo theoryAttribute,
        object[] dataRow) =>
        new[]
        {
            (IXunitTestCase)Activator.CreateInstance(
                typeof(CheckTestCase<,,>).MakeGenericType((Type[])dataRow[0]),
                new object[]
                {
                    DiagnosticMessageSink,
                    discoveryOptions.MethodDisplayOrDefault(),
                    discoveryOptions.MethodDisplayOptionsOrDefault(),
                    testMethod,
                    (int)dataRow[1]
                })
        };
}
```



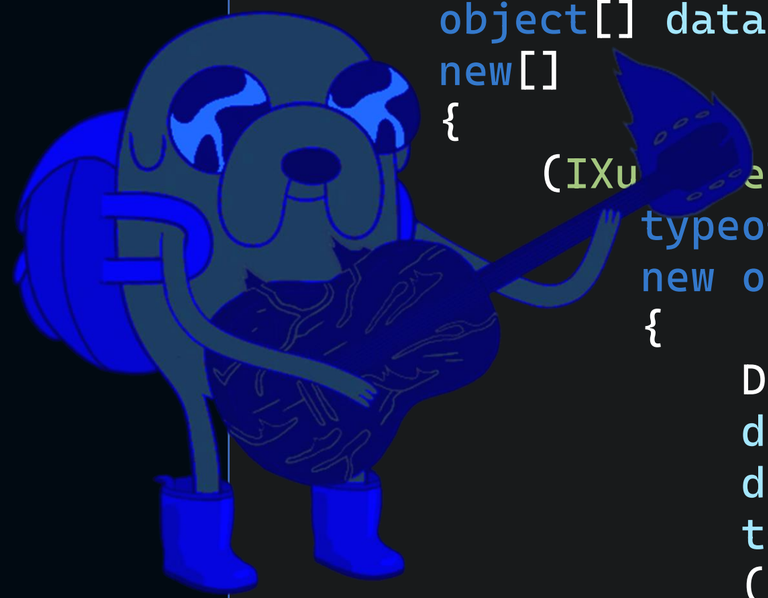
# Подготовка

```
public class CheckTheoryDiscoverer : TheoryDiscoverer
{
    public CheckTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod,
        IAttributeInfo theoryAttribute,
        object[] dataRow) =>
        new[]
        {
            (IXunitTestCase)Activator.CreateInstance(
                typeof(CheckTestCase<,,>).MakeGenericType((Type[])dataRow[0]),
                new object[]
                {
                    DiagnosticMessageSink,
                    discoveryOptions.MethodDisplayOrDefault(),
                    discoveryOptions.MethodDisplayOptionsOrDefault(),
                    testMethod,
                    (int)dataRow[1]
                })
        };
}
```



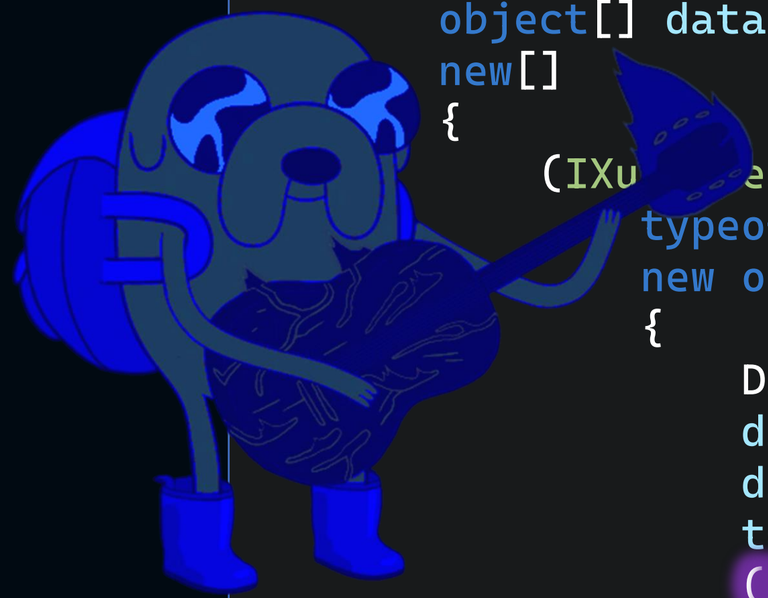
# Подготовка

```
public class CheckTheoryDiscoverer : TheoryDiscoverer
{
    public CheckTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod,
        IAttributeInfo theoryAttribute,
        object[] dataRow) =>
        new[]
        {
            (IXu      estCase)Activator.CreateInstance(
                typeof(CheckTestCase<,,>).MakeGenericType((Type[])dataRow[0]),
                new object[]
                {
                    DiagnosticMessageSink,
                    discoveryOptions.MethodDisplayOrDefault(),
                    discoveryOptions.MethodDisplayOptionsOrDefault(),
                    testMethod,
                    (int)dataRow[1]
                })
        };
}
```



# Подготовка

```
public class CheckTheoryDiscoverer : TheoryDiscoverer
{
    public CheckTheoryDiscoverer(IMessageSink diagnosticMessageSink)
        : base(diagnosticMessageSink) { }

    protected override IEnumerable<IXunitTestCase> CreateTestCasesForDataRow(
        ITestFrameworkDiscoveryOptions discoveryOptions,
        ITestMethod testMethod,
        IAttributeInfo theoryAttribute,
        object[] dataRow) =>
        new[]
        {
            (IXu      estCase)Activator.CreateInstance(
                typeof(CheckTestCase<,,>).MakeGenericType((Type[])dataRow[0]),
                new object[]
                {
                    DiagnosticMessageSink,
                    discoveryOptions.MethodDisplayOrDefault(),
                    discoveryOptions.MethodDisplayOptionsOrDefault(),
                    testMethod,
                    (int)dataRow[1]
                })
        };
}
```



# Подготовка

```
public class CheckActionAttribute<TProvider, TCategory, TRequirement> :
    BeforeAfterTestAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckActionAttribute(int value) => this.value = value;
}
```

# Подготовка

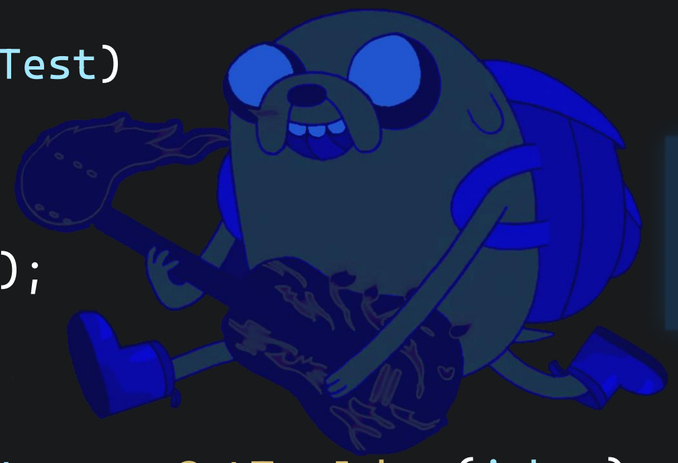
```
public class CheckActionAttribute<TProvider, TCategory, TRequirement> :
    BeforeAfterTestAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckActionAttribute(int value) => this.value = value;
}
```

# Подготовка

```
public class CheckActionAttribute<TProvider, TCategory, TRequirement> :
    BeforeAfterTestAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckActionAttribute(int value) => this.value = value;
}
```

# Подготовка

```
public class CheckActionAttribute<TProvider, TCategory, TRequirement> :
    BeforeAfterTestAttribute
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public override void After(MethodInfo methodUnderTest)
    {
        // Arrange
        IJokeProvider provider = new TProvider();
        IEnumerable<IJoke> jokes = provider.GetJokes();
        ICategory category = new TCategory();

        // Act
        IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

        // Assert
        IRequirement requirement = new TRequirement();
        bool actual = requirement.IsMet(jokesOnCategory, value);
        Assert.True(actual);
    }
}
```



# Подготовка

```
public class CheckTestCaseRunner<TProvider, TCategory, TRequirement> : XunitTestCaseRunner
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckTestCaseRunner(IXunitTestCase testCase, string displayName,
        string skipReason, object[] constructorArguments, int value,
        IMessageBus messageBus, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
        : base(testCase, displayName, skipReason, constructorArguments, null,
            messageBus, aggregator, cancellationTokenSource) =>
        this.value = value;

    protected override List<BeforeAfterTestAttribute> GetBeforeAfterTestAttributes() =>
        base.GetBeforeAfterTestAttributes()
        .Prepend(new CheckActionAttribute<TProvider, TCategory, TRequirement>(value))
        .ToList();
}
```

# Подготовка

```
public class CheckTestCaseRunner<TProvider, TCategory, TRequirement> : XunitTestCaseRunner
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckTestCaseRunner(IXunitTestCase testCase, string displayName,
        string skipReason, object[] constructorArguments, int value,
        IMessageBus messageBus, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
        : base(testCase, displayName, skipReason, constructorArguments, null,
            messageBus, aggregator, cancellationTokenSource) =>
        this.value = value;

    protected override List<BeforeAfterTestAttribute> GetBeforeAfterTestAttributes() =>
        base.GetBeforeAfterTestAttributes()
        .Prepend(new CheckActionAttribute<TProvider, TCategory, TRequirement>(value))
        .ToList();
}
```

# Подготовка

```
public class CheckTestCaseRunner<TProvider, TCategory, TRequirement> : XunitTestCaseRunner
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckTestCaseRunner(IXunitTestCase testCase, string displayName,
        string skipReason, object[] constructorArguments, int value,
        IMessageBus messageBus, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
        : base(testCase, displayName, skipReason, constructorArguments, null,
            messageBus, aggregator, cancellationTokenSource) =>
        this.value = value;

    protected override List<BeforeAfterTestAttribute> GetBeforeAfterTestAttributes() =>
        base.GetBeforeAfterTestAttributes()
        .Prepend(new CheckActionAttribute<TProvider, TCategory, TRequirement>(value))
        .ToList();
}
```

# Подготовка

```
public class CheckTestCaseRunner<TProvider, TCategory, TRequirement> : XunitTestCaseRunner
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckTestCaseRunner(IXunitTestCase testCase, string displayName,
        string skipReason, object[] constructorArguments, int value,
        IMessageBus messageBus, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
        : base(testCase, displayName, skipReason, constructorArguments, null,
            messageBus, aggregator, cancellationTokenSource) =>
        this.value = value;

    protected override List<BeforeAfterTestAttribute> GetBeforeAfterTestAttributes() =>
        base.GetBeforeAfterTestAttributes()
        .Prepend(new CheckActionAttribute<TProvider, TCategory, TRequirement>(value))
        .ToList();
}
```

# Подготовка

```
public class CheckTestCaseRunner<TProvider, TCategory, TRequirement> : XunitTestCaseRunner
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckTestCaseRunner(IXunitTestCase testCase, string displayName,
        string skipReason, object[] constructorArguments, int value,
        IMessageBus messageBus, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
        : base(testCase, displayName, skipReason, constructorArguments, null,
            messageBus, aggregator, cancellationTokenSource) =>
        this.value = value;

    protected override List<BeforeAfterTestAttribute> GetBeforeAfterTestAttributes() =>
        base.GetBeforeAfterTestAttributes()
        .Prepend(new CheckActionAttribute<TProvider, TCategory, TRequirement>(value))
        .ToList();
}
```

# Подготовка

```
public class CheckTestCaseRunner<TProvider, TCategory, TRequirement> : XunitTestCaseRunner
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    private readonly int value;

    public CheckTestCaseRunner(IXunitTestCase testCase, string displayName,
        string skipReason, object[] constructorArguments, int value,
        IMessageBus messageBus, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource)
        : base(testCase, displayName, skipReason, constructorArguments, null,
            messageBus, aggregator, cancellationTokenSource) =>
        this.value = value;

    protected override List<BeforeAfterTestAttribute> GetBeforeAfterTestAttributes() =>
        base.GetBeforeAfterTestAttributes()
        .Prepend(new CheckActionAttribute<TProvider, TCategory, TRequirement>(value))
        .ToList();
}
```

# Готово!

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public override Task<RunSummary> RunAsync(
        IMessageSink diagnosticMessageSink, IMessageBus messageBus,
        object[] constructorArguments, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource) =>
        new CheckTestCaseRunner<TProvider, TCategory, TRequirement>(this,
            DisplayName, SkipReason, constructorArguments, value, messageBus,
            aggregator, cancellationTokenSource).RunAsync();
```

# Готово!

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public override Task<RunSummary> RunAsync(
        IMessageSink diagnosticMessageSink, IMessageBus messageBus,
        object[] constructorArguments, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource) =>
        new CheckTestCaseRunner<TProvider, TCategory, TRequirement>(this,
            DisplayName, SkipReason, constructorArguments, value, messageBus,
            aggregator, cancellationTokenSource).RunAsync();
```

# Готово!

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public override Task<RunSummary> RunAsync(
        IMessageSink diagnosticMessageSink, IMessageBus messageBus,
        object[] constructorArguments, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource) =>
        new CheckTestCaseRunner<TProvider, TCategory, TRequirement>(this,
            DisplayName, SkipReason, constructorArguments, value, messageBus,
            aggregator, cancellationTokenSource).RunAsync();
```

# Готово!

```
public class CheckTestCase<TProvider, TCategory, TRequirement> : XunitTestCase
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // ...

    public override Task<RunSummary> RunAsync(
        IMessageSink diagnosticMessageSink, IMessageBus messageBus,
        object[] constructorArguments, ExceptionAggregator aggregator,
        CancellationTokenSource cancellationTokenSource) =>
        new CheckTestCaseRunner<TProvider, TCategory, TRequirement>(this,
            DisplayName, SkipReason, constructorArguments, value, messageBus,
            aggregator, cancellationTokenSource).RunAsync();
```

# Проверка

```
public class IntegrationTests
{
    [CheckTheory]
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(76)]
    public void TestRussianJokes() { }

    [CheckTheory]
    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
}
```

# Проверка

```
public class IntegrationTests
{
    [CheckTheory]
    [Check<RussianJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<RussianJokes, ForParty, AreOfMinimalFun>(76)]
    public void TestRussianJokes() { }

    [CheckTheory]
    [Check<GibraltarJokes, AboutCoders, AreOfMinimalFun>(75)]
    [Check<GibraltarJokes, ForParty, AreOfMinimalFun>(75)]
    public void TestGibraltarJokes() { }
}
```

# Проверка

Test

- Jokes.XUnitTests (4)
  - Jokes.XUnitTests (4)
    - IntegrationTests (4)
      - TestRussianJokes (2)
        - TestRussianJokes<RussianJokes, ForParty, AreOfMinimalFun>(value: 76)
        - TestRussianJokes<RussianJokes, AboutCoders, AreOfMinimalFun>(value: 75)
      - TestGibraltarJokes (2)
        - TestGibraltarJokes<GibraltarJokes, ForParty, AreOfMinimalFun>(value: 75)
        - TestGibraltarJokes<GibraltarJokes, AboutCoders, AreOfMinimalFun>(value: 75)

**Test Detail Summary**

TestRussianJokes<RussianJokes, ForParty, AreOfMinimalFun>(value: 76)

Source: IntegrationTests.cs line 12

Duration: < 1 ms

Message:

```
Assert.True() Failure
Expected: True
Actual:   False
```

Stack Trace:

```
CheckActionAttribute`3.Af                                    UnderTest)
```

# Check Point

- ✓ XUnit разделяет атрибуты-тесты и атрибуты-данные
- ✓ Для обнаружения атрибутов и данных нужны отдельные Discoverer
- ✓ Точкой расширения является комбинация атрибут + Discoverer
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Обмен данными происходит с помощью сериализации

- ✓ Необработанные исключения ломают распознавание
- ✓ Наследование от Xunit.Sdk.XunitTestCase позволяет гибко настраивать тесты
- ✓ Xunit.Sdk.BeforeAfterTestAttribute — АОП в XUnit

# С XUnit - ВСЁ

# MSTest

# Анализ

```
//
// Summary:
//     Attribute to define in-line data for a test method.
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
public class DataRowAttribute : Attribute, ITestDataSource
{
    // ...
}
```

# Анализ

```
//
// Summary:
//     Test data source for data driven tests.
public interface ITestDataSource
{
    //
    // Summary:
    //     Gets the test data from custom test data source.
    IEnumerable<object[]> GetData(MethodInfo methodInfo);

    //
    // Summary:
    //     Gets the display name corresponding to test data row for displaying in TestResults.
    string GetDisplayName(MethodInfo methodInfo, object[] data);
}
```

# Анализ

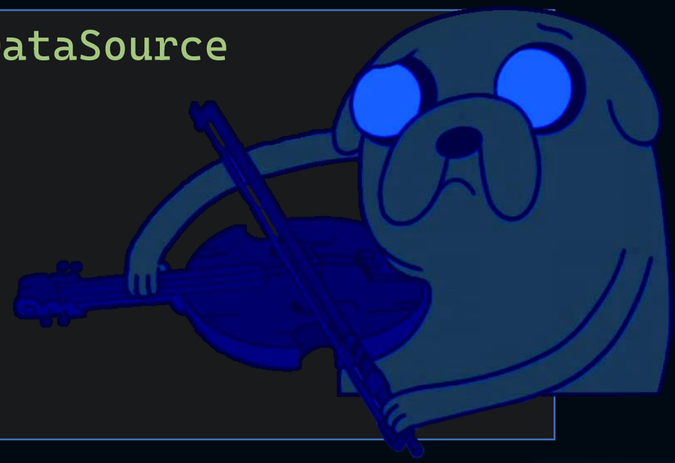
```
//
// Summary:
//     Test data source for data driven tests.
public interface ITestDataSource
{
    //
    // Summary:
    //     Gets the test data from custom test data source.
    IEnumerable<object[]> GetData(MethodInfo methodInfo);

    //
    // Summary:
    //     Gets the display name corresponding to test data row for displaying in TestResults.
    string GetDisplayName(MethodInfo methodInfo, object[] data);
}
```

# Анализ

```
//
// Summary:
//     Test data source for data driven tests.
public interface ITestDataSource
{
    //
    // Summary:
    //     Gets the test data from custom test data source.
    IEnumerable<object[]> GetData(MethodInfo methodInfo);

    //
    // Summary:
    //     Gets the display name corresponding to test data row for displaying in TestResults.
    string GetDisplayName(MethodInfo methodInfo, object[] data);
}
```

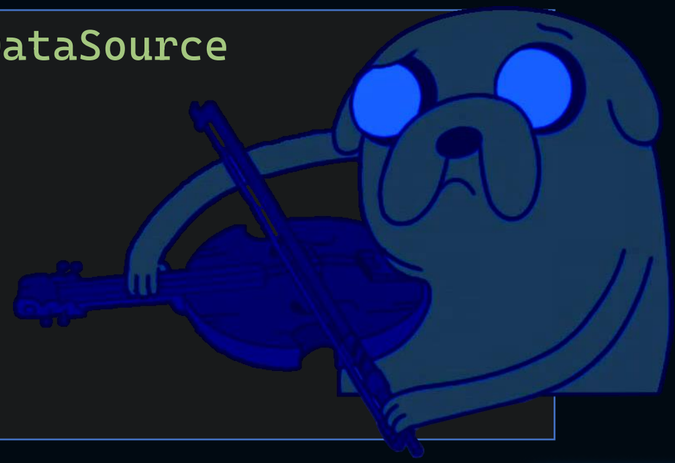
# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    private readonly Type[] typeArguments;

    public GDataRowAttribute(params object[] arguments)
        : base(arguments[0], arguments.Skip(1).ToArray())
        typeArguments = GetType().GetGenericArguments();
}
```



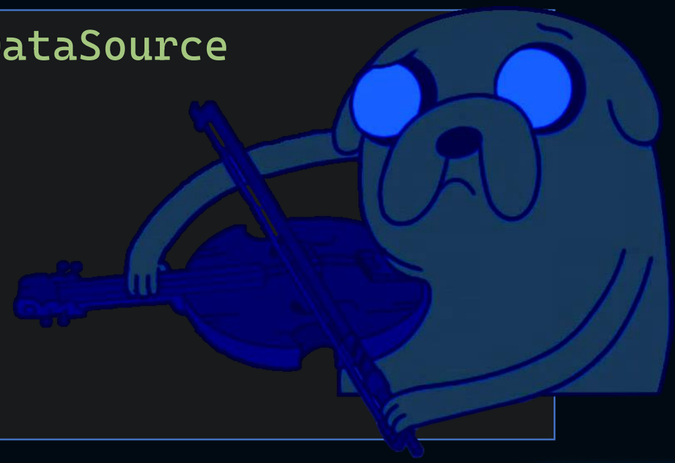
# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    private readonly Type[] typeArguments;

    public GDataRowAttribute(params object[] arguments)
        : base(arguments[0], arguments.Skip(1).ToArray())
        typeArguments = GetType().GetGenericArguments();
}
```



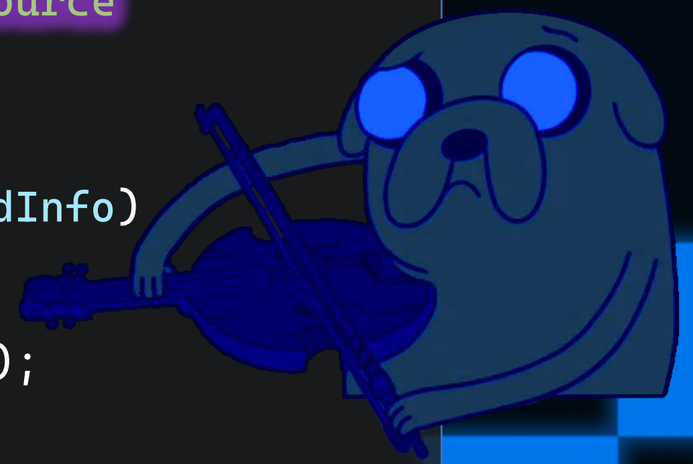
# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    private readonly Type[] typeArguments;

    public GDataRowAttribute(params object[] arguments)
        : base(arguments[0], arguments.Skip(1).ToArray())
        typeArguments = GetType().GetGenericArguments();
}
```



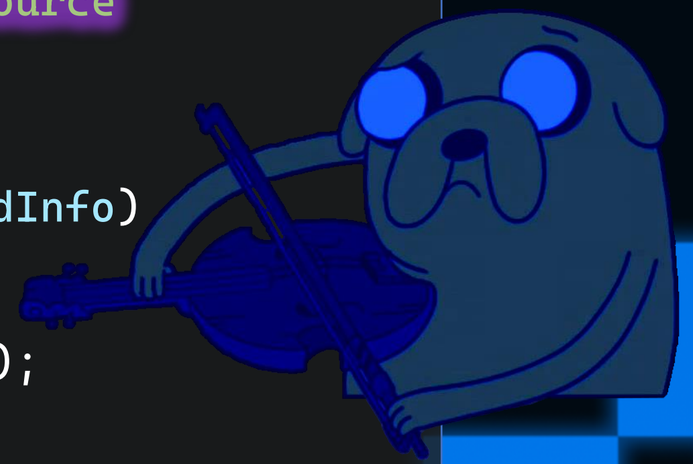
# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    // ...

    public new IEnumerable<object[]> GetData(MethodInfo methodInfo)
    {
        foreach (var data in base.GetData(methodInfo))
            yield return data.Prepend(typeArguments).ToArray();
    }
}
```



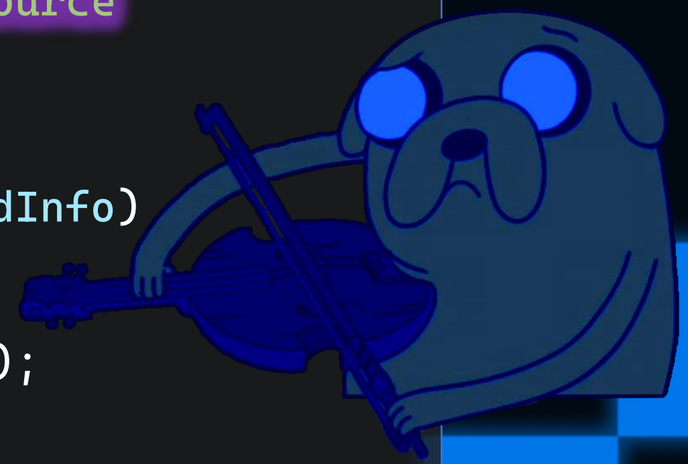
# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    // ...

    public new IEnumerable<object[]> GetData(MethodInfo methodInfo)
    {
        foreach (var data in base.GetData(methodInfo))
            yield return data.Prepend(typeArguments).ToArray();
    }
}
```



# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    // ...

    public new IEnumerable<object[]> GetData(MethodInfo methodInfo)
    {
        foreach (var data in base.GetData(methodInfo))
            yield return data.Prepend(typeArguments).ToArray();
    }
}
```



# Решение

```
public class GDataRowAttribute : DataRowAttribute, ITestDataSource
{
    // ...

    public new string GetDisplayName(MethodInfo methodInfo, object[] data)
    {
        if (!string.IsNullOrWhiteSpace(DisplayName))
            return DisplayName;

        return $"{methodInfo.Name}<{
            string.Join(',', ((Type[])data[0]).Select(type => type.Name))}>({
            string.Join(',', data.Skip(1))})";
    }
}
```

# Решение

```
public class ReflectionHelper
{
    private readonly Type type;

    public static BindingFlags Flags { get; } =
        BindingFlags.Public |
        BindingFlags.NonPublic |
        BindingFlags.Instance;

    public ReflectionHelper(Type type) => this.type = type;
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public object InvokeMethod(object target, string name, Type[] types, object[] arguments)
    {
        var method = type.GetMethod(name, Flags, types) ??
            throw new ArgumentException($"No such method '{name}' found on {
                type.FullName}.", nameof(name));

        return method.Invoke(target, arguments);
    }
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public object GetProperty(object target, string name)
    {
        var property = type.GetProperty(name, Flags) ??
            throw new ArgumentException($"No such property '{name}' found on {
                type.FullName}.", nameof(name));

        return property.GetValue(target);
    }
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

# Решение

```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

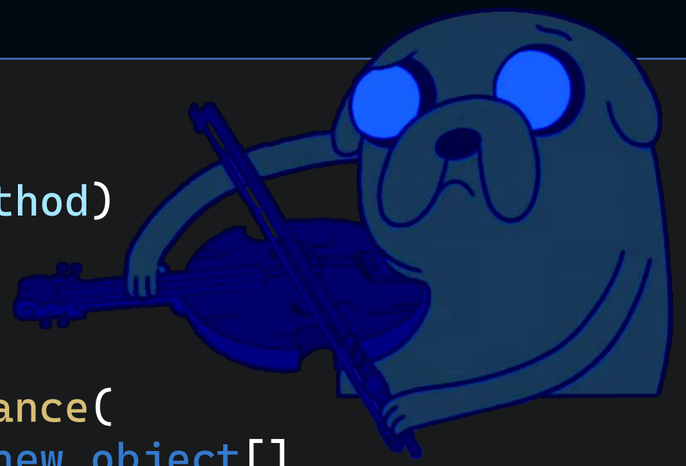
# Решение

```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

# Решение

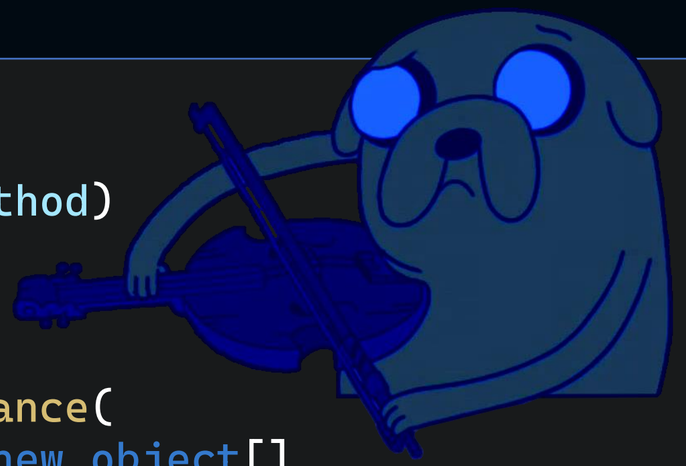
```
public class ReflectionHelper
{
    // ...

    public MethodInfo MakeGenericMethod(ITestMethod testMethod)
    {
        var typeArguments = (Type[])testMethod.Arguments[0];
        var method = testMethod.MethodInfo.DeclaringType.GetMethods()
            .SingleOrDefault(m => m.Name.Equals(testMethod.MethodInfo.Name) &&
                                  m.GetGenericArguments().Length == typeArguments.Length) ??
            throw new ArgumentException($"Method '{testMethod.TestMethodName}`{
                typeArguments.Length}' not found.");

        return method.MakeGenericMethod(typeArguments);
    }
}
```

# Решение



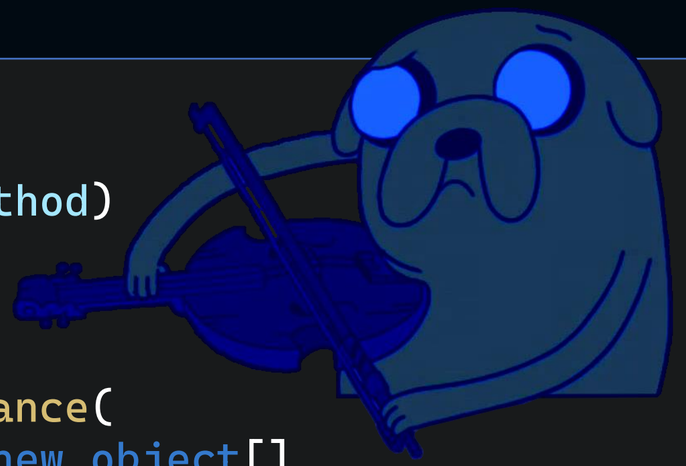
```
public class GTestMethodAttribute : DataTestMethodAttribute
{
    public override TestResult[] Execute(ITestMethod testMethod)
    {
        var type = testMethod.GetType();
        var helper = new ReflectionHelper(type);
        var genericTest = (ITestMethod)Activator.CreateInstance(
            type, ReflectionHelper.Flags, binder: default, new object[]
            {
                helper.MakeGenericMethod(testMethod),
                helper.GetProperty(testMethod, "Parent"),
                helper.GetProperty(testMethod, "TestMethodOptions")
            },
            culture: default);

        helper.InvokeMethod(
            genericTest,
            "SetArguments",
            new[] { typeof(object[]) },
            new object[] { testMethod.Arguments.Skip(1).ToArray() });

        return base.Execute(genericTest);
    }
}
```

# Решение



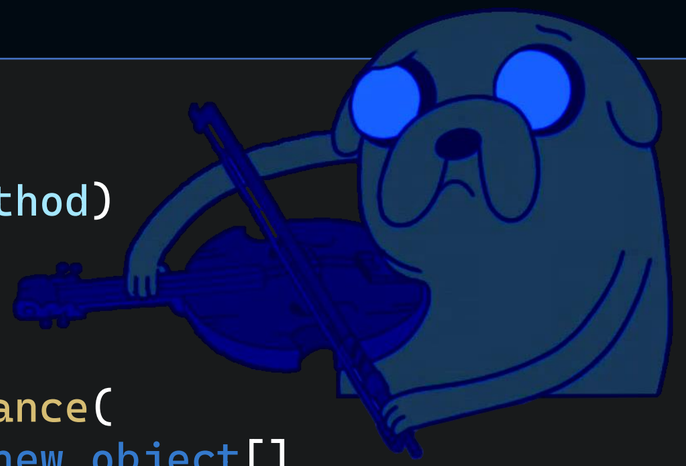
```
public class GTestMethodAttribute : DataTestMethodAttribute
{
    public override TestResult[] Execute(ITestMethod testMethod)
    {
        var type = testMethod.GetType();
        var helper = new ReflectionHelper(type);
        var genericTest = (ITestMethod)Activator.CreateInstance(
            type, ReflectionHelper.Flags, binder: default, new object[]
            {
                helper.MakeGenericMethod(testMethod),
                helper.GetProperty(testMethod, "Parent"),
                helper.GetProperty(testMethod, "TestMethodOptions")
            },
            culture: default);

        helper.InvokeMethod(
            genericTest,
            "SetArguments",
            new[] { typeof(object[]) },
            new object[] { testMethod.Arguments.Skip(1).ToArray() });

        return base.Execute(genericTest);
    }
}
```

# Решение



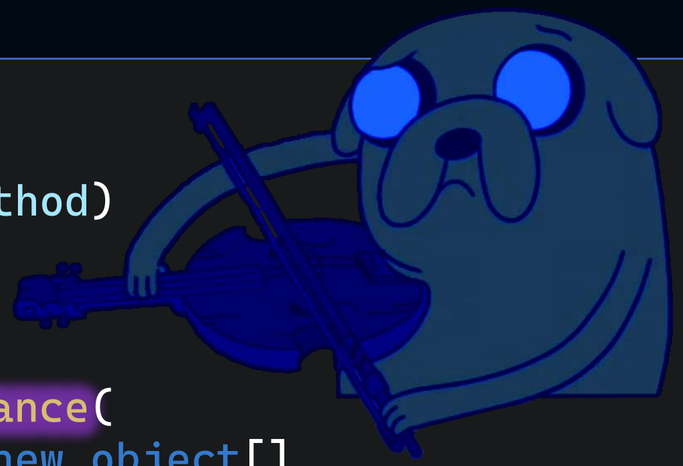
```
public class GTestMethodAttribute : DataTestMethodAttribute
{
    public override TestResult[] Execute(ITestMethod testMethod)
    {
        var type = testMethod.GetType();
        var helper = new ReflectionHelper(type);
        var genericTest = (ITestMethod)Activator.CreateInstance(
            type, ReflectionHelper.Flags, binder: default, new object[]
            {
                helper.MakeGenericMethod(testMethod),
                helper.GetProperty(testMethod, "Parent"),
                helper.GetProperty(testMethod, "TestMethodOptions")
            },
            culture: default);

        helper.InvokeMethod(
            genericTest,
            "SetArguments",
            new[] { typeof(object[]) },
            new object[] { testMethod.Arguments.Skip(1).ToArray() });

        return base.Execute(genericTest);
    }
}
```

# Решение



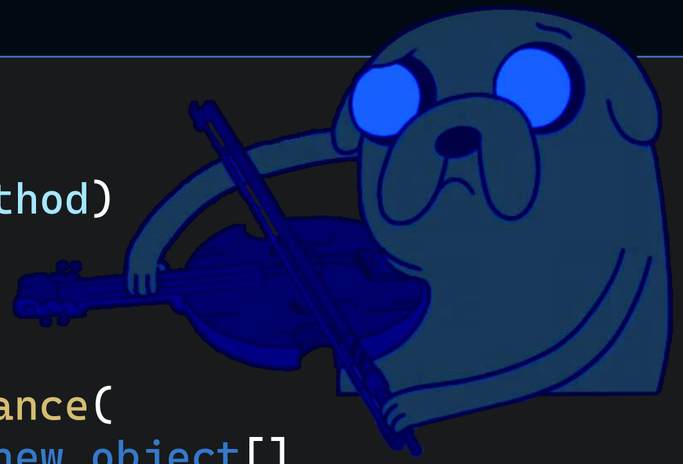
```
public class GTestMethodAttribute : DataTestMethodAttribute
{
    public override TestResult[] Execute(ITestMethod testMethod)
    {
        var type = testMethod.GetType();
        var helper = new ReflectionHelper(type);
        var genericTest = (ITestMethod)Activator.CreateInstance(
            type, ReflectionHelper.Flags, binder: default, new object[]
            {
                helper.MakeGenericMethod(testMethod),
                helper.GetProperty(testMethod, "Parent"),
                helper.GetProperty(testMethod, "TestMethodOptions")
            },
            culture: default);

        helper.InvokeMethod(
            genericTest,
            "SetArguments",
            new[] { typeof(object[]) },
            new object[] { testMethod.Arguments.Skip(1).ToArray() });

        return base.Execute(genericTest);
    }
}
```

# Решение



```
public class GTestMethodAttribute : DataTestMethodAttribute
{
    public override TestResult[] Execute(ITestMethod testMethod)
    {
        var type = testMethod.GetType();
        var helper = new ReflectionHelper(type);
        var genericTest = (ITestMethod)Activator.CreateInstance(
            type, ReflectionHelper.Flags, binder: default, new object[]
            {
                helper.MakeGenericMethod(testMethod),
                helper.GetProperty(testMethod, "Parent"),
                helper.GetProperty(testMethod, "TestMethodOptions")
            },
            culture: default);

        helper.InvokeMethod(
            genericTest,
            "SetArguments",
            new[] { typeof(object[]) },
            new object[] { testMethod.Arguments.Skip(1).ToArray() });

        return base.Execute(genericTest);
    }
}
```

# Решение



```
public class GTestMethodAttribute : DataTestMethodAttribute
{
    public override TestResult[] Execute(ITestMethod testMethod)
    {
        var type = testMethod.GetType();
        var helper = new ReflectionHelper(type);
        var genericTest = (ITestMethod)Activator.CreateInstance(
            type, ReflectionHelper.Flags, binder: default, new object[]
            {
                helper.MakeGenericMethod(testMethod),
                helper.GetProperty(testMethod, "Parent"),
                helper.GetProperty(testMethod, "TestMethodOptions")
            },
            culture: default);

        helper.InvokeMethod(
            genericTest,
            "SetArguments",
            new[] { typeof(object[]) },
            new object[] { testMethod.Arguments.Skip(1).ToArray() });

        return base.Execute(genericTest);
    }
}
```

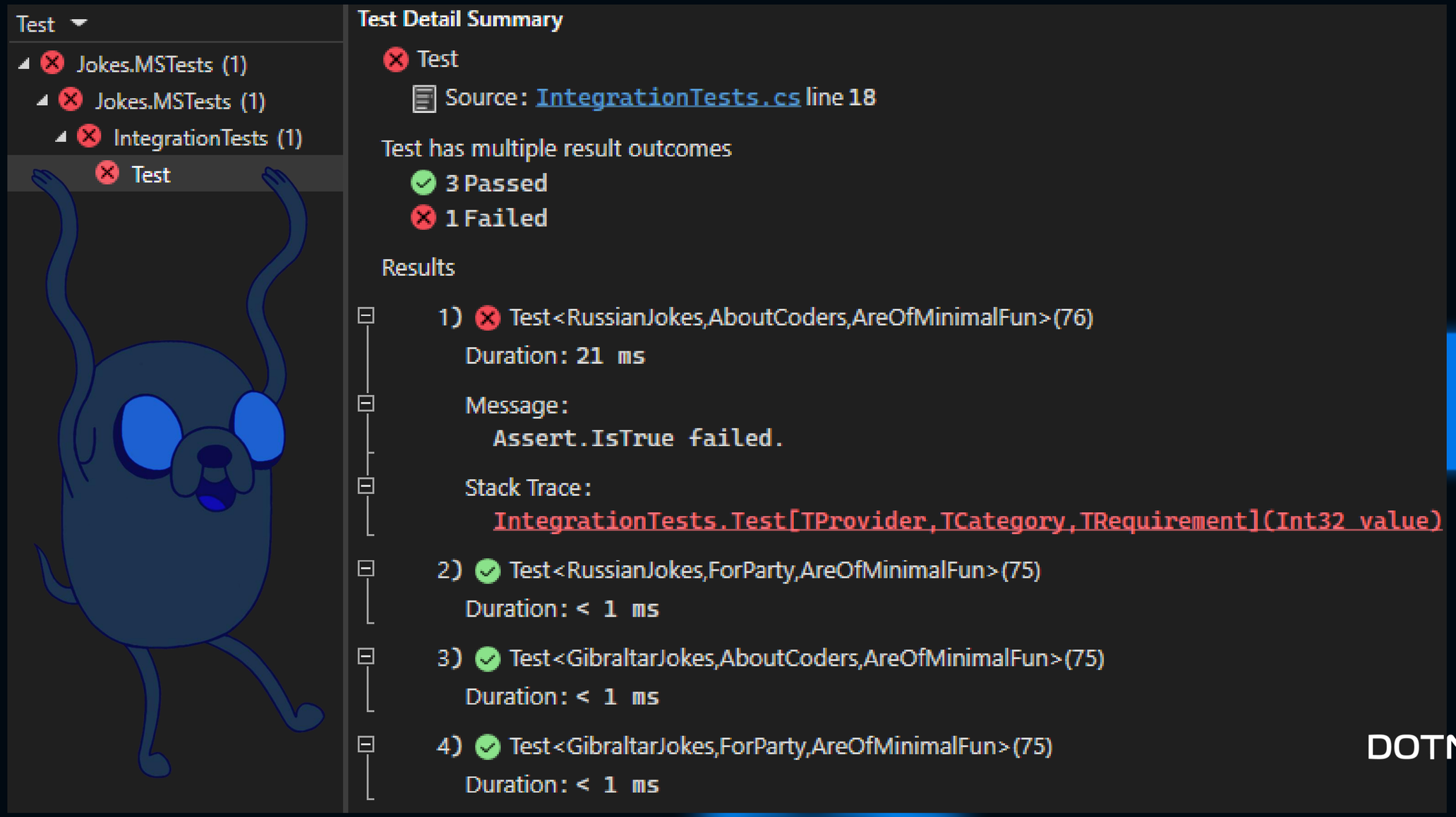
# Обобщенный тест

```
[GTestMethod]
[GDataRow<RussianJokes, AboutCoders, AreOfMinimalFun>(76)]
[GDataRow<RussianJokes, ForParty, AreOfMinimalFun>(75)]
// ...
public void Test() { }

public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

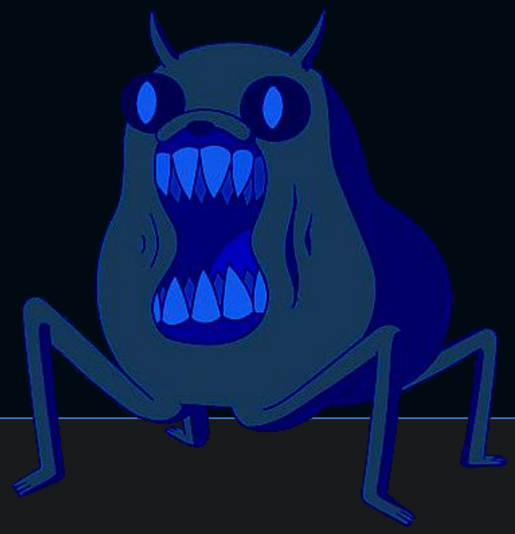
# Обобщенный тест

```
[GTestMethod]
[GDataRow<RussianJokes, AboutCoders, AreOfMinimalFun>(76)]
[GDataRow<RussianJokes, ForParty, AreOfMinimalFun>(75)]
// ...
public void Test() { }

public void Test<TProvider, TCategory, TRequirement>(int value)
    where TProvider : IJokeProvider, new()
    where TCategory : ICategory, new()
    where TRequirement : IRequirement, new()
{
    // Arrange
    IJokeProvider provider = new TProvider();
    IEnumerable<IJoke> jokes = provider.GetJokes();
    ICategory category = new TCategory();

    // Act
    IEnumerable<IScoredJoke> jokesOnCategory = category.GetTopJokes(jokes);

    // Assert
    IRequirement requirement = new TRequirement();
    bool actual = requirement.IsMet(jokesOnCategory, value);
    Assert.IsTrue(actual);
}
```

# Результат


Test
Jokes.MSTests (1)
Jokes.MSTests (1)
IntegrationTests (1)
Test
Test Detail Summary
Test
Source: IntegrationTests.cs line 18
Test has multiple result outcomes
3 Passed
1 Failed
Results
1) Test<RussianJokes,AboutCoders,AreOfMinimalFun>(76)
Duration: 21 ms
Message:
Assert.IsTrue failed.
Stack Trace:
IntegrationTests.Test[TProvider,TCategory,TRequirement](Int32 value)
2) Test<RussianJokes,ForParty,AreOfMinimalFun>(75)
Duration: < 1 ms
3) Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)
Duration: < 1 ms
4) Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)
Duration: < 1 ms

# Плохой тест



```
[GTestMethod]
// ...
[GDataRow<object, object, object>(75)]
public void Test() { }
```

# Плохой тест

Test
- Jokes.XUnitTests (4)
- Jokes.Tests (4)
- Jokes.MSTests (1)
  - Jokes.MSTests (1)
    - IntegrationTests (1)
      - Test

**Test Detail Summary**

Test

Source: IntegrationTests.cs line 19

Test has multiple result outcomes

3 Passed

2 Failed

Results

1) Test<RussianJokes,AboutCoders,AreOfMinimalFun>(76)

Duration: 22 ms

Message:

```
Assert.IsTrue failed.
```

Stack Trace:

IntegrationTests.Test[TProvider,TCategory,TRequirement](Int32 value) line 37

2) Test<RussianJokes,ForParty,AreOfMinimalFun>(75)

Duration: < 1 ms

3) Test<GibraltarJokes,AboutCoders,AreOfMinimalFun>(75)

Duration: < 1 ms

4) Test<GibraltarJokes,ForParty,AreOfMinimalFun>(75)

Duration: < 1 ms

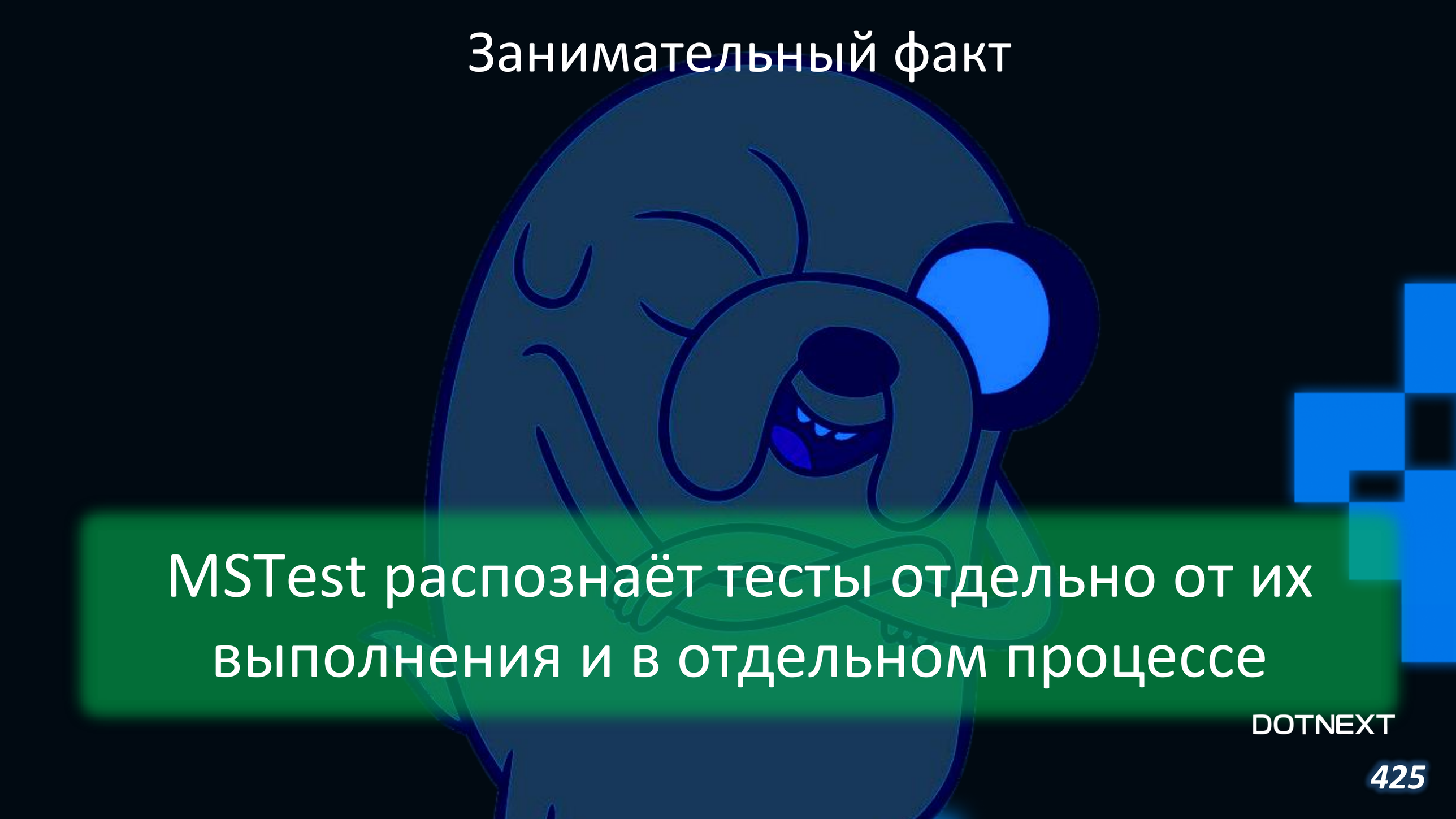
5) Test<Object,Object,Object>(75)

Duration: 2 ms

Message:

```
Exception thrown while executing test. If using extension of TestMethodAttribute then please contact vendor.
Error message: GenericArguments[0], 'System.Object', on 'Void Test[TProvider,TCategory,TRequirement](Int32)' violates
   at System.Reflection.RuntimeMethodInfo.MakeGenericMethod(Type[] methodInstantiation)
   at Jokes.MSTests.ReflectionHelper.MakeGenericMethod(ITestMethod testMethod) in D:\repos\dotNext\Jokes\Jokes.MSTests
   at Jokes.MSTests.GTestMethodAttribute.Execute(ITestMethod testMethod) in D:\repos\dotNext\Jokes\Jokes.MSTests\GT
   at Microsoft.VisualStudio.TestPlatform.MSTest.TestAdapter.Execution.TestMethodRunner.ExecuteTest(TestMethodInfo te
```

# Занимательный факт



MSTest распознаёт тесты отдельно от их выполнения и в отдельном процессе

# Check Point

- ✓ MSTest разделяет атрибуты-тесты и атрибуты-данные
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Расширение MSTest, по сравнению с остальными тестовыми, фреймворками значительно затруднено

# Check Point

- ✓ MSTest разделяет атрибуты-тесты и атрибуты-данные
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Расширение MSTest, по сравнению с остальными тестовыми, фреймворками значительно затруднено

# Check Point

- ✓ MSTest разделяет атрибуты-тесты и атрибуты-данные
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Расширение MSTest, по сравнению с остальными тестовыми, фреймворками значительно затруднено

# Check Point

- ✓ MSTest разделяет атрибуты-тесты и атрибуты-данные
- ✓ Обнаружение и выполнение происходят в разных процессах
- ✓ Расширение MSTest, по сравнению с остальными тестовыми, фреймворками значительно затруднено

# С MSTest - ВСЁ

# Вывод

Возможности тестовых фреймворков выходят далеко за рамки того, чем мы привыкли пользоваться в повседневной работе. Например, они из коробки предоставляют средства аспектно-ориентированного программирования, или с помощью некоторых дополнительных действий позволяют взаимодействовать с обобщёнными методами, и даже дают возможность создания новых пользовательских тестовых атрибутов для написания тестов с гибко настраиваемым поведением.

# Источники

- ✓ https://stackoverflow.com/questions/2364929/nunit-testcase-with-generics
- ✓ https://github.com/nunit/nunit
- ✓ https://stackoverflow.com/questions/39570641/xunit-theory-test-using-generics
- ✓ https://github.com/xunit/xunit
- ✓ https://github.com/microsoft/testfx
- ✓ https://github.com/icsharpcode/ILSpy
- ✓ https://github.com/Fody/Home/blob/master/pages/addins.md

# Спасибо за внимание

# Вопросы