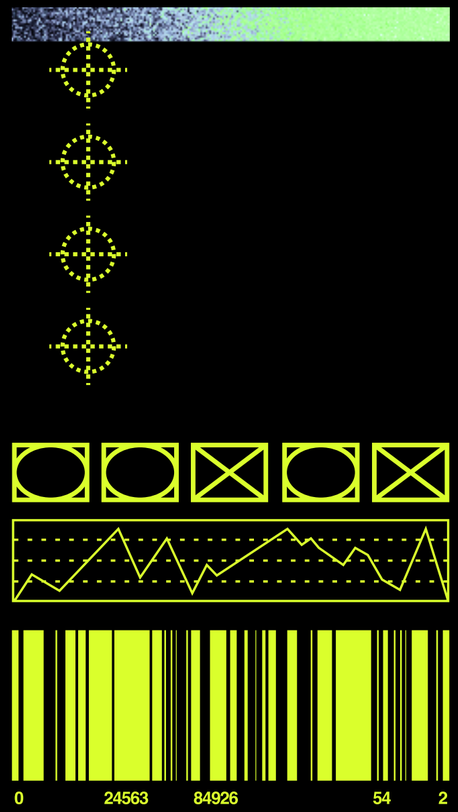
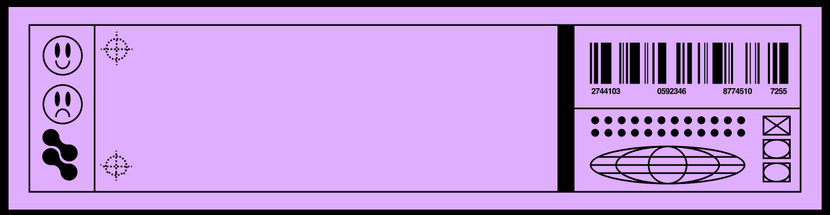
DI-КОНТЕЙНЕРЫ В NUNIT-ТЕСТАХ



Вадим Мартынов, инженер в Яндексе



ПРИВЕТ!
МЕНЯ ЗОВУТ ВАДИМ

СЕГОДНЯ ГОВОРИМ ПРО

- как обычно пишутся тесты
- что в этом плохого
- как избавиться от плохих практик
- как добавить хорошие

NUNIT. ОЖИДАНИЕ

What Is NUnit?

NUnit is a unit-testing framework for all .Net languages. Initially ported from JUnit, the current production release, version 3, has been completely rewritten with many new features and support for a wide range of .NET platforms.

NUNIT. ОЖИДАНИЕ

What Is NUnit?

NUnit is a unit-testing framework for all .Net languages. Initially ported from JUnit, the current production release, version 3, has been completely rewritten with many new features and support for a wide range of .NET platforms.

What is unit testing?

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually scrutinized for proper operation. Software developers and sometimes QA staff complete unit tests during the development process. The main objective of unit testing is to isolate written code to test and determine if it works as intended.

NUNIT. РЕАЛЬНОСТЬ

👌 Юнит-тесты

😐 Интеграционные тесты

🧐 Тесты на методы HTTP API

🫠 E2E-тесты

ПИШЕМ ТЕСТЫ

```
[TestFixture]
class FeatureOne
{
    [Test]
    public void TestObject_should_do_something()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        var testData = new TestData();
        var testObject = CreateEntity(session, testData);
        SomeAction(testObject);
        SomeAssert(testObject);
    }
}
```

ПИШЕМ ТЕСТЫ

```
[Test]
public void TestObject_should_do_something()
{
    var session = Authenticate(Constants.Login, Constants.Password);
    var testData = new TestData();
    var testObject = CreateEntity(session, testData);
    SomeAction(testObject);
    SomeAssert(testObject);
}

[Test]
public void TestObject_should_do_something()
{
    var session = Authenticate(Constants.Login, Constants.Password);
    var testData = new TestData();
    var testObject = CreateEntity(session, testData);
    AnotherAction(testObject);
    AnotherAssert(testObject);
}
```

ПИШЕМ ТЕСТЫ

```
[Test]
public void TestObject_should_do_something()
{
    var session = Authenticate(Constants.Login, Constants.Password);
    var testData = new TestData();
    var testObject = CreateEntity(session, testData);
    SomeAction(testObject);
    SomeAssert(testObject);
}

[Test]
public void TestObject_should_do_something()
{
    var session = Authenticate(Constants.Login, Constants.Password);
    var testData = new TestData();
    var testObject = CreateEntity(session, testData);
    AnotherAction(testObject);
    AnotherAssert(testObject);
}
```

ПИШЕМ ТЕСТЫ

```
TestObject testObject;
[SetUp]
public void AuthenticateAndPrepareData()
{
    var session = Authenticate(Constants.Login, Constants.Password);
    testObject = CreateEntity(session, new TestData());
}
[Test]
public void TestObject_should_do_something()
{
    SomeAction(testObject);
    SomeAssert(testObject);
}
[Test]
public void TestObject_should_do_something_more()
{
    AnotherAction(testObject);
    AnotherAssert(testObject);
}
```

ПИШЕМ ТЕСТЫ

```
...
[Test]
public void TestObject_should_do_something()
{
    SomeAction(testObject);
    SomeAssert(testObject);
}
[Test]
public void TestObject_should_do_something_more()
{
    AnotherAction(testObject);
    AnotherAssert(testObject);
}
[Test]
public void Invalid_TestObject_should_do_something_bad()
{
    testObject = CreateEntity(session, new BadTestData());
    SomeAction(testObject);
    AssertInvalid(testObject);
}
```

ПИШЕМ ТЕСТЫ

```
TestObject testObject;
TestObject invalidObject;
[SetUp]
public void AuthenticateAndPrepareData()
{
    var session = Authenticate(Constants.Login, Constants.Password);
    testObject = CreateEntity(session, new TestData());
    invalidObject = CreateEntity(session, new BadTestData());
}
[Test]
public void TestObject_should_do_something() { ... }
[Test]
public void TestObject_should_do_something_more() { ... }
[Test]
public void Invalid_TestObject_should_do_something_bad() { ... }
[Test]
public void Invalid_TestObject_should_do_something_bad_again() { ... }
```

ПИШЕМ ТЕСТЫ

```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

ПИШЕМ ТЕСТЫ

+ DRY

```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

ПИШЕМ ТЕСТЫ

+ DRY
+ EASY

```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

ПИШЕМ ТЕСТЫ

+ DRY
+ EASY
- 😳

```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

ПИШЕМ ТЕСТЫ

+ DRY
+ EASY
- 
- 

```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

ПИШЕМ ТЕСТЫ

+ DRY
+ EASY
- 
- 
- 

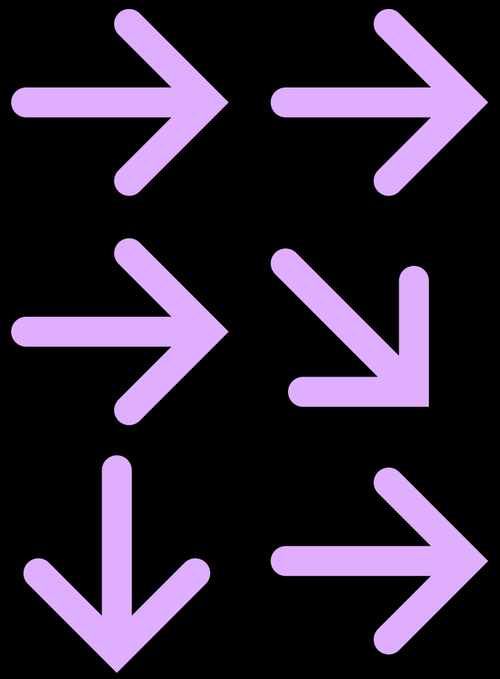
```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

ПИШЕМ ТЕСТЫ

+ DRY
+ EASY
- 
- 
- 
- 

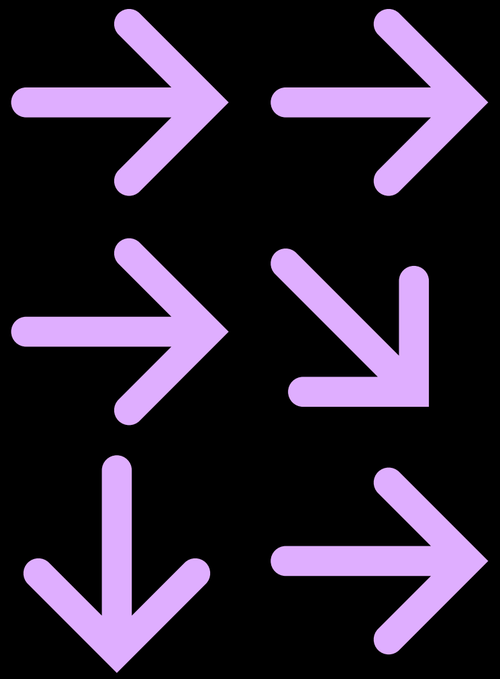
```
class BaseTest
{
    protected TestObject testObject;
    protected TestObject invalidObject;
    [SetUp]
    public void AuthenticateAndPrepareData()
    {
        var session = Authenticate(Constants.Login, Constants.Password);
        testObject = CreateEntity(session, new TestData());
        invalidObject = CreateEntity(session, new BadTestData());
    }
}
[TestFixture]
class FeatureOne : BaseTest
{
   [Test]
   public void TestObject_should_do_something() { ...}
}
[TestFixture]
class FeatureTwo : BaseTest { ... }
```

FIXTURE VS SUITE



```
[TestFixture]
class FeatureOne
```

FIXTURE VS SUITE



Test Fixtures

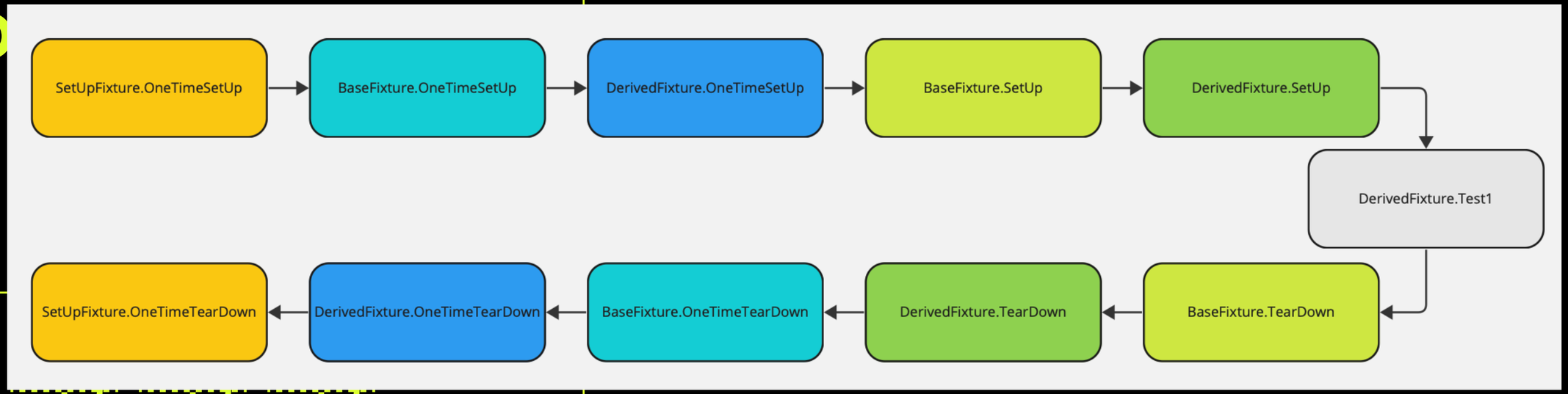
```
[TestFixture]
class FeatureOne
```

Test Fixtures

-
  - A test fixture is a fixed state of a set of objects used as a baseline for running tests.
- A test fixture is something used to consistently test some item, device, or piece of software.

ПОЛУЧАЕМ ТАКИЕ ПР В

Сложные и запутанные тесты из-за наследования и иерархии сетапов



ПОЛУЧАЕМ ТАКИЕ ПРОБЛЕМЫ В ТЕСТАХ

Сложные и запутанные тесты из-за наследования и иерархии сетапов

Распараллеливание тестов вызывает боль из-за избыточной связности или общего кода

ПОЛУЧАЕМ ТАКИЕ ПРОБЛЕМЫ В ТЕСТАХ

Сложные и запутанные тесты из-за наследования и иерархии сетапов

Распараллеливание тестов вызывает боль из-за избыточной связности или общего кода

Запуск отдельного теста вызывает длительное действие подготовки данных или провайдеров, которые этому отдельному тесту могут быть не нужны

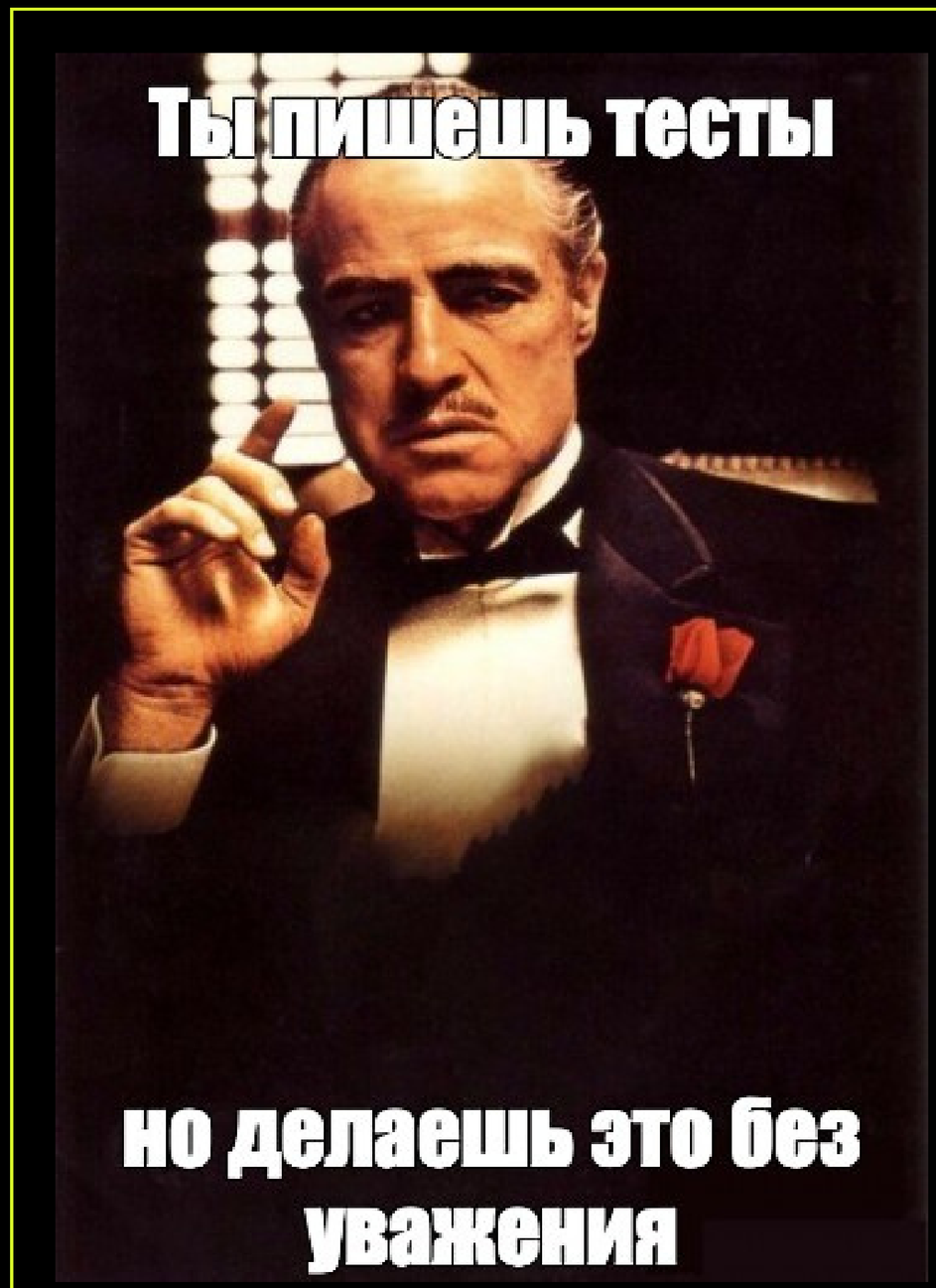
ПОЛУЧАЕМ ТАКИЕ ПРОБЛЕМЫ В ТЕСТАХ

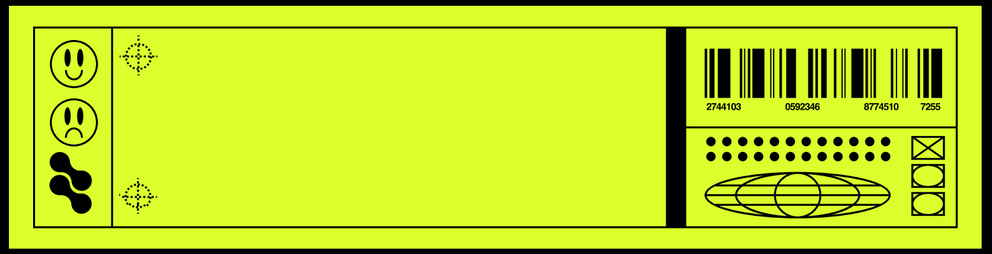
Сложные и запутанные тесты из-за наследования и иерархии сетапов

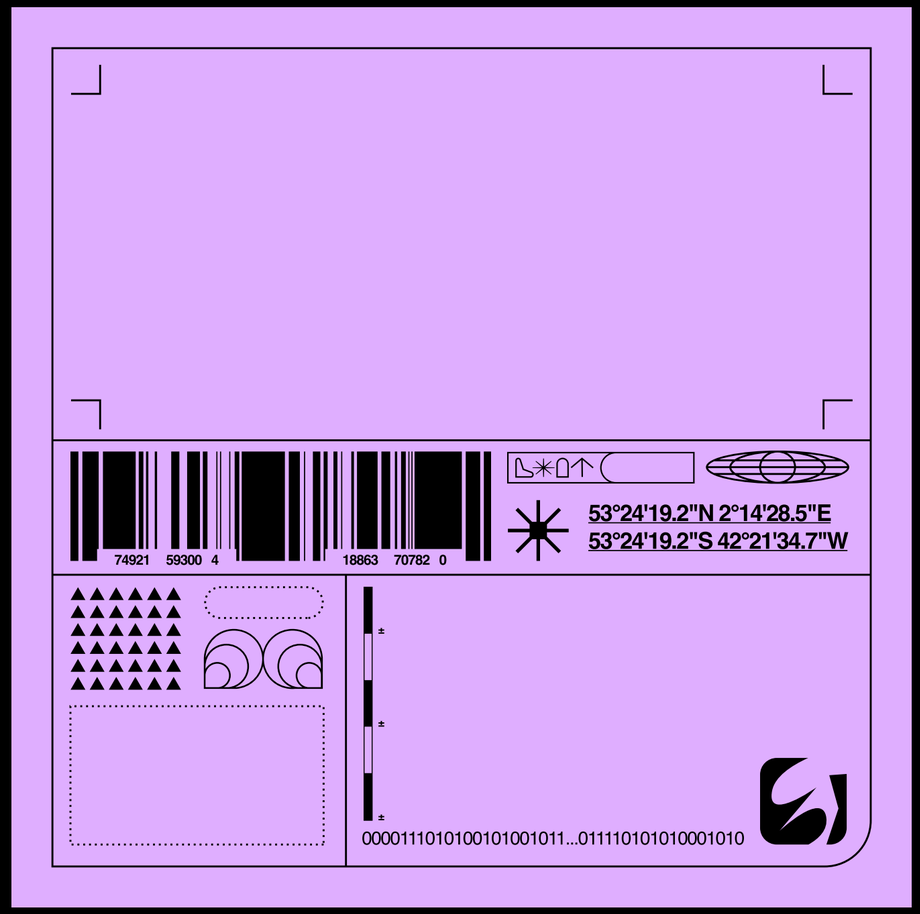
Распараллеливание тестов вызывает боль из-за избыточной связности или общего кода

Запуск отдельного теста вызывает длительное действие подготовки данных или провайдеров, которые этому отдельному тесту могут быть не нужны

Из-за базовых классов и наличия SetUp неясно, в каких условиях работают тесты, что им нужно, а что нет

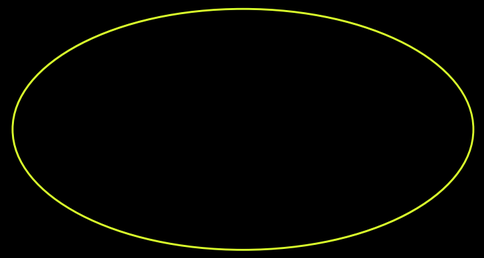

Ты пишешь тесты
но делаешь это без уважения

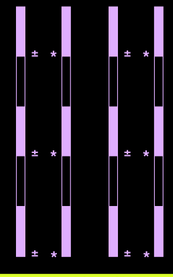




А как по-другому?

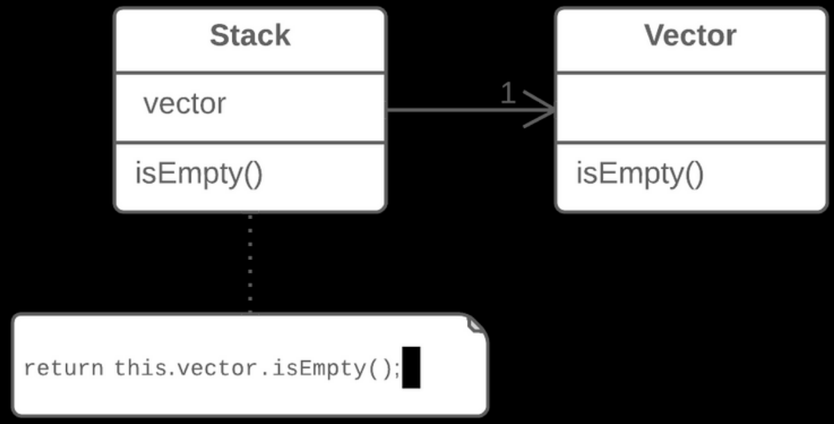
давайте исправлять

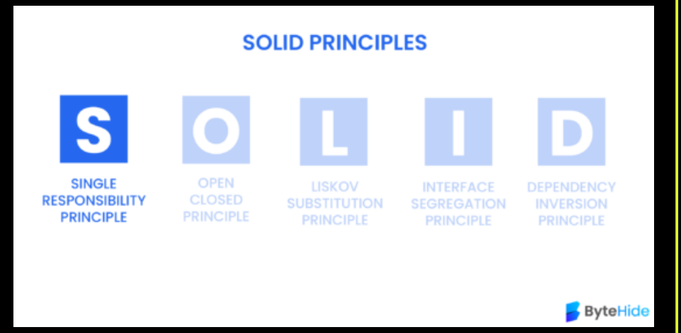


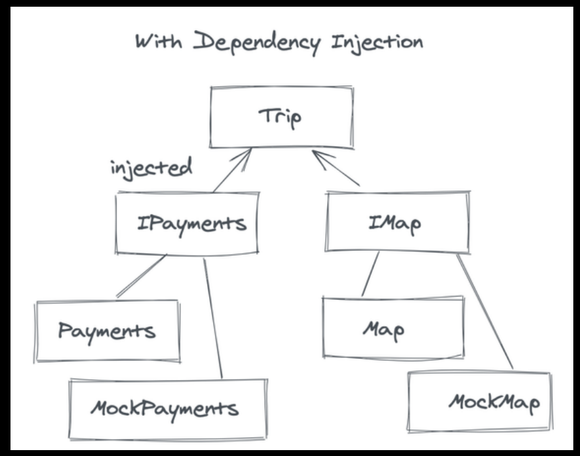


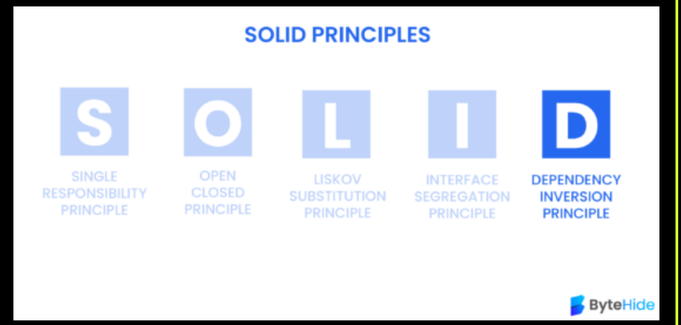
ДЕЛАЕМ ПО-ДРУГОМУ

~ Делегирование
~ SRP
~ DIP
~ Внедрение зависимостей









DI-КОНТЕЙНЕРЫ!

ОТКАЗЫВА-
ЕМСЯ ОТ
ПЛОХОГО

ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class UsersClient
{
    private User user;
    public void PrepareUser()
        => this.user = Deserialize(SentHttpPostRequest("/users"));

    public User GetUser() => this.user;
}

var usersClient = new UsersClient();
usersClient.PrepareUser();
var user1 = portalClient.GetUser();
usersClient.PrepareUser();
var user2 = portalClient.GetUser();
```

ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class UsersClient
{
    private User user;
    public void PrepareUser()
        => this.user = Deserialize(SentHttpPostRequest("/users"));

    public User GetUser() => this.user;
}

var usersClient = new UsersClient();
usersClient.PrepareUser();
var user1 = portalClient.GetUser();
usersClient.PrepareUser();
var user2 = portalClient.GetUser();
```

ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class UsersClient
{
    private User user;
    public void PrepareUser()
        => this.user = Deserialize(SentHttpPostRequest("/users"));

    public User GetUser() => this.user;
}

var usersClient = new UsersClient();
usersClient.PrepareUser();
var user1 = portalClient.GetUser();
usersClient.PrepareUser();
var user2 = portalClient.GetUser();
```

ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class Fixture
{
    private User user;
    private Guid orgId;
    [SetUp]
    public void Prepare()
    {
        this.user = new PortalClient().CreateUser();
        this.orgId = Guid.NewGuid();
    }
    [Test]
    public void Test1()
    {
        var result = Act1(this.user, this.org);
        result.Should().BeCorrect();
    }
    [Test]
    public void Test2()
    {
        var result = Act2(this.user);
        result.Should().BeCorrect();
    }
}
```

ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class Fixture
{
    private (User, Guid) Prepare() // Убрали SetUp, сделали возвращаемое значение
        => (new PortalClient().CreateUser(), Guid.NewGuid());

    [Test]
    public void Test1()
    {
        var (user, orgId) = this.Prepare(); // Вызываем метод подготовки явно
        var result = Act1(user, orgId);
        result.Should().BeCorrect();
    }
    [Test]
    public void Test2()
    {
        var (user, orgId) = this.Prepare(); // Вызываем метод подготовки явно
        var result = Act2(user);
        result.Should().BeCorrect();
    }
}
```

ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class Fixture
{
    private User PrepareUser() => new PortalClient().CreateUser();
    private (User, Guid) Prepare() => (this.PrepareUser(), Guid.NewGuid());

    [Test]
    public void Test1()
    {
        var (user, orgId) = this.Prepare(); // Продолжаем использовать старый метод
        var result = Act1(user, orgId);
        result.Should().BeCorrect();
    }
    [Test]
    public void Test2()
    {
        var user = this.PrepareUser(); // Заиспользовали новый метод
        var result = Act2(user);
        result.Should().BeCorrect();
    }
}
```

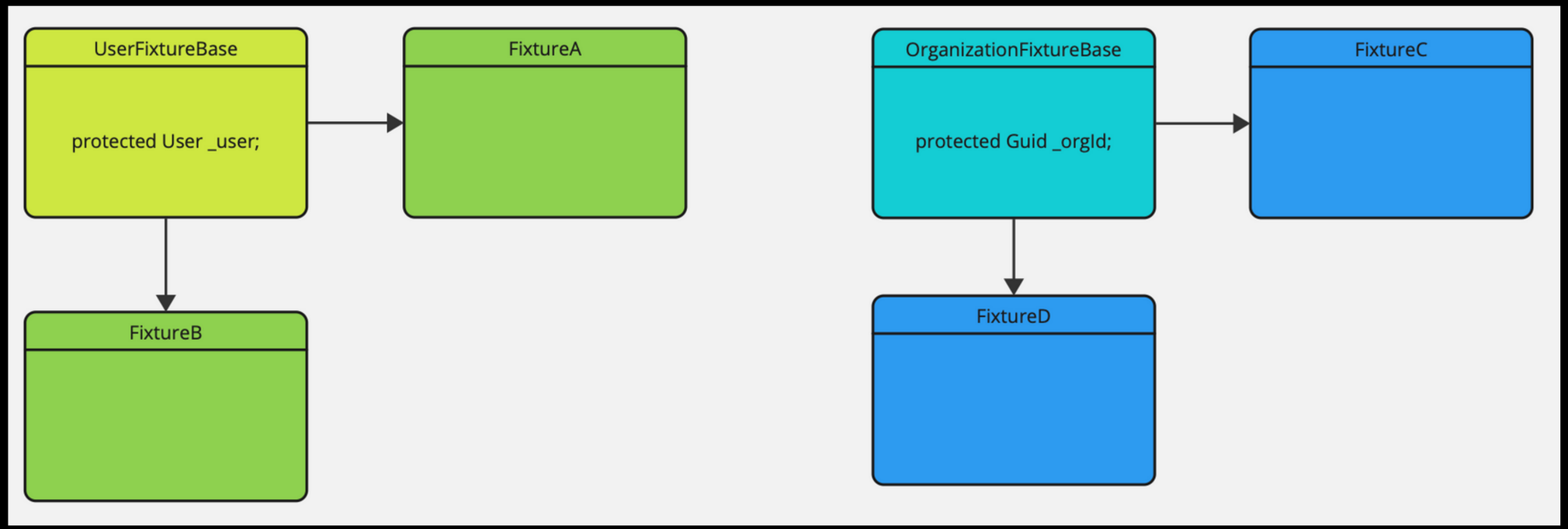
ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Атрибут SetUp

```
class Fixture
{
    private readonly ConcurrentBag<User> createdUsers  = new();

    [OneTimeTearDown]
    public void DeleteAllCreatedUsers() // Чистка всех созданных пользовтелей
    {
        foreach(var user in this.createdUsers)
            new PortalClient().DeleteUser(user);
    }

    private User PrepareUser()
    {
        var user = new PortalClient().CreateUser();
        this.createdUsers.Add(user);  // сохраняем пользователя для дальнейшего удаления
        return user;
    }

    private (User, Guid) Prepare() => (this.PrepareUser(), Guid.NewGuid());

    // тут идут тесты
 }
```

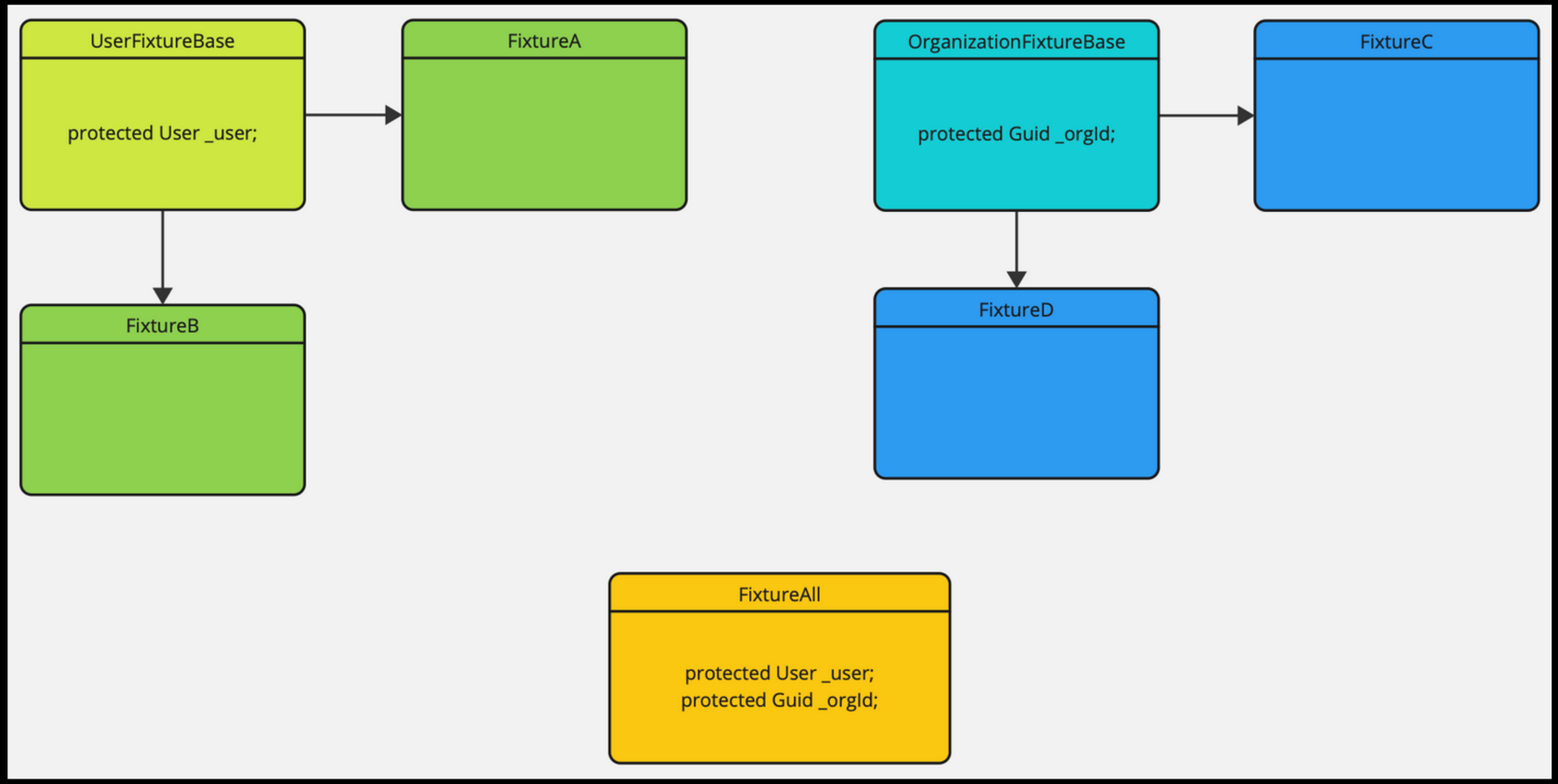
ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Наследование
TestFixture



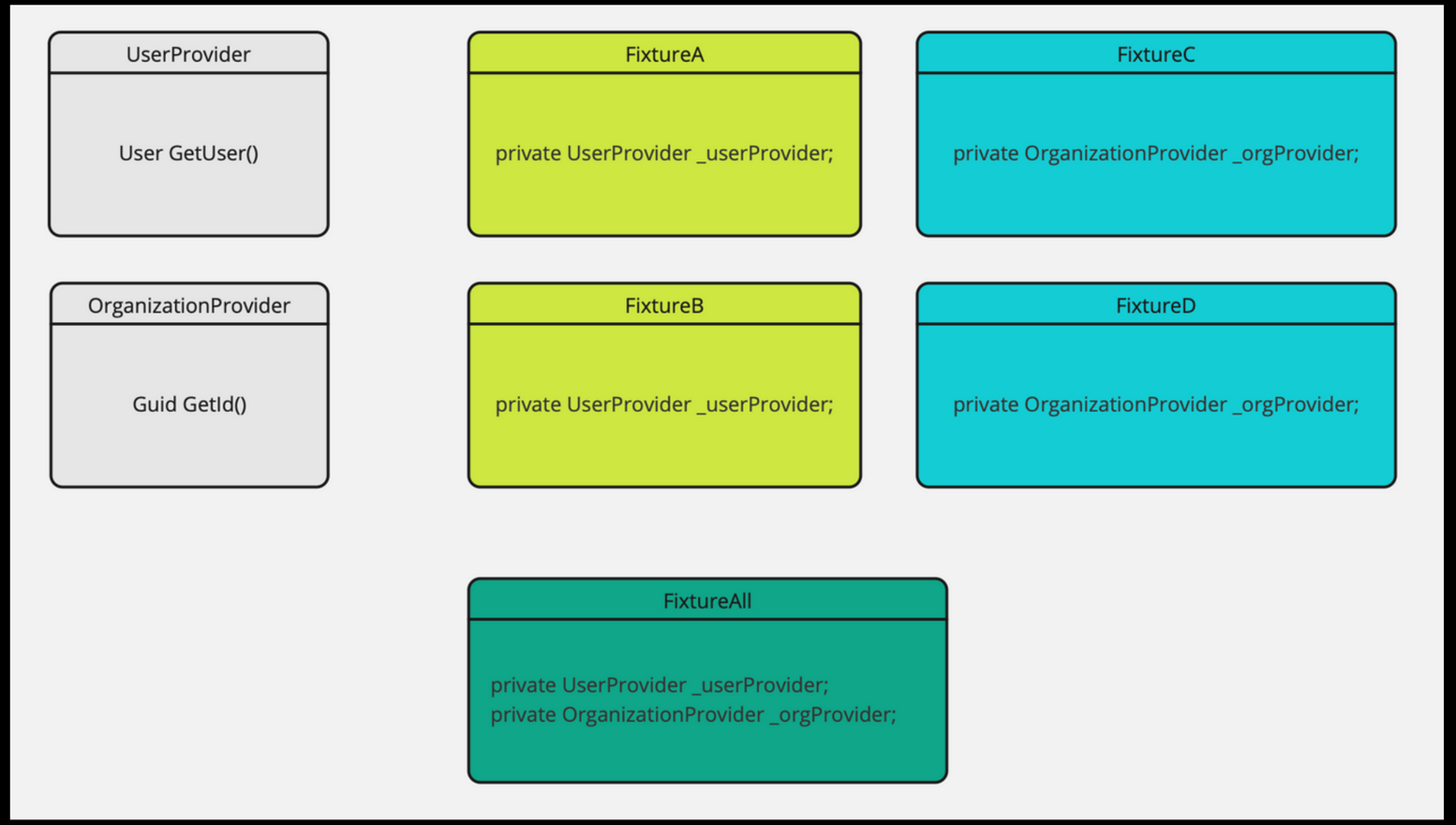
ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Наследование
TestFixture



ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Наследование TestFixture



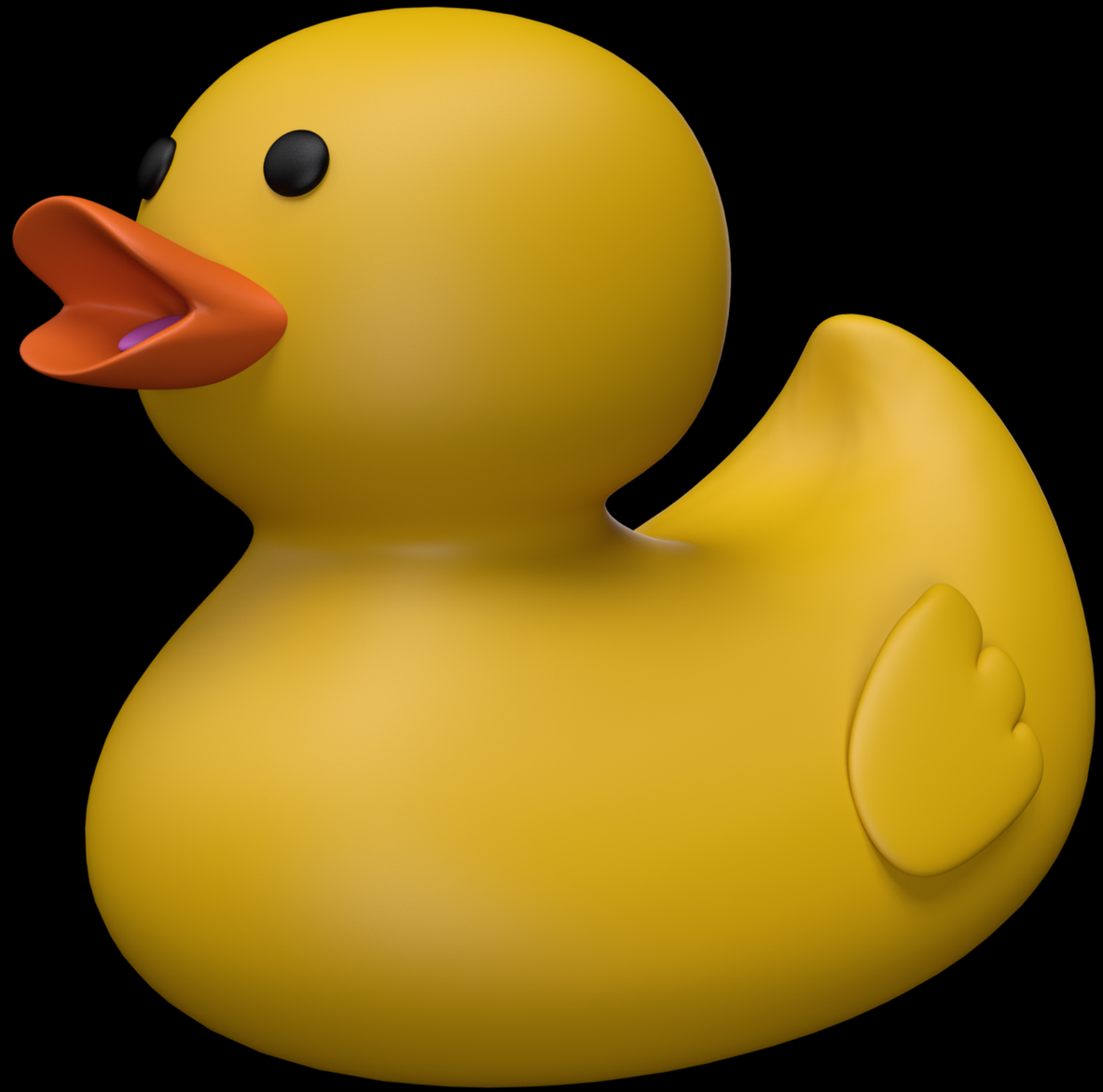
ОТКАЗЫВАЕМСЯ ОТ ПЛОХОГО

Наследование

TestFixture

```
class UserProvider { ... }
class Fixture1 { private UserProvider = new(); ... }
class Fixture2 { private UserProvider = new(); ...}
class OrganizationProvider { ... }
class FixtureA { private OrganizationProvider = new(); ...}
class FixtureB { private OrganizationProvider = new(); ...}

class FixtureAll
{
    private UserProvider = new();
    private OrganizationProvider = new();
    ....
}
```

ДОБАВЛЯЕМ ХОРОШЕЕ

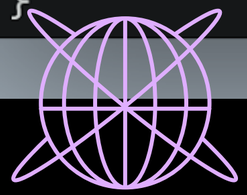


DI-контейнер

```
class Fixture
{
    private readonly ServiceProvider serviceProvider;
    public Fixture()
    {
        var serviceCollection = new ServiceCollection()
            .AddSingleton<ILog, NUnitLog>() // регистрируем зависимости в контейнере
            .AddSingleton<IPortalClient, PortalClient>() // регистрируем зависимости в контейнере
            .AddSingleton<UserPreparer>() // регистрируем провайдеры в контейнере
            .AddSingleton<DataPreparer>(); // регистрируем провайдеры в контейнере
       this.serviceProvider = serviceCollection.BuildServiceProvider(new() { ValidateOnBuild = true });
    }
    [OneTimeTearDown]
    public void DeleteAllCreatedUsers() => this.serviceProvider.Resolve<UserPreparer>().DeleteAllCreatedUsers();
    [Test]
    public void Test1()
    {
        var (user, orgId) = this.serviceProvider.Resolve<DataPreparer>().Get();
        var result = Act1(user);
        result.Should().BeCorrect();
    }
    // Другие тесты
}
```

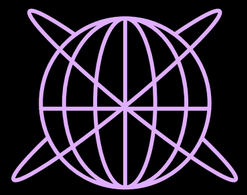
DI-контейнер

```
static class Extensions
{
    // Портальный клиент много где нужен, сделали переиспользуемый метод
    public static IServiceCollection RegisterPortalClient(this IServiceCollection collection)
        => collection
            .AddSingleton<ILog, NUnitLog>()
            .AddSingleton<IPortalClient, PortalClient>();
}

// Класс собирает DI-контейнер с нужными настройками
class ServiceProviderBuilder
{
    public ServiceProvider Build(Action<IServiceCollection> configureServices)
    {
        var serviceCollection = new ServiceCollection();
        configureServices(serviceCollection);
        return serviceCollection.BuildServiceProvider(new() { ValidateOnBuild = true });
    }
}
```



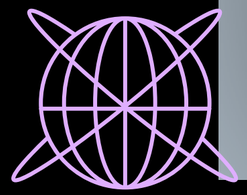
DI-контейнер

```
class Fixture
{
    private readonly ServiceProvider serviceProvider;
    public Fixture() // Создаем DI-контейнер с помощью новой фабрики
        => this.serviceProvider = new ServiceProviderBuilder.Build(collection
            => collection
                .RegisterPortalClient() // Регистрируем зависимости, необходимые конкретной TestFixture
                .AddSingleton<UserPreparer>()
                .AddSingleton<DataPreparer>());

    [OneTimeTearDown]
    public void DeleteAllCreatedUsers()
        => this.serviceProvider.Resolve<UserPreparer>().DeleteAllCreatedUsers();

    // тут идут тесты
}
```



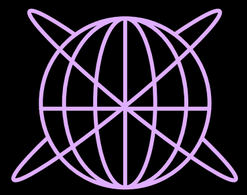
Очистка данных

```
// Провайдер, который умеет чистить все созданные им данные
class UserPreparer(IPortalClient portalClient) : IDisposable
{
    private readonly ConcurrentBag<User> createdUsers = new();

    public void Dispose()
    {
        foreach(var user in this.createdUsers)
            this.portalClient.DeleteUser(user);
    }

    public User Get()
    {
        var user = new this.portalClient.CreateUser();
        this.createdUsers.Add(user);
        return user;
    }
}
```



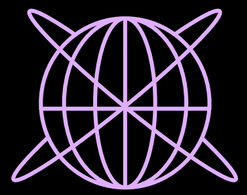
Очистка данных

```
[OneTimeTearDown]
public async Task DisposeContainer()
{
    // Универсальная чистилка
    // Очистит пользователей в UserPreparer,
    // если экземпляр этого класса-сервиса был получен из контейнера
    await this.serviceProvider.DisposeAsync();
}
```



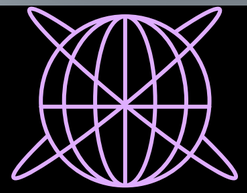
Очистка данных

```
public abstract class AbstractCleaner<TEntityKey> : ICleaner<TEntityKey>
{
    private ConcurrentBag<TEntityKey> savedEntityKeys = new();
    public event Action<ThrownExceptionEventArgs> ExceptionThrown = args => { };

    public void AddEntity(TEntityKey entityKey) => savedEntityKeys.Add(entityKey);

    public virtual async ValueTask DisposeAsync()
    {
        foreach (var entityKey in savedEntityKeys)
            await SafeDeleteAsync(entityKey).ConfigureAwait(false);

        savedEntityKeys = new();
    }

    protected abstract Task DeleteEntityAsync(TEntityKey entityKey);

    private async Task SafeDeleteAsync(TEntityKey entityKey)
    {
        try
        {
            await DeleteEntityAsync(entityKey).ConfigureAwait(false);
        }
        catch (Exception ex)
        {
            ExceptionThrown?.Invoke(new(ex));
        }
    }
}
```



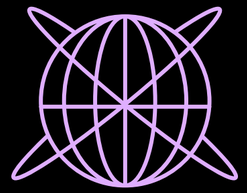
Общая инфраструктура

```
abstract class FixtureBase
{
    private readonly ServiceProvider serviceProvider;

    protected FixtureBase(Action<IServiceCollection> configureServices)
        => this.serviceProvider = new ServiceProviderBuilder().Build(configureServices);

    [OneTimeTearDown]
    public async Task DisposeContainer()
      => await this.serviceProvider.DisposeAsync();

    // Метод, позволяет получать зависимости Fixture
    protected TService Resolve<TService>()
      => this.serviceProvider.Resolve<TService>();
}
```



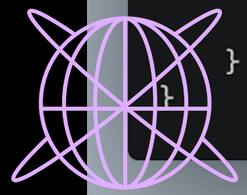
Общая инфраструктура

```
class Fixture : FixtureBase
{
    public Fixture() : base(collection ⇒ collection
        .RegisterPortalClient()
        .AddSingleton<UserPreparer>()
        .AddSingleton<DataPreparer>()) {}

    [Test]
    public void Test1()
    {
        var (user, orgId) = this.Resolve<DataPreparer>().Get(); // Резолв немного упростился
        var result = Act3(user, org);
        result.Should().BeCorrect();
    }
}
```



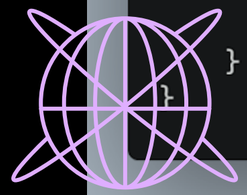
Instance per test

```
abstract class FixtureBase
{
    // Идентификатор текущего запущенного теста
    private static string CurrentTestId ⇒ TestContext.CurrentContext.Test.ID;

    // Тесты работают одновременно. Поэтому используем потокобезопасное хранилище скоупов
    private readonly ConcurrentDictionary<string, AsyncServiceScope> serviceScopes = new();
    private readonly ServiceProvider serviceProvider;

    protected FixtureBase(Action<IServiceCollection> configureServices)
        ⇒ this.serviceProvider = new ServiceProviderBuilder.Build(configureServices);

    [OneTimeTearDown]
    public async Task DisposeContainer() ⇒ await this.serviceProvider.DisposeAsync();

    [TearDown]
    public async Task RemoveScope()
    {
        // Скоупы тоже стоит диспозить. Например, завершить работу с `WebDriver` сразу после окончания теста
        if (this.serviceScopes.TryRemove(CurrentTestId, out var serviceScope))
            await serviceScope.DisposeAsync();
    }

    protected TServices Resolve<TServices>()
        ⇒ this.serviceScopes
            .GetOrAdd(CurrentTestId, _ ⇒ this.serviceProvider.CreateAsyncScope())
            .ServiceProvider
            .GetRequiredService<TServices>();
}
```



Instance per test

```
abstract class FixtureBase
{
    // Идентификатор текущего запущенного теста
    private static string CurrentTestId ⇒ TestContext.CurrentContext.Test.ID;

    // Тесты работают одновременно. Поэтому используем потокобезопасное хранилище скоупов
    private readonly ConcurrentDictionary<string, AsyncServiceScope> serviceScopes = new();
    private readonly ServiceProvider serviceProvider;

    protected FixtureBase(Action<IServiceCollection> configureServices)
        ⇒ this.serviceProvider = new ServiceProviderBuilder.Build(configureServices);

    [OneTimeTearDown]
    public async Task DisposeContainer() ⇒ await this.serviceProvider.DisposeAsync();

    [TearDown]
    public async Task RemoveScope()
    {
        // Скоупы тоже стоит диспозить. Например, завершить работу с `WebDriver` сразу после окончания теста
        if (this.serviceScopes.TryRemove(CurrentTestId, out var serviceScope))
            await serviceScope.DisposeAsync();
    }

    protected TServices Resolve<TServices>()
        ⇒ this.serviceScopes
            .GetOrAdd(CurrentTestId, _ ⇒ this.serviceProvider.CreateAsyncScope())
            .ServiceProvider
            .GetRequiredService<TServices>();
}
```

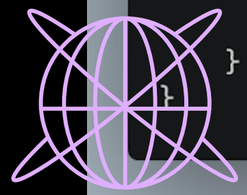


Instance per test

```
abstract class FixtureBase
{
    // Идентификатор текущего запущенного теста
    private static string CurrentTestId ⇒ TestContext.CurrentContext.Test.ID;

    // Тесты работают одновременно. Поэтому используем потокобезопасное хранилище скоупов
    private readonly ConcurrentDictionary<string, AsyncServiceScope> serviceScopes = new();
    private readonly ServiceProvider serviceProvider;

    protected FixtureBase(Action<IServiceCollection> configureServices)
        ⇒ this.serviceProvider = new ServiceProviderBuilder.Build(configureServices);

    [OneTimeTearDown]
    public async Task DisposeContainer() ⇒ await this.serviceProvider.DisposeAsync();

    [TearDown]
    public async Task RemoveScope()
    {
        // Скоупы тоже стоит диспозить. Например, завершить работу с `WebDriver` сразу после окончания теста
        if (this.serviceScopes.TryRemove(CurrentTestId, out var serviceScope))
            await serviceScope.DisposeAsync();
    }

    protected TServices Resolve<TServices>()
        ⇒ this.serviceScopes
            .GetOrAdd(CurrentTestId, _ ⇒ this.serviceProvider.CreateAsyncScope())
            .ServiceProvider
            .GetRequiredService<TServices>();
}
```

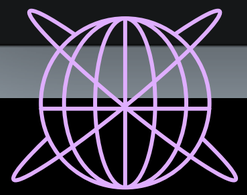
Instance per test

```
abstract class FixtureBase
{
    // Идентификатор текущего запущенного теста
    private static string CurrentTestId ⇒ TestContext.CurrentContext.Test.ID;

    // Тесты работают одновременно. Поэтому используем потокобезопасное хранилище скоупов
    private readonly ConcurrentDictionary<string, AsyncServiceScope> serviceScopes = new();
    private readonly ServiceProvider serviceProvider;

    protected FixtureBase(Action<IServiceCollection> configureServices)
        ⇒ this.serviceProvider = new ServiceProviderBuilder.Build(configureServices);

    [OneTimeTearDown]
    public async Task DisposeContainer() ⇒ await this.serviceProvider.DisposeAsync();

    [TearDown]
    public async Task RemoveScope()
    {
        // Скоупы тоже стоит диспозить. Например, завершить работу с `WebDriver` сразу после окончания теста
        if (this.serviceScopes.TryRemove(CurrentTestId, out var serviceScope))
            await serviceScope.DisposeAsync();
    }

    protected TServices Resolve<TServices>()
        ⇒ this.serviceScopes
            .GetOrAdd(CurrentTestId, _ ⇒ this.serviceProvider.CreateAsyncScope())
            .ServiceProvider
            .GetRequiredService<TServices>();
}
```



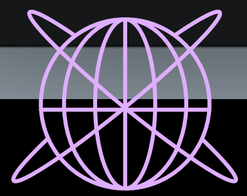
DI в поля класса для singleton

```
private readonly IReadOnlyCollection<TestDependencyInfo> testDependencies;

protected FixtureBase(Action<IServiceCollection> configureServices)
{
    var serviceCollection = new ServiceCollection();
    configureServices(serviceCollection);

    // Получаем объявленные в TestFixture-наследнике зависимости
    this.testDependencies = this.GetTestDependencies().ToArray();

    // Регистрируем объявленные зависимости в контейнере
    this.testDependencies.RegisterDependencies(serviceCollection.TryAddSingleton);

    this.serviceProvider = serviceCollection.BuildServiceProvider(new()
                                        { ValidateOnBuild = true, ValidateScopes = true });
  }

  [OneTimeSetUp]
  public void InitializeDependencies() // Резолвим зависимости из контейнера и присваиваем их в поля TestFixture
    => this.testDependencies.InitializeDependencies(serviceProvider.GetRequiredService);
```



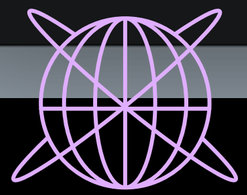
DI в поля класса для singleton

```
private readonly IReadOnlyCollection<TestDependencyInfo> testDependencies;

protected FixtureBase(Action<IServiceCollection> configureServices)
{
    var serviceCollection = new ServiceCollection();
    configureServices(serviceCollection);

    // Получаем объявленные в TestFixture-наследнике зависимости
    this.testDependencies = this.GetTestDependencies().ToArray();

    // Регистрируем объявленные зависимости в контейнере
    this.testDependencies.RegisterDependencies(serviceCollection.TryAddSingleton);

    this.serviceProvider = serviceCollection.BuildServiceProvider(new()
                                        { ValidateOnBuild = true, ValidateScopes = true });
}

[OneTimeSetUp]
public void InitializeDependencies() // Резолвим зависимости из контейнера и присваиваем их в поля TestFixture
  => this.testDependencies.InitializeDependencies(serviceProvider.GetRequiredService);
```



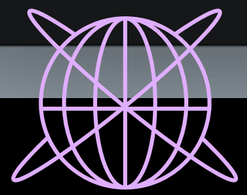
DI в поля класса для singleton

```
private readonly IReadOnlyCollection<TestDependencyInfo> testDependencies;

protected FixtureBase(Action<IServiceCollection> configureServices)
{
    var serviceCollection = new ServiceCollection();
    configureServices(serviceCollection);

    // Получаем объявленные в TestFixture-наследнике зависимости
    this.testDependencies = this.GetTestDependencies().ToArray();

    // Регистрируем объявленные зависимости в контейнере
    this.testDependencies.RegisterDependencies(serviceCollection.TryAddSingleton);

    this.serviceProvider = serviceCollection.BuildServiceProvider(new()
                                        { ValidateOnBuild = true, ValidateScopes = true });
}

[OneTimeSetUp]
public void InitializeDependencies() // Резолвим зависимости из контейнера и присваиваем их в поля TestFixture
  => this.testDependencies.InitializeDependencies(serviceProvider.GetRequiredService);
```



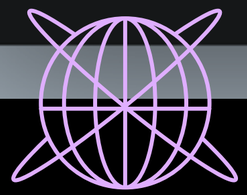
DI в поля класса для singleton

```
private readonly IReadOnlyCollection<TestDependencyInfo> testDependencies;

protected FixtureBase(Action<IServiceCollection> configureServices)
{
    var serviceCollection = new ServiceCollection();
    configureServices(serviceCollection);

    // Получаем объявленные в TestFixture-наследнике зависимости
    this.testDependencies = this.GetTestDependencies().ToArray();

    // Регистрируем объявленные зависимости в контейнере
    this.testDependencies.RegisterDependencies(serviceCollection.TryAddSingleton);

    this.serviceProvider = serviceCollection.BuildServiceProvider(new()
                                        { ValidateOnBuild = true, ValidateScopes = true });
}

[OneTimeSetUp]
public void InitializeDependencies() // Резолвим зависимости из контейнера и присваиваем их в поля TestFixture
    => this.testDependencies.InitializeDependencies(serviceProvider.GetRequiredService);
```



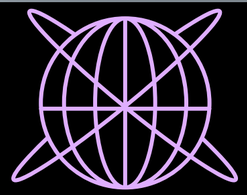
DI в поля класса для singleton

```
class Fixture : FixtureBase
{
    // Заявляем, что в TestFixture есть Singleton зависимость и её нужно автоматически зарегистрировать
    [TestDependency(TestDependencyConfig.ShouldRegister)]
    private readonly UserPreparer userPreparer;

    public Fixture()
        : base(collection ⇒ collection
            .RegisterPortalClient()  // общие зависимости регистрируем как Singleton
            .AddScoped<DataPreparer>()) {} // зависимость одного теста регистрируем как Scoped

    [Test]
    public void Test1()
    {
        var user = this.userPreparer.Get(); // Используем поле TestFixture
        var result = Act1(user);
        result.Should().BeCorrect();
    }
}
```



Начинаем использовать общие фичи

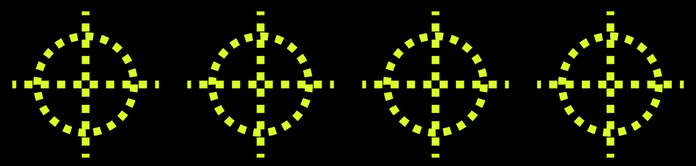
```
// Общий интерфейс для запускаемых фич внутри теста
interface ISetUp
{
    // Метод синхронный чтобы работать с AsyncLocal-контекстом теста, а не его копией
    void SetUp();
}

// Реализация трассировок для теста
class SpanFeature(ITracer tracer) : ISetUp, IDisposable
{
    private readonly ITracer tracer;
    private ISpanBuilder? span;
    public void SetUp() => this.span = this.tracer.BeginNewTrace(); // Начинаем трассировку
    public void Dispose() => this.span?.Dispose();                   // Завершаем трассировку
}
```



Начинаем использовать общие фичи

```
// Все тесты внутри всех TestFixture могут быть параллельными
[Parallelizable(ParallelScope.Children)]
internal abstract class FixtureBase
{
    [SetUp]
    public void RunSetUp()
    {
        foreach (var setUp in this.GetScope().GetServices<ISetUp>())
            setUp.SetUp();
    }

    [TearDown]
    public async Task RemoveScope()
    {
        if (this.serviceScopes.TryRemove(CurrentTestId, out var serviceScope))
            await serviceScope.DisposeAsync();
    }
    // остальная инфра FixtureBase
}
```

НО ВСЕ ЕЩЕ МОЖНО ВЫСТРЕЛИТЬ СЕБЕ В НОГУ



Продолжать использовать базовые классы в TestFixture, провайдерах данных или контроллах, даже если у вас есть DI-контейнер

НО ВСЕ ЕЩЕ МОЖНО ВЫСТРЕЛИТЬ СЕБЕ В НОГУ

Продолжать использовать базовые классы в TestFixture, провайдерах данных или контроллах, даже если у вас есть DI-контейнер

Не использовать Lazy для долгих операций в конструкторах

НО ВСЕ ЕЩЕ МОЖНО ВЫСТРЕЛИТЬ СЕБЕ В НОГУ

Продолжать использовать базовые классы в TestFixture, провайдерах данных или контроллах, даже если у вас есть DI-контейнер

Не использовать Lazy для долгих операций в конструкторах

Регистрировать классы с данными в контейнере, долго доставать эти данные

НО ВСЕ ЕЩЕ МОЖНО ВЫСТРЕЛИТЬ СЕБЕ В НОГУ

Продолжать использовать базовые классы в TestFixture, провайдерах данных или контроллах, даже если у вас есть DI-контейнер

Не использовать Lazy для долгих операций в конструкторах

Регистрировать классы с данными в контейнере, долго доставать эти данные

Переиспользовать контейнер из приложения в тесте

В ИТОГЕ

ПЛЮСЫ

-- Во-первых, это красиво
-- Чистый код тестов: инфраструктура вынесена отдельно, есть только регистрации и тест с AAA
-- Параллельный запуск за счёт scoped-зависимостей и независимой работы с данными
-- Простой быстрый запуск отдельного теста (ничего лишнего)
-- Все данные и зависимости есть в самом тесте (ну, и в полях класса)

В ИТОГЕ

плюсы	минусы
-- Во-первых, это красиво -- Чистый код тестов – инфраструктура вынесена отдельно, есть только регистрации и тест с AAA -- Параллельный запуск за счёт scoped-зависимостей и независимой работы с данными -- Простой быстрый запуск отдельного теста (ничего лишнего) -- Все данные и зависимости есть в самом тесте (ну, и в полях класса)	**Непривычно**

В ИТОГЕ

плюсы	минусы	подводные камни
-- Во-первых, это красиво -- Чистый код тестов – инфраструктура вынесена отдельно, есть только регистрации и тест с AAA -- Параллельный запуск за счёт scoped-зависимостей и независимой работы с данными -- Простой быстрый запуск отдельного теста (ничего лишнего) -- Все данные и зависимости есть в самом тесте (ну, и в полях класса)	Непривычно	


ВОТ ТАК, С ПОМОЩЬЮ НЕХИТРЫХ ПРИСПОСОБЛЕНИЙ
БУХАНКУ БЕЛОГО (ИЛИ ЧЕРНОГО) ХЛЕБА
МОЖНО ПРЕВРАТИТЬ В ТРОЛЛЕЙБУС...
НО ЗАЧЕМ?!
ХМЕЛЬ В РУКИ.

X

xUnit.net

xUnit.net. Lessons Learned

Изоляция тестов: 1 объект класса на каждый тест
Вместо [SetUp] и[TearDown] используются конструктор и Dispose
Вместо [TestFixtureSetup] и [TestFixtureTearDown] используются отдельные классы fixture, которые можно инжектировать через конструктор

xUnit.net

```
public class DatabaseFixture : IDisposable
{
    public SqlConnection Db { get; private set; }
    public DatabaseFixture() {
        Db = new SqlConnection("MyConnectionString");
        // ... initialize data in the test database ...
    }

    public void Dispose() {
        // ... clean up test data from the database ...
    }
}

public class MyDatabaseTests(DatabaseFixture databaseFixture, AnotherFixture anotherFixture)
  : IClassFixture<DatabaseFixture>, IClassFixture<AnotherFixture>
{
    // ... write tests, using databaseFixture.Db to get access to the DB ...
}
```

xUnit.net Проблемы

Внедряет зависимости, но не умеет их собирать	Соорудить что-то для сборки очень нетривиально

xUnit.net Проблемы

```
public class CustomTestAssemblyRunner : XunitTestAssemblyRunner
{
    private readonly IContainer _container;

    public CustomTestAssemblyRunner(ITestAssembly testAssembly, IEnumerable<IXunitTestCase> testCases,
IMessageSink diagnosticMessageSink, IMessageSink executionMessageSink, ITestFrameworkExecutionOptions
executionOptions)
        : base(testAssembly, testCases, diagnosticMessageSink, executionMessageSink, executionOptions)
    {
        var builder = new ContainerBuilder();

        // Здесь каким-то образом достаем и регистрируем зависимости.
        // А ещё по идее это же надо делать на нижележащих уровнях.

        _container = builder.Build();
    }

    protected override XunitTestCollectionRunner CreateTestCollectionRunner(ITestCollection
testCollection)
    {
        return new CustomTestCollectionRunner(_container, testCollection, DiagnosticMessageSink,
MessageBus, TestCaseOrderer, new ExceptionAggregator(Aggregator), CancellationTokenSource);
    }
}
```

Вопросы

меня можно найти в tg
@Vadimyan

Веду канал про .net
@rnddotnetnews