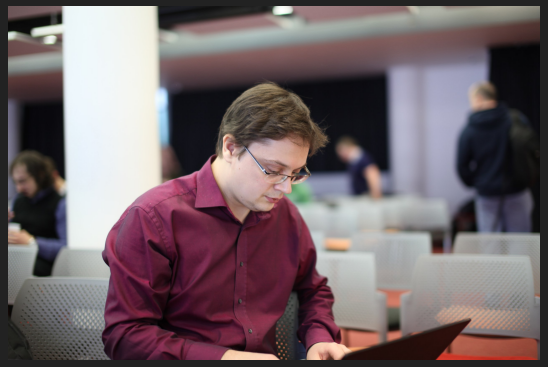
Шаблон Lifetime: для сложного Disposing

Станислав Сидристый

Станислав Сидристый

Epam Systems



- Epam, Kaspersky, Luxoft, ...
- Выступал на конференциях .NEXT, Apps4All Forum, CLRium #1, #2, #3, QA: Conference и проч.
- .NET via C#, CLI/C++, JS

epam

CLRium
World Wildlife

IDisposable

```
public class Disposable : IDisposable
{
    bool _disposed;

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if(disposing)
        {
            // освобождаем управляемые ресурсы
        }
        // освобождаем неуправляемые ресурсы
    }

    protected void CheckDisposed()
    {
        if(_disposed)
        {
            throw new ObjectDisposedException();
        }
    }

    ~Disposable()
    {
        Dispose(false);
    }
}
```

1. Основным плюсом шаблона является возможность детерминированного освобождения ресурсов: тогда, когда это необходимо
2. Введение общеизвестного способа узнать, что конкретный тип требует разрушения его экземпляров в конце использования
3. При грамотной реализации шаблона работа спроектированного типа станет безопасной

1. Основным плюсом шаблона является возможность детерминированного освобождения ресурсов: тогда, когда это необходимо
2. Введение общеизвестного способа узнать, что конкретный тип требует разрушения его экземпляров в конце использования
3. При грамотной реализации шаблона работа спроектированного типа станет безопасной

1. Основным плюсом шаблона является возможность детерменированного освобождения ресурсов: тогда, когда это необходимо
2. Введение общеизвестного способа узнать, что конкретный тип требует разрушения его экземпляров в конце использования
3. При грамотной реализации шаблона работа спроектированного типа станет безопасной

Минусы

Неявная публичная оферта

```
CheckDisposed()
```

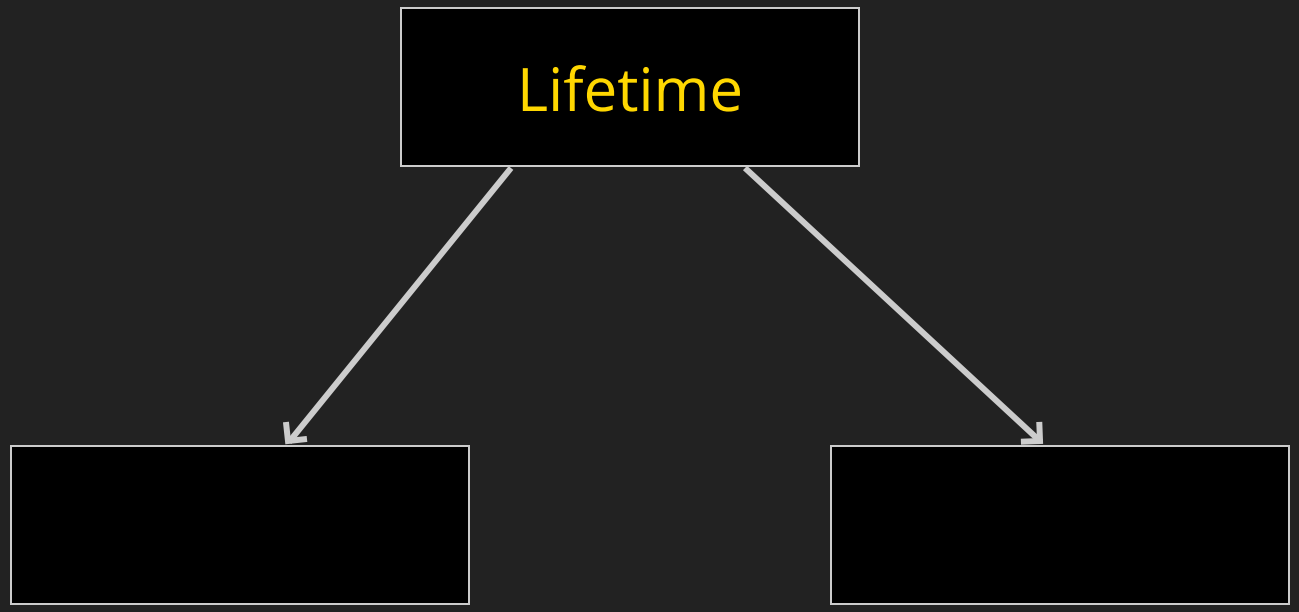
Необходимость интерфейсов
наследовать `IDisposable`

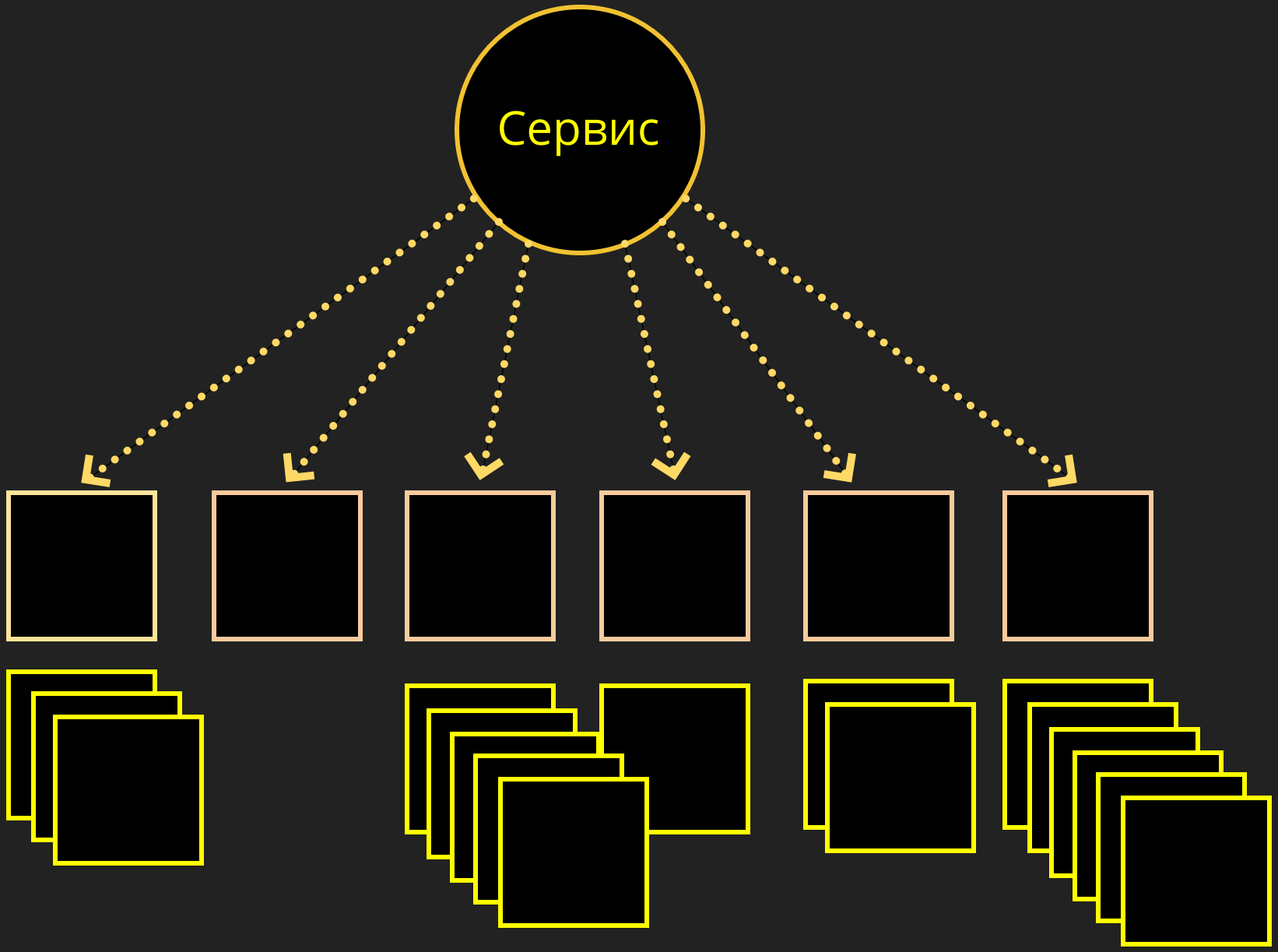
Раздельное объявление Ctor и Dispose()

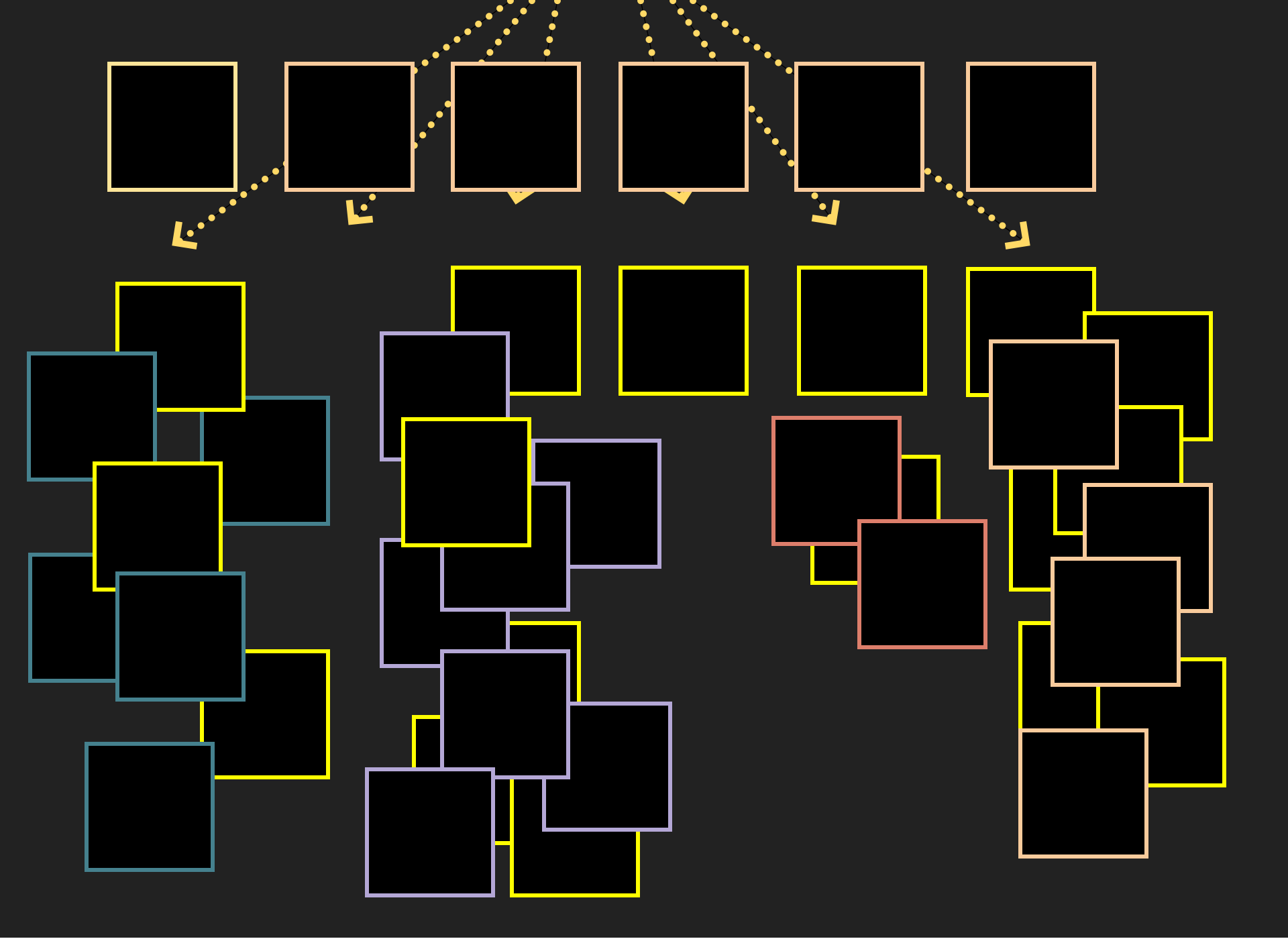
Проблема использования шаблона на графах объектов, многослойной архитектуре

Lifetime


Dispose


Lifetime


Сервис



```
public class Lifetime
{

    public bool IsTerminated { get; }

    public void Add(Action action);

    // Statics
    public static Lifetime Etheral { get; }

    public static LifetimeDef Define();

    public static LifetimeDef DefineDependent(OuterLifetime parent);

    public static Lifetime WhenAll(params OuterLifetime[] lifetimes);

    public static Lifetime WhenAny(params OuterLifetime[] lifetimes);
}
```

```
public struct OuterLifetime
{
    public bool IsTerminated { get; }

    public static implicit operator OuterLifetime(Lifetime lifetime);

    public static implicit operator OuterLifetime(LifetimeDef lifetime)
}
```

```
public class LifetimeDef
{
    public Lifetime Lifetime { get; }

    public void Terminate();
}
```

Demo

Результаты

~~Неявная публичная оферта~~

Явно указано, от чего зависит время жизни

~~Необходимость интерфейсов наследовать IDisposable~~

Интерфейсы остаются чистыми

~~Проблема использования шаблона на графах объектов, многослойной архитектуре~~

Легко

~~Раздельное объявление `ctor` и `Dispose()`~~

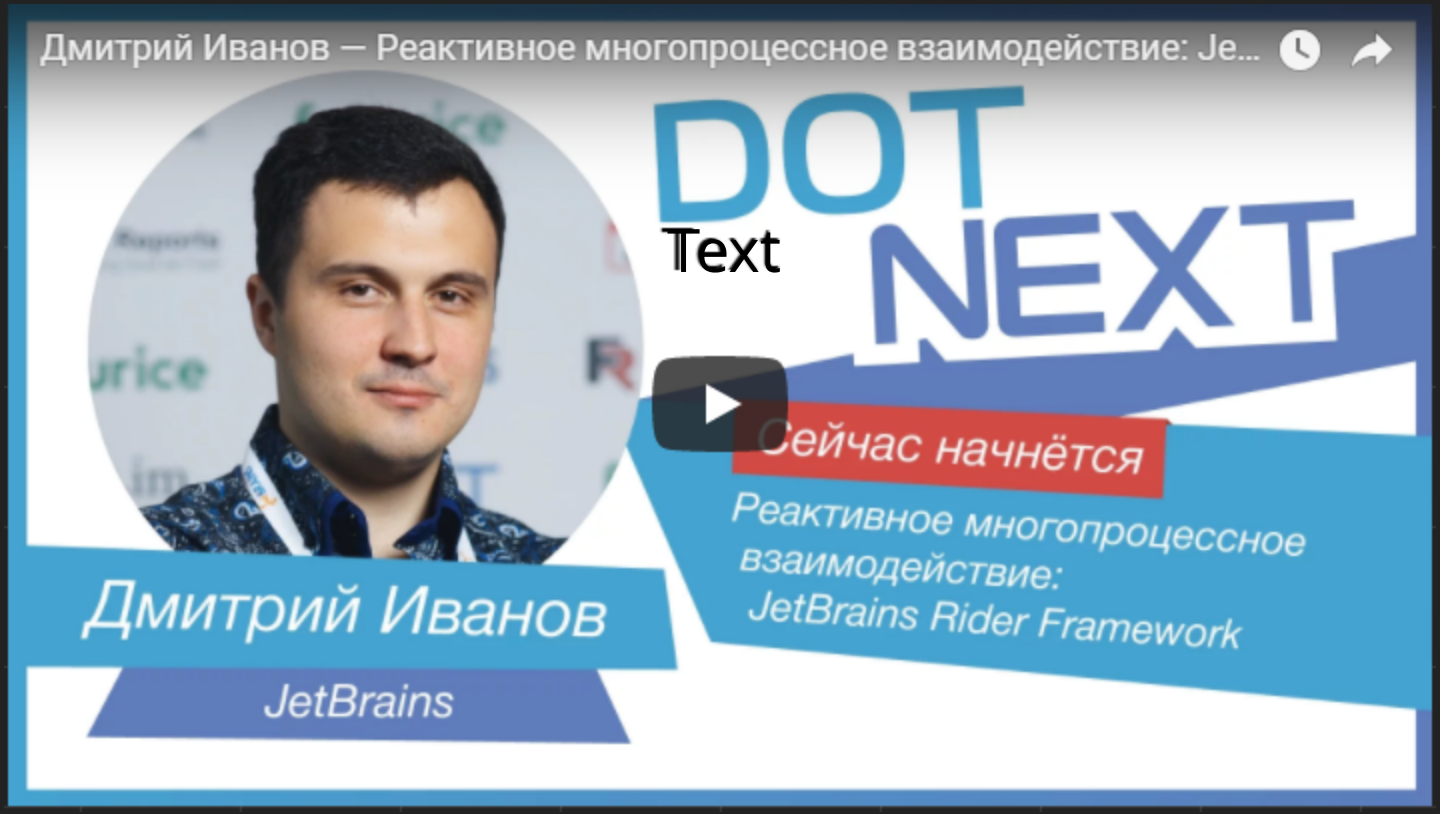
Слитное объявление регистрации и очистки

~~`CheckDisposed()`~~

Только для объектов, имеющих явный LifetimeDef

Links

https://www.youtube.com/watch?v=cfPEN5_6UtI



Links

https://github.com/sidristij/dotnetbook/

QA

github.com/sidristij

sunex.development@gmail.com