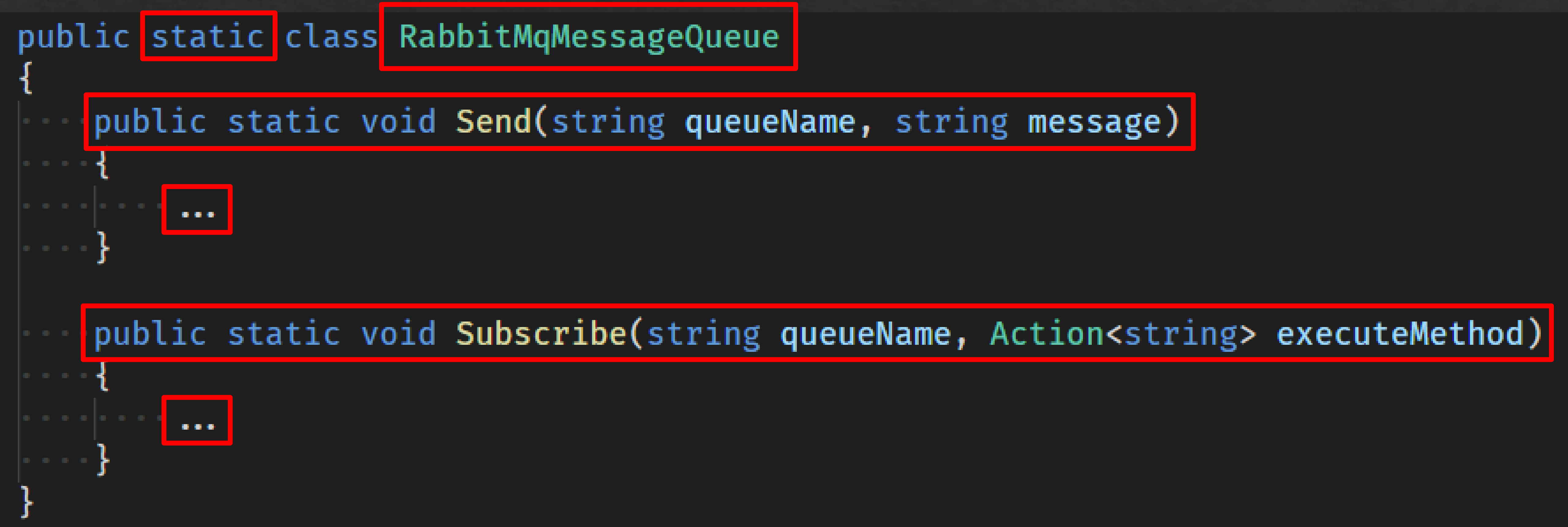
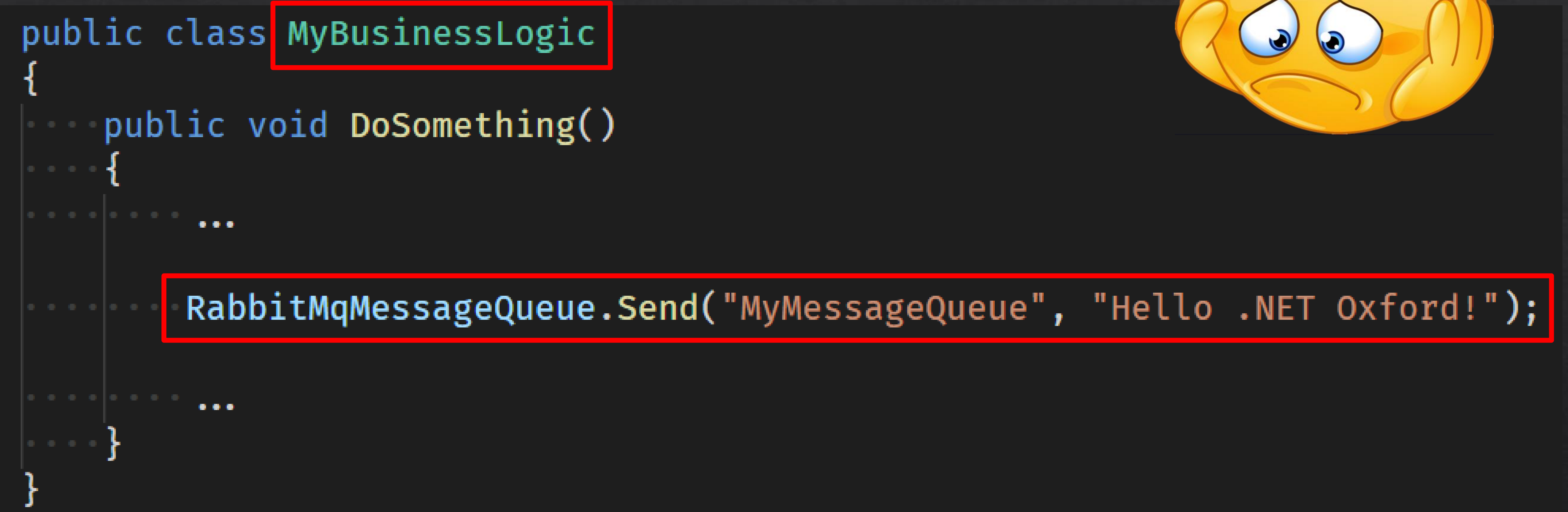
IoC Containers, Dependency Injection and Inversion

Dan Clarke - @dracan



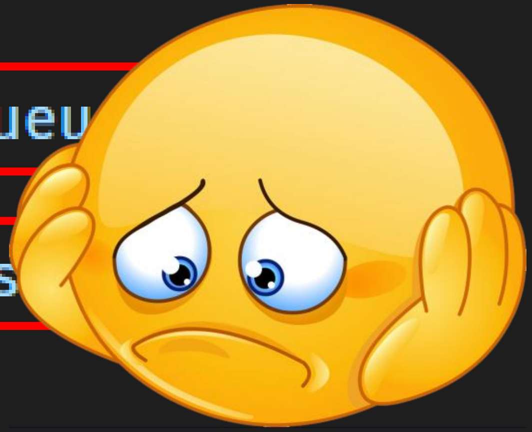
```
public static class RabbitMqMessageQueue
{
    public static void Send(string queueName, string message)
    {
        ...
    }

    public static void Subscribe(string queueName, Action<string> executeMethod)
    {
        ...
    }
}
```

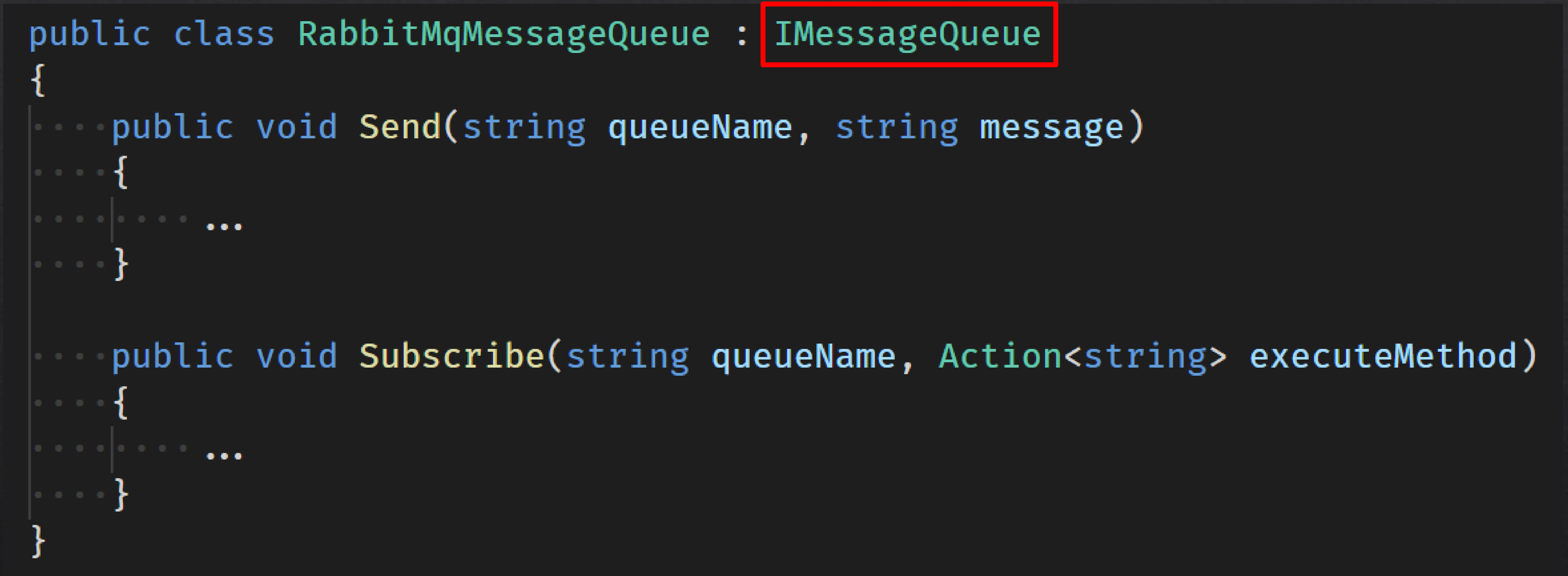


```
public class MyBusinessLogic
{
    public void DoSomething()
    {
        ...

        RabbitMqMessageQueue.Send("MyMessageQueue", "Hello .NET Oxford!");

        ...
    }
}
```

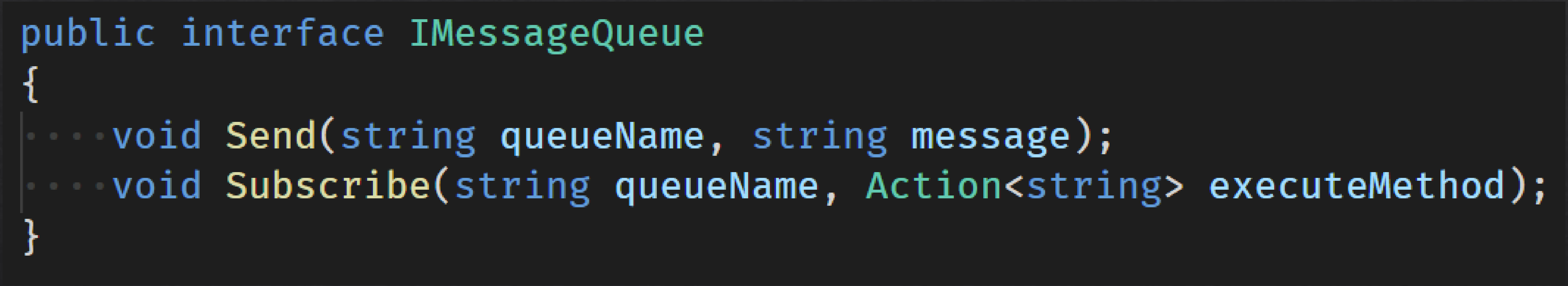
```
public class MyBusinessLogic
{
    public void DoSomething()
    {
            ...

        var messageQueue = new RabbitMqMessag
        messageQueue.Send("MyMessageQueue", "                        ");

            ...
    }
}
```

```
public class MyBusinessLogic
{
    private readonly RabbitMqMessageQueue _messageQueu

    public MyBusinessLogic(RabbitMqMessageQueue mess
    {
        _messageQueue = messageQueue;
    }

    public void DoSomething()
    {
        ...

        _messageQueue.Send("MyMessageQueue", "Hello .NET Oxford!");

        ...
    }
}
```



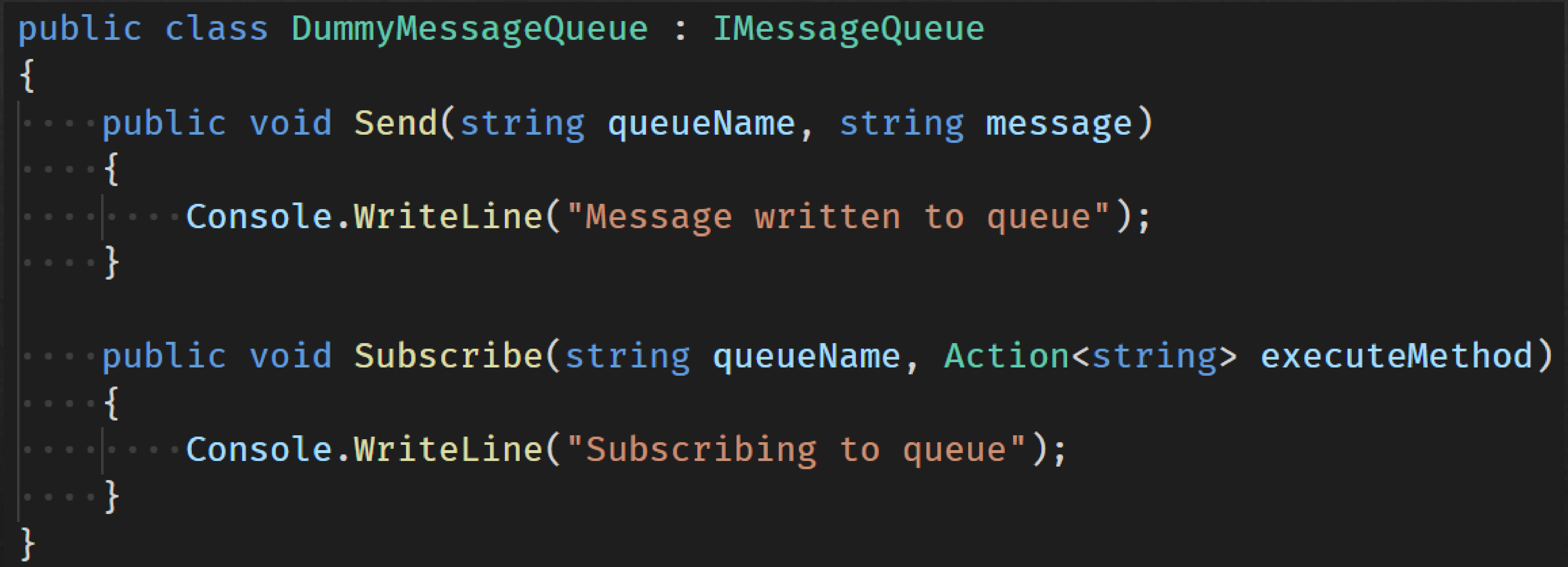


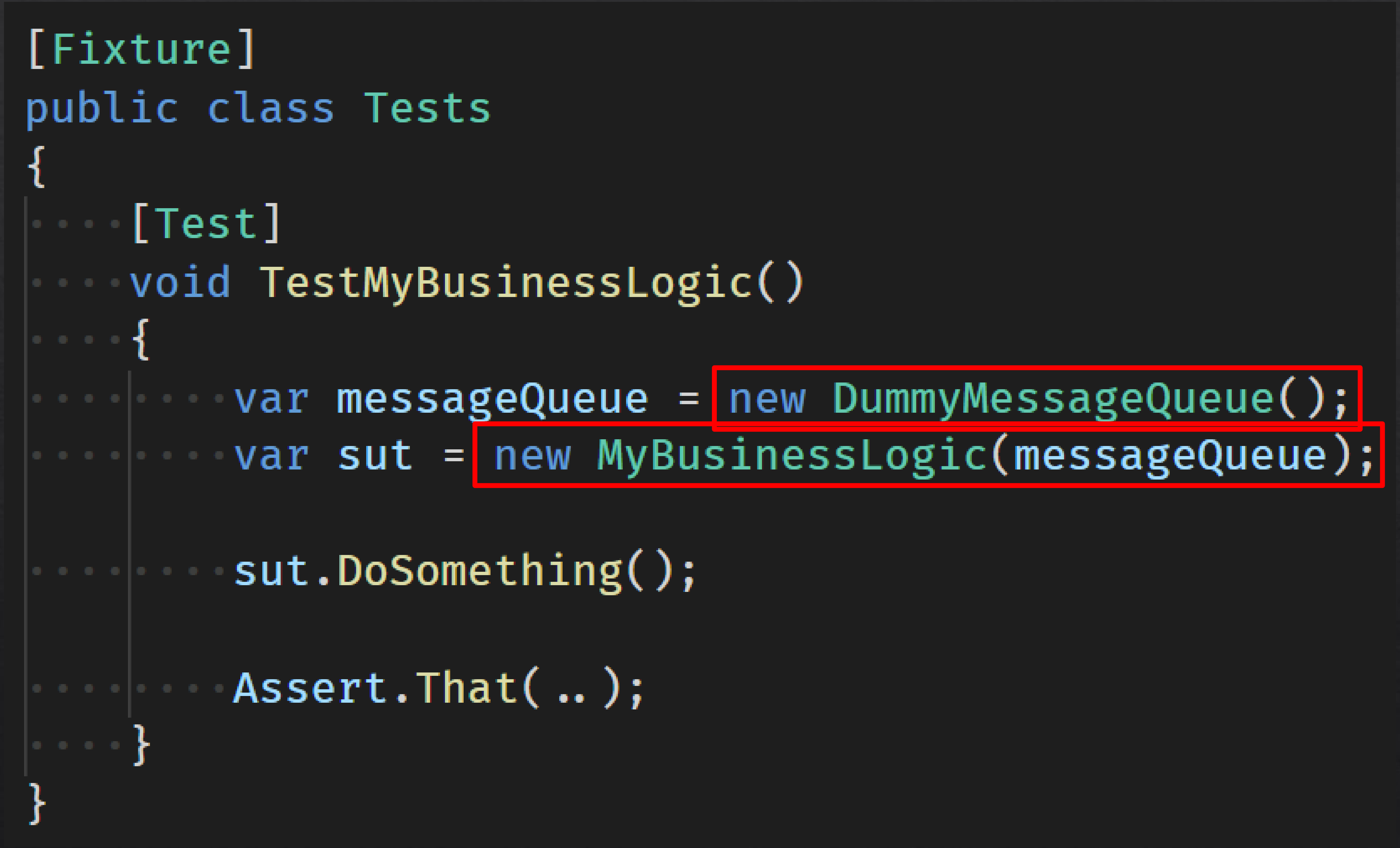
```
public class RabbitMqMessageQueue : IMessageQueue
{
    public void Send(string queueName, string message)
    {
        ...
    }

    public void Subscribe(string queueName, Action<string> executeMethod)
    {
        ...
    }
}
```



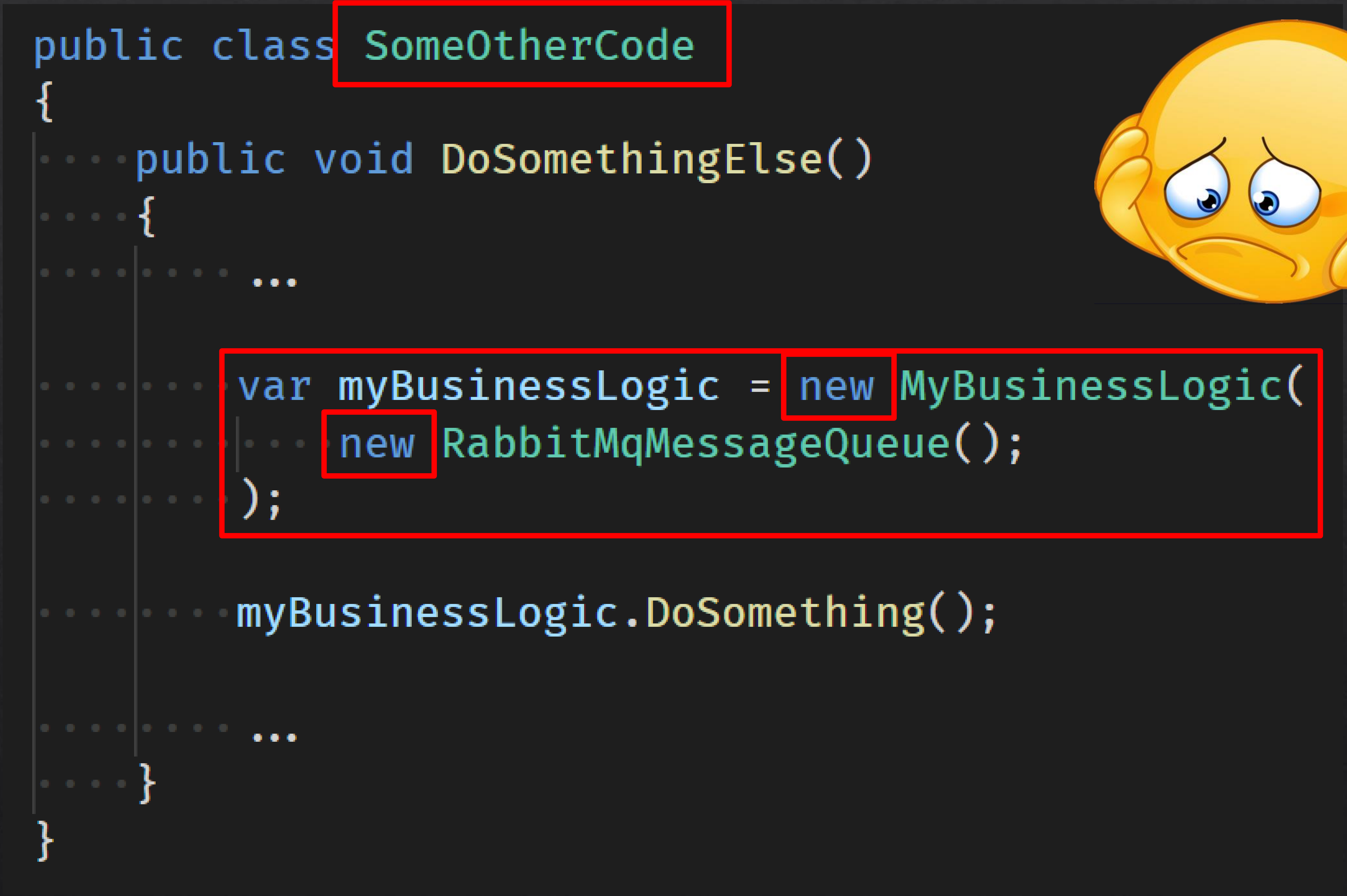
```csharp
public interface IMessageQueue
{
    void Send(string queueName, string message);
    void Subscribe(string queueName, Action<string> executeMethod);
}
```

```
public class MyBusinessLogic
{
    private readonly IMessageQueue _messageQueue;

    public MyBusinessLogic(IMessageQueue messageQue
    {
        _messageQueue = messageQueue;
    }

    public void DoSomething()
    {
        ...

        _messageQueue.Send("MyMessageQueue", "Hello .NET Oxford!");

        ...
    }
}
```

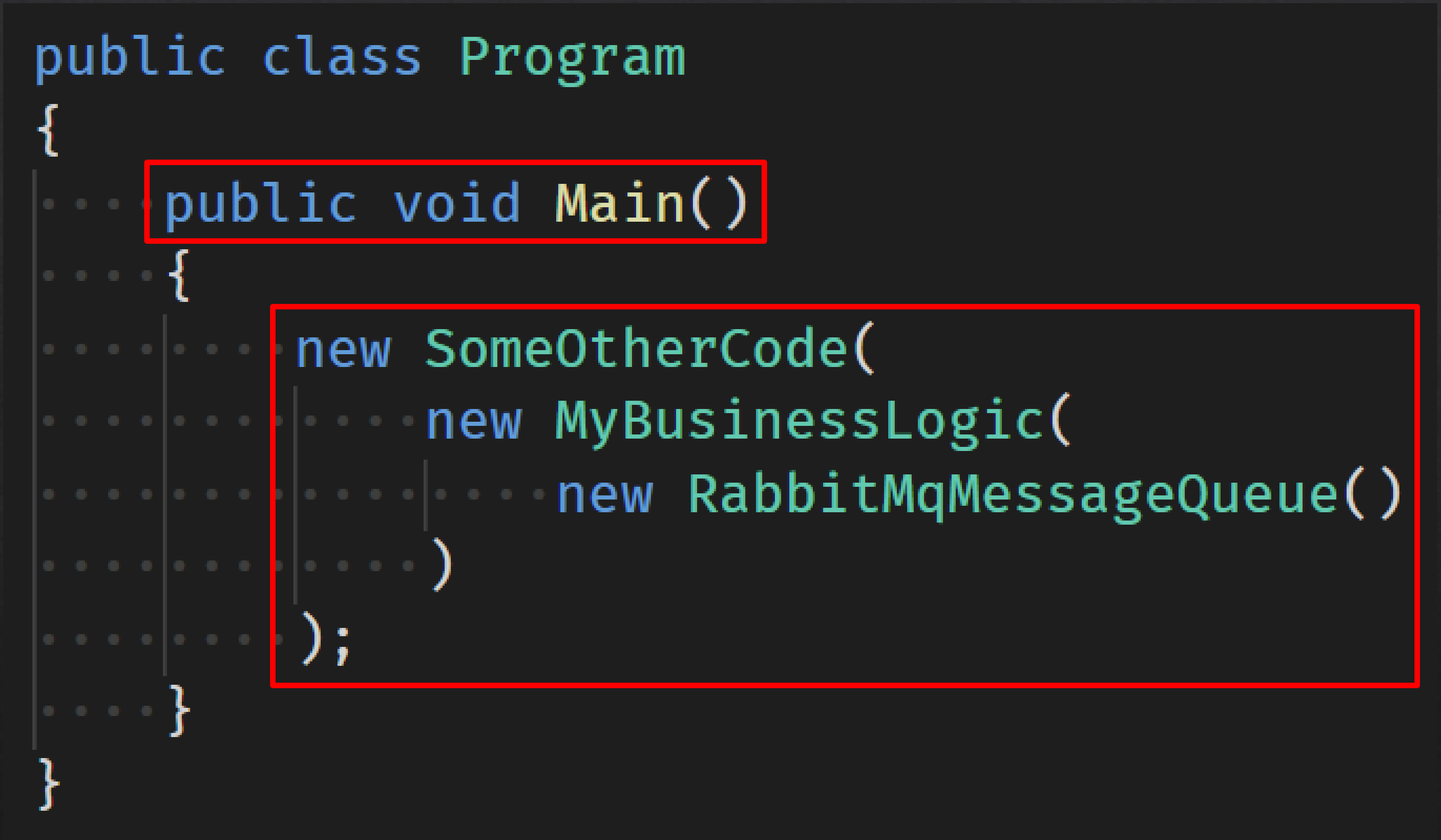

```csharp
public class DummyMessageQueue : IMessageQueue
{
    public void Send(string queueName, string message)
    {
        Console.WriteLine("Message written to queue");
    }

    public void Subscribe(string queueName, Action<string> executeMethod)
    {
        Console.WriteLine("Subscribing to queue");
    }
}
```



```
[Fixture]
public class Tests
{
    [Test]
    void TestMyBusinessLogic()
    {
        var messageQueue = new DummyMessageQueue();
        var sut = new MyBusinessLogic(messageQueue);

        sut.DoSomething();

        Assert.That(..);
    }
}
```



```
public class SomeOtherCode
{
    public void DoSomethingElse()
    {
        ...

        var myBusinessLogic = new MyBusinessLogic(
            new RabbitMqMessageQueue();
        );

        myBusinessLogic.DoSomething();


        ...
    }
}
```



```
public class Program
{
    public void Main()
    {
        new SomeOtherCode(
            new MyBusinessLogic(
                new RabbitMqMessageQueue()
            )
        );
    }
}
```

IoC Containers

AUTOFAC

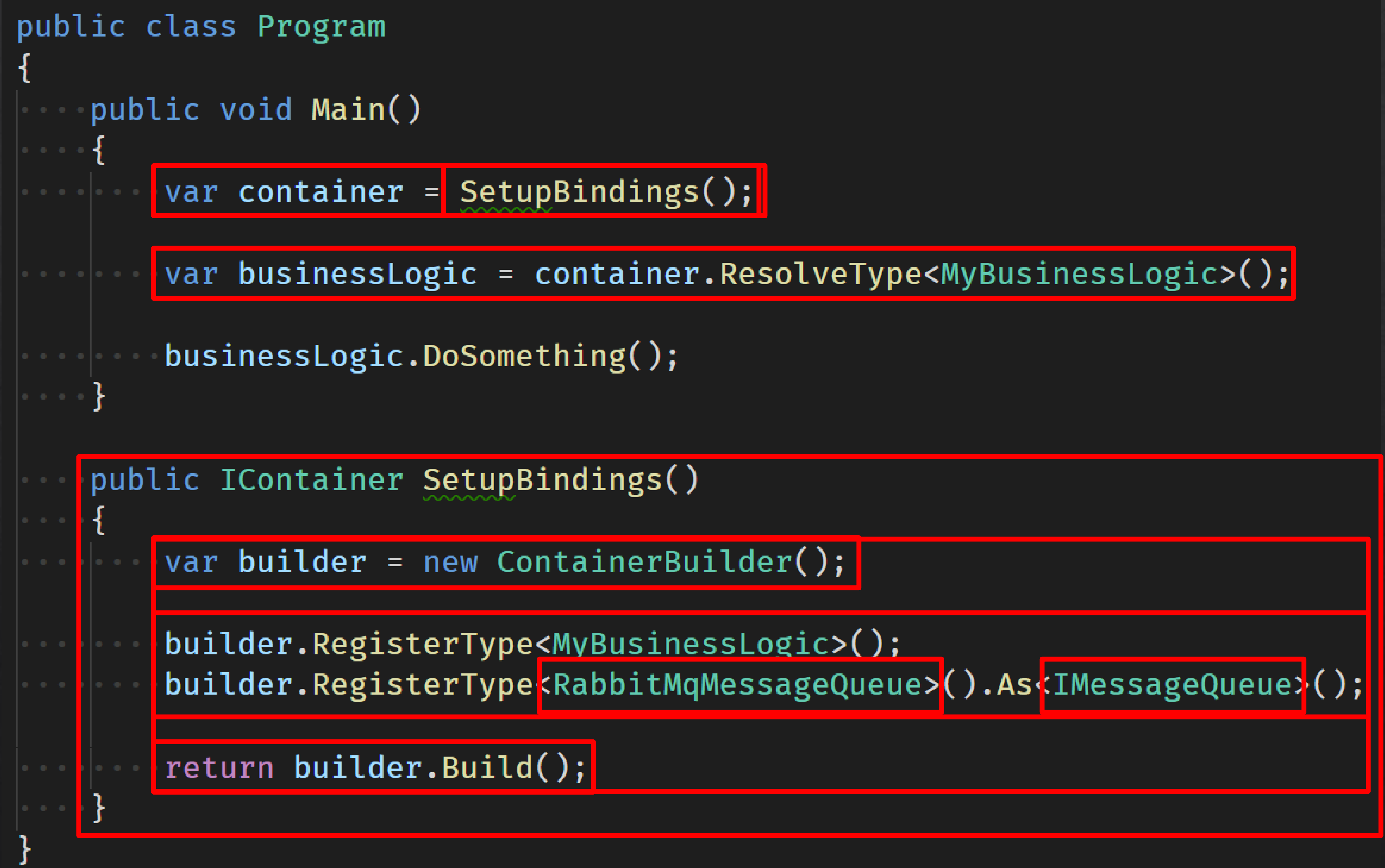
Castle Project

StructureMap

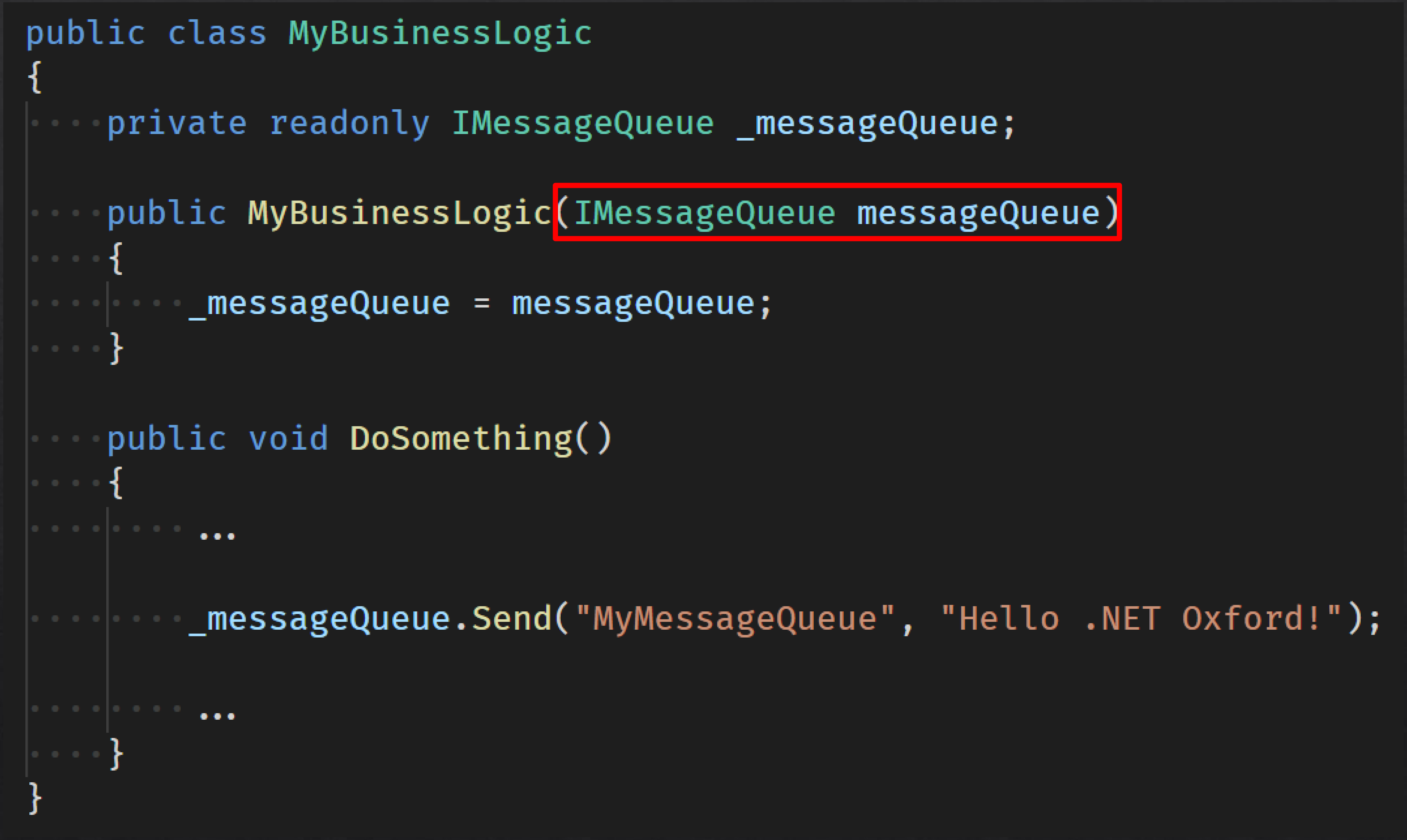
UNITY

.NET Core

Ninject!
the ninja of .net dependency injectors

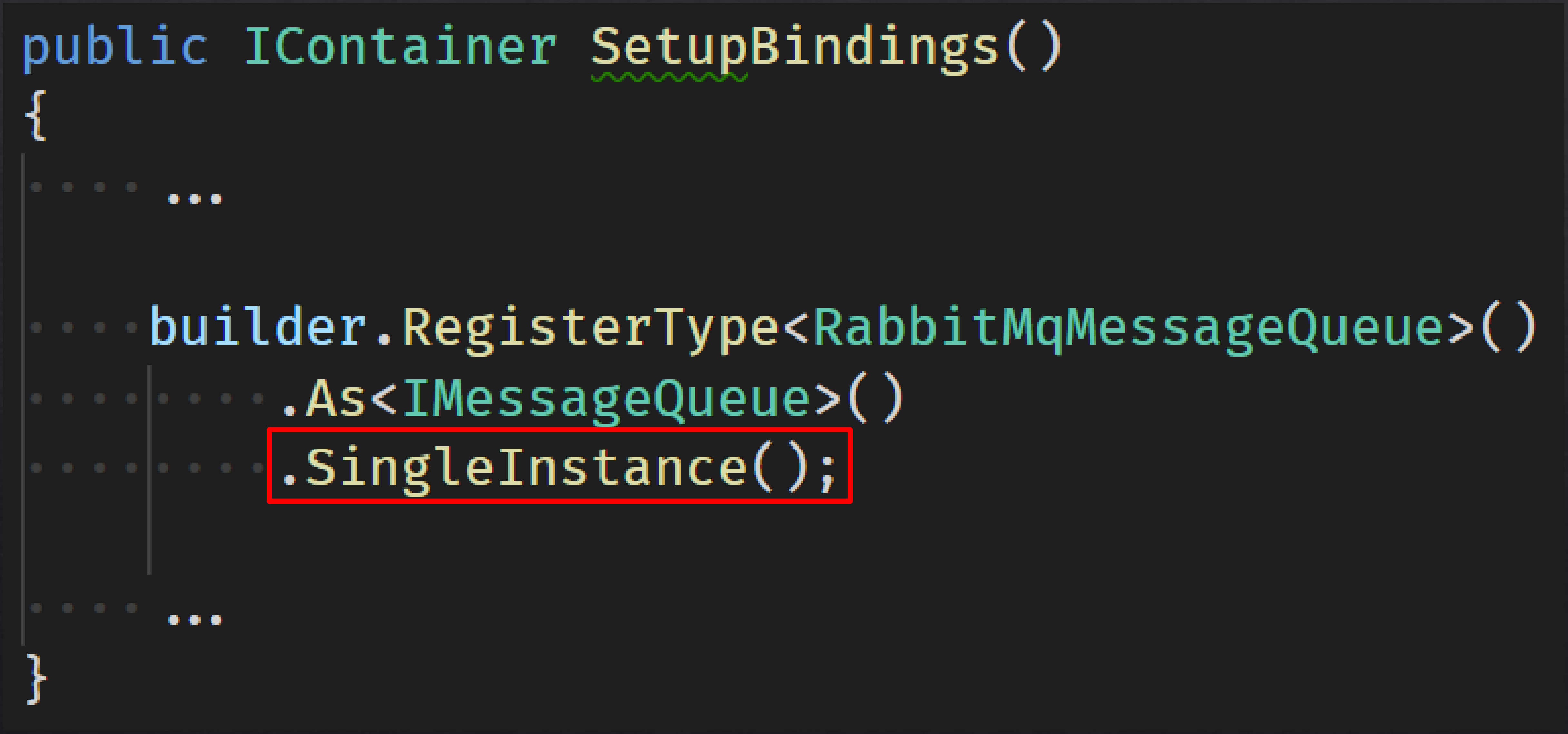


```
public class Program
{
    public void Main()
    {
        var container = SetupBindings();

        var businessLogic = container.ResolveType<MyBusinessLogic>();

        businessLogic.DoSomething();
    }

    public IContainer SetupBindings()
    {
        var builder = new ContainerBuilder();

        builder.RegisterType<MyBusinessLogic>();
        builder.RegisterType<RabbitMqMessageQueue>().As<IMessageQueue>();

        return builder.Build();
    }
}
```



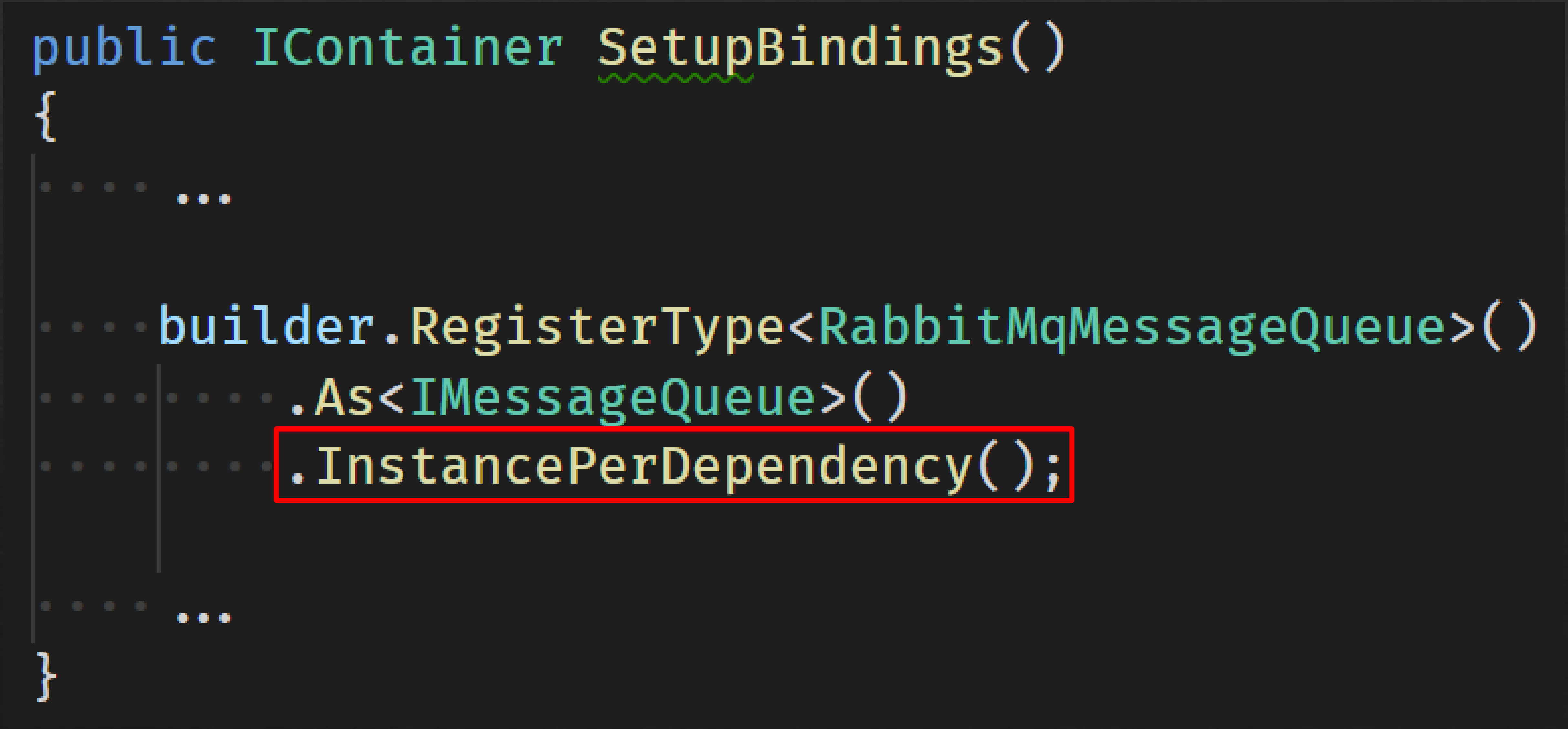
```csharp
public class MyBusinessLogic
{
    private readonly IMessageQueue _messageQueue;

    public MyBusinessLogic(IMessageQueue messageQueue)
    {
        _messageQueue = messageQueue;
    }

    public void DoSomething()
    {
        ...

        _messageQueue.Send("MyMessageQueue", "Hello .NET Oxford!");

        ...
    }
}
```

Singleton scope



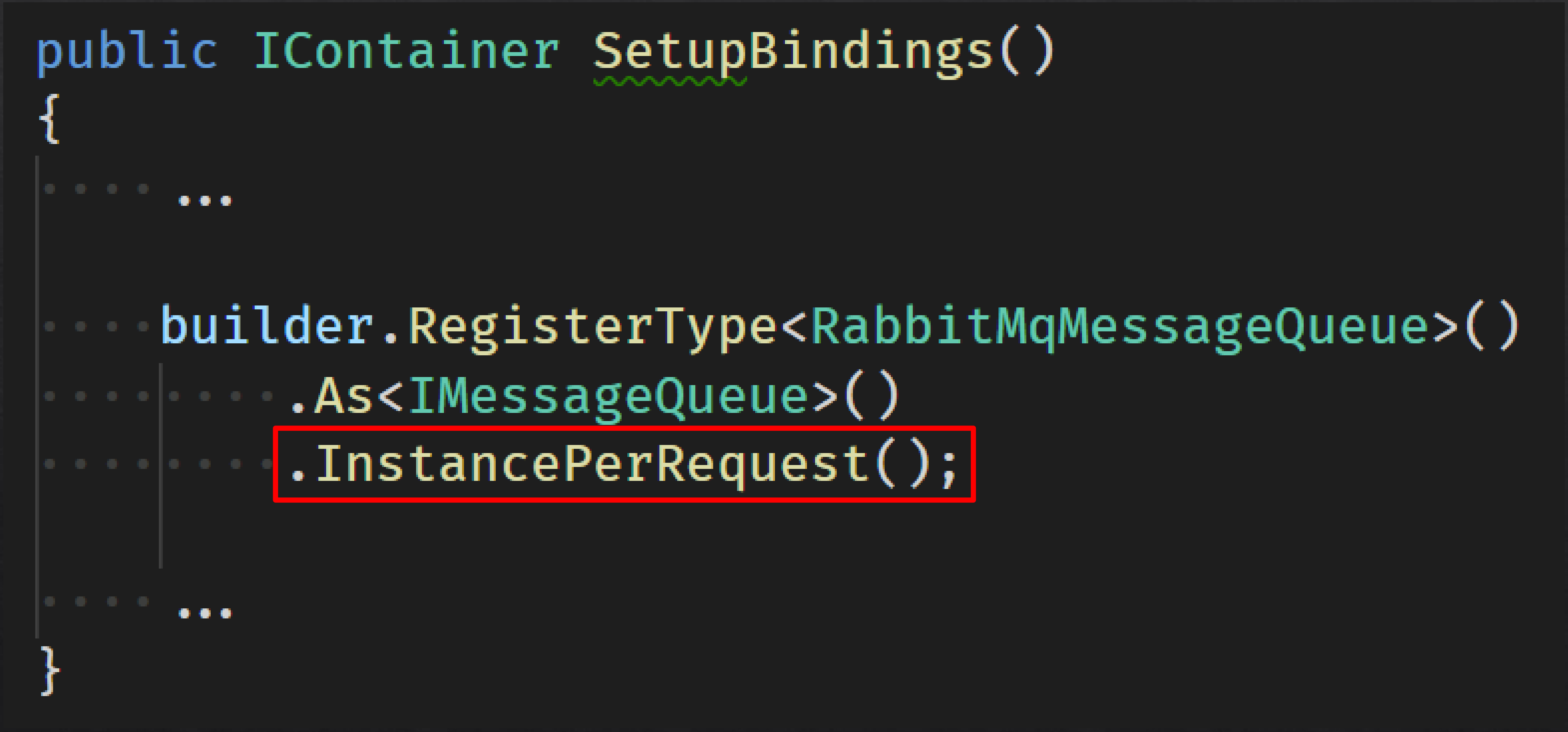
```
public IContainer SetupBindings()
{
    ...

    builder.RegisterType<RabbitMqMessageQueue>()
        .As<IMessageQueue>()
        .SingleInstance();

    ...
}
```

Instance per Dependency Scope

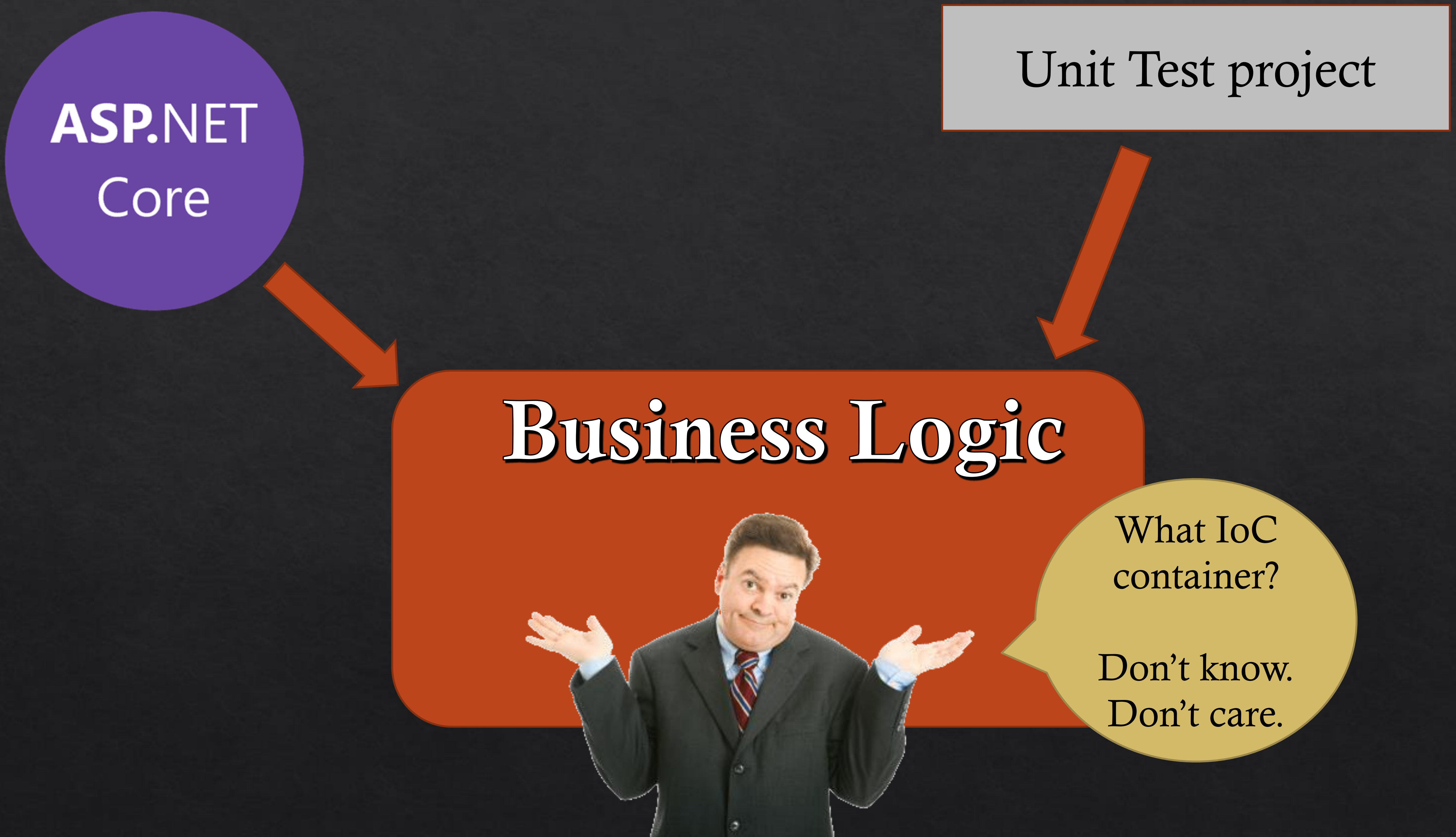


```
public IContainer SetupBindings()
{
    ...

    builder.RegisterType<RabbitMqMessageQueue>()
        .As<IMessageQueue>()
        .InstancePerDependency();

    ...
}
```

Instance per Request



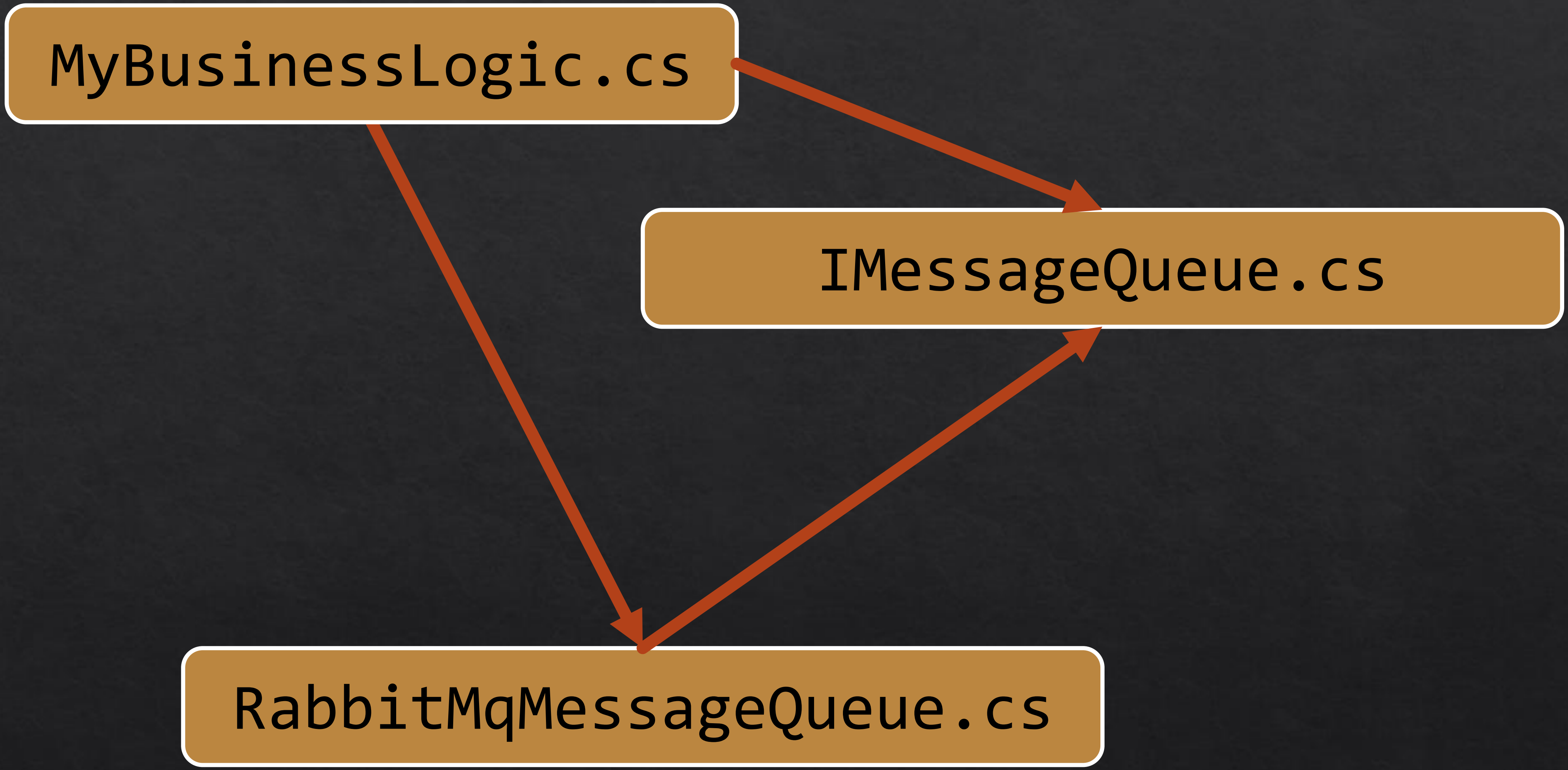
```
public IContainer SetupBindings()
{
    ...

    builder.RegisterType<RabbitMqMessageQueue>()
        .As<IMessageQueue>()
        .InstancePerRequest();

    ...
}
```


ASP.NET Core
Unit Test project
Business Logic
What IoC container?
Don’t know. Don’t care.

Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions (e.g. interfaces).

- Abstractions should not depend on details. Details (concrete implementations) should depend on abstractions.

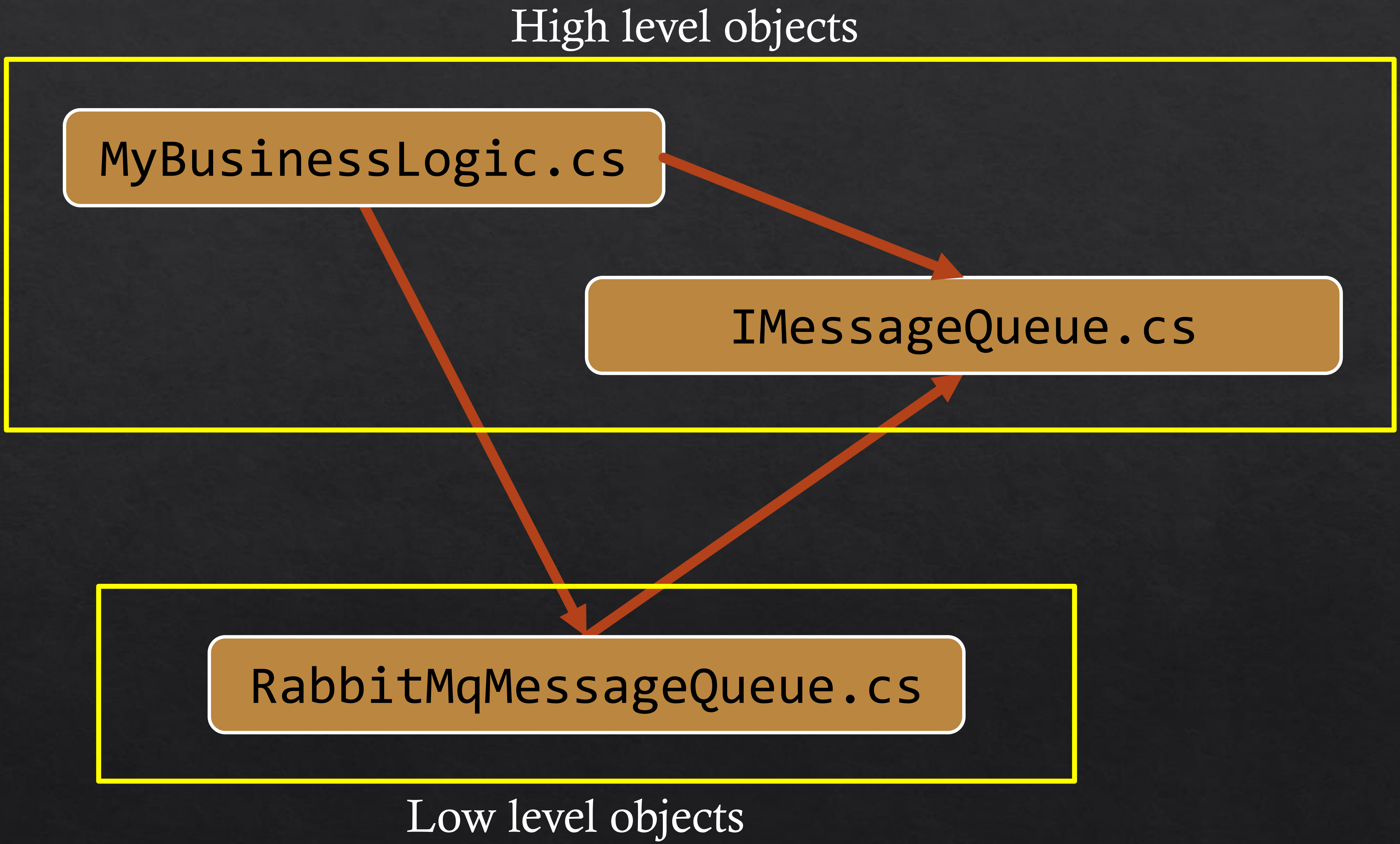

MyBusinessLogic.cs
IMessageQueue.cs
RabbitMqMessageQueue.cs

Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions (e.g. interfaces).

- Abstractions should not depend on details. Details (concrete implementations) should depend on abstractions.

What does “Inversion” mean though???


High level objects
MyBusinessLogic.cs
IMessageQueue.cs
RabbitMqMessageQueue.cs
Low level objects

@dracan