

Dependency Injection Testing Entity Framework

Rasmus Lystrøm
Associate Professor
ITU
rnie@itu.dk

```
1 using System;
2
3 namespace BDSA2018.Lecture01
4 {
5     1 reference
6     public class Program
7     {
8         1 reference
9         public static void Main(string[] args)
10        {
11            Console.WriteLine("Hello World!");
12        }
13    }
14 }
15
16 BDSA2018.Lecture01.Tests.csproj
17
18 ProgramTests.cs
19
20 BDSA2018.Lecture01.sln
```

```
1 using System;
2 using System.IO;
3 using Xunit;
4
5 namespace BDSA2018.Lecture01.Tests
6 {
7     0 references | Run All Tests | Debug All Tests
8     public class ProgramTests
9     {
10         [Fact]
11         0 references | Run Test | Debug Test
12         public void Main_prints_HelloWorld()
13         {
14             // Arrange
15             var writer = new StringWriter();
16             Console.SetOut(writer);
17
18             // Act
19             Program.Main(new string[] { });
20
21             // Assert
22             var output = writer.GetStringBuilder().ToString();
23             Assert.Equal("Hello World!", output);
24         }
25     }
26 }
```

```
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.1.3\System.Private.CoreLib.dll'. Skipped loading symbols. Module was already loaded.
the debugger option 'Just My Code' is enabled.
Loaded 'C:\Users\rasmus\l\Desktop\BDSA2018.Lecture01\BDSA2018.Lecture01\bin\Debug\netcoreapp2.1\BDSA2018.Lecture01.dll'. Skipped loading symbols. Module was already loaded.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.1.3\System.Runtime.dll'. Skipped loading symbols. Module was already loaded.
the debugger option 'Just My Code' is enabled.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.1.3\System.Console.dll'. Skipped loading symbols. Module was already loaded.
the debugger option 'Just My Code' is enabled.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.1.3\System.Threading.dll'. Skipped loading symbols. Module was already loaded.
the debugger option 'Just My Code' is enabled.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.1.3\System.Runtime.Extensions.dll'. Skipped loading symbols. Module was already loaded.
the debugger option 'Just My Code' is enabled.
Hello World!
The program '[15304] BDSA2018.Lecture01.dll' has exited with code 0 (0x0).
```

Agenda

Testing ...

Dependency Injection

Testing Entity Framework

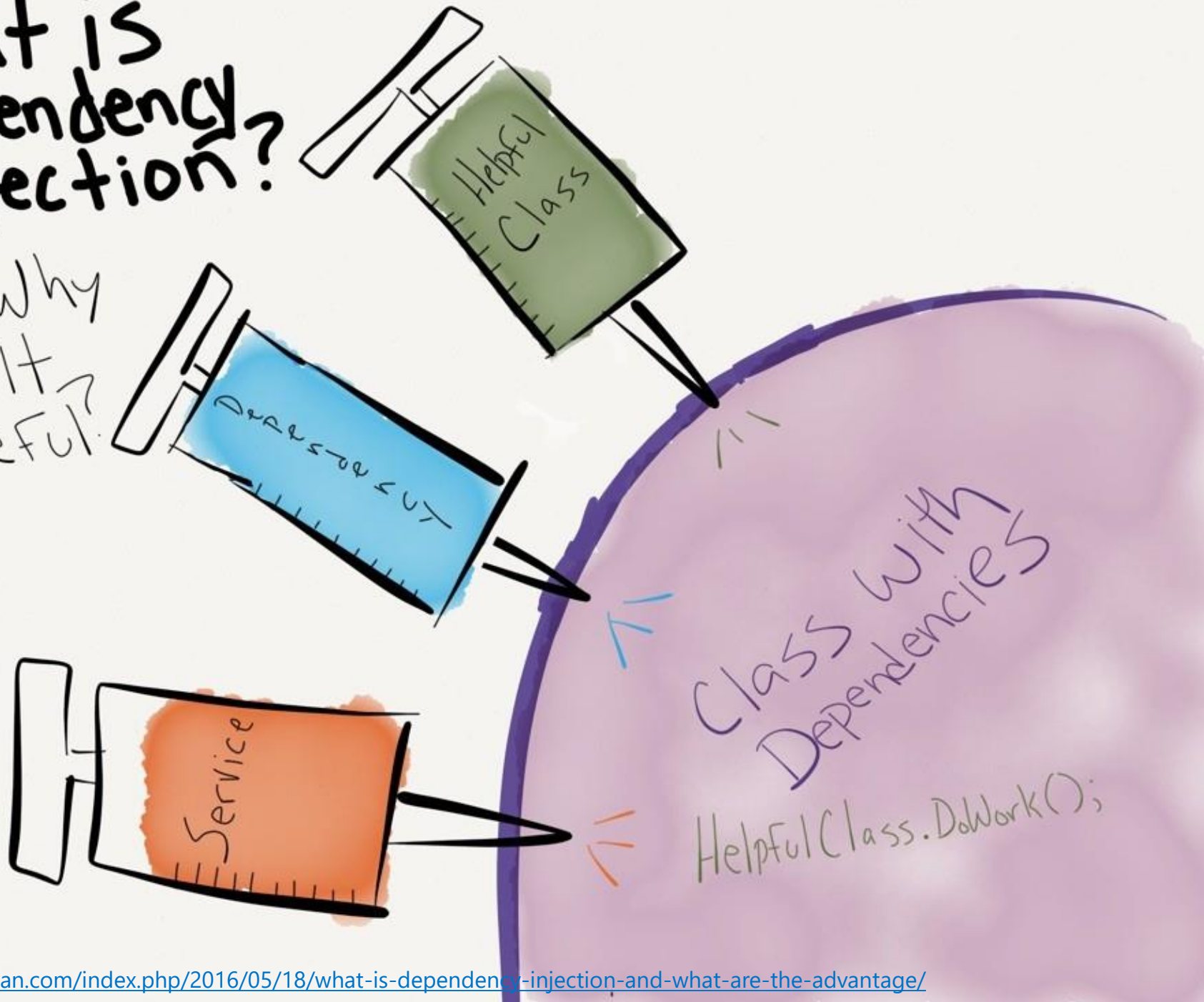
Testing ...

- a. Testing live databases is hard
- b. Testing live full systems is hard
- c. By transitivity: Testing ... is hard...



What is Dependency Injection?

Why is it useful?



Dependency Injection (DI)

Software design pattern which implements Inversion of Control (IoC)

Constructor
Injection

Property (setter)
Injection

Interface
Injection

Structured readable code

Testable code

Dependency Inversion Principle

Separation of Concerns

Rock SOLID!!!

Pun intended

AWESOME!!

Programming to interface, not implementation...

```
public interface IFooService
{
    bool Update(Foo foo);
}

public class FooService : IFooService
{
    bool Update(Foo foo)
    {
        // Implementation
    }
}
```

Using IF

```
public class  
{  
    public  
    {  
        IFc  
        var  
        IFc  
        ret  
    }  
}
```



Constructor Injection (preferred)

```
public class Worker
{
    private readonly IFooService _service;

    public Worker(IFooService service)
    {
        _service = service;
    }

    public bool DoWork(FooDto fooDto)
    {
        // Implementation
    }
}
```

Private readonly
field

Initialize from
constructor

Property Injection

Public setter

```
public class Worker
{
    public IFooService Service { private get; set; }

    public void DoWork(Foo foo)
    {
        Service?.Update(foo);
    }
}
```

Is this King?

Interface Injection

```
public interface IServiceSetter<T>
{
    void SetService(T service);
}
```

```
public interface IServiceSetter<T>
{
    T Service { set; }
}
```

Interface Injection II

Interface

```
public class Worker : IServiceSetter<IFooService>
{
    private IFooService _service;

    public void SetService(IFooService service)
    {
        _service = service;
    }

    public void DoWork(FooDto fooDto)
    {
        // Implementation
    }
}
```

Implement
interface

Interface Injection III

Interface

```
public class Worker : IServiceSetter<IFooService>
{
    public IFooService Service { private get; set; }

    public bool DoWork(FooDto fooDto)
    {
        // Implementation
    }
}
```

Implement
interface

Best practices

Use Adapter to
enable interface
if needed

Use constructor
injection

Program to
interface

Use an IoC
container

In a couple of
weeks...

IoC Container

```
PM> Install-Package Microsoft.Extensions.DependencyInjection
```

```
IServiceCollection services = new ServiceCollection();
```

```
services.AddScoped<IService, Service>();
```

```
var provider = services.BuildServiceProvider();
```

```
var service = provider.GetRequiredService<IService>();
```

Unit Testing

Unit Testing Best Practices

Never test against a live database, file, or web service

Single Responsibility Principle

Only test the "System Under Test"

Atomic tests

Use either mocks or stubs

Stub testing



Test stub

```
public class FooServiceFalseStub : IFooService
{
    public bool Update(Foo foo)
    {
        return false;
    }
}
```

Stub testing II

```
public class WorkerTests
{
    [Fact]
    public void DoWork_when_IFooService_Update_false_returns_false()
    {
        IFooService service = new FooServiceFalseStub();

        using (var worker = new Worker(service))
        {
            var result = worker.DoWork(new FooDto());

            Assert.False(result);
        }
    }
}
```

Mock testing

Mock using Moq

```
public class WorkerTests
{
    [Fact]
    public void DoWork_when_IFooService_Update_returns_false_returns_false()
    {
        var mock = new Mock<IFooService>();
        IFooService service = mock.Object;

        using (var worker = new Worker(service))
        {
            var result = worker.DoWork(new FooDto());

            Assert.False(result);
        }
    }
}
```

Mock testing II

Configure the mock



```
public class WorkerTests
{
    [Fact]
    public void DoWork_when_IFooService_Update_true_returns_true()
    {
        var mock = new Mock<IFooService>();
        mock.Setup(m => m.Update(It.IsAny<Foo>())).Returns(true);

        using (var worker = new Worker(mock.Object))
        {
            var result = worker.DoWork(new FooDto());

            Assert.True(result);
        }
    }
}
```

Demo

Testing Entity Framework

In Memory Database

```
dotnet add package Microsoft.EntityFrameworkCore.InMemory  
// In Memory Database:  
var builder = new DbContextOptionsBuilder<MyContext>()  
                .UseInMemoryDatabase(databaseName: nameof(<name>));
```

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite  
// SQLite:  
var connection = new SqlConnection("DataSource=:memory:");  
connection.Open();
```

```
var builder = new DbContextOptionsBuilder<MyContext>()  
                .UseSqlite(connection);
```

```
var context = new FuturamaContext(builder.Options);  
context.Database.EnsureCreated();
```


Best practices

Wrap in logical
units/service
classes/repositories

Program to
interface

Don't test
built-in code...

Demo