Vejledende Løsning

EjendomBeregner - Unit Test

03-Ejendomsberegner-Unittest

Programmering - 2. semester

Opgaven

Lav Unit Tests til "Ejendomsberegner - Interface og IoC" med xUnit v3 og Moq

Minimumskrav:

- Unit test af BeregnKvadratmeter metoden
- Unit test af indlæsning fra tekstfil

Hints fra opgaven:

- Anvend Moq til at mocke afhængigheder
- Lav en adapter indkapsling af File.ReadAllLines()
- Brug Adapter Pattern (IFile interface)

Arkitektur oversigt

Ejendomsberegner.Core

Model/Lejemaal.cs
IEjendomBeregnerService
ILejemaalRepository
EjendomBeregnerService
LejemaalFraFilRepository
FileReaderAdapter/IFile
FileReaderAdapter/FileAdapter

Ejendomsberegner.Test

EjendomsberegnerServiceTest
LejemaalFraFilRepositoryTest

xUnit v3 + Moq

EjendomsberegnerIoC

Program.cs
LejemaalData.csv

Microsoft.Extensions.DI

← refererer til →

Test og IoC projekterne refererer begge til Core

Core har ingen afhængigheder til de andre projekter (DIP)

Model: Lejemaal (record)

```
namespace Ejendomsberegner.Core.Model;

public record Lejemaal
{
    public int Lejlighednummer { get; set; }
    public double Kvadratmeter { get; set; }
    public int AntalRum { get; set; }
}
```

Record type giver automatisk Equals(), GetHashCode(), ToString()

Vigtigt for Assert.Equal() i tests - records sammenlignes på værdi

Tre properties: Lejlighednummer, Kvadratmeter, AntalRum

Svarer til en linje i CSV-filen

Interfaces - nøglen til testbarhed

```
// Service interface
public interface IEjendomBeregnerService
{
    double BeregnKvadratmeter();
}

// Repository interface
public interface ILejemaalRepository
{
    List<Lejemaal> HentLejemaal();
}

// File adapter interface
public interface IFile
{
    string[] ReadAllLines();
}
```

Hvert interface kan mockes i unit tests

IEjendomBeregnerService - beregningslogik

ILejemaalRepository - dataadgang

IFile - adapter til filsystem (File.ReadAllLines)

EjendomBeregnerService

```
public class EjendomBeregnerService : IEjendomBeregnerService
{
    private readonly ILejemaalRepository _repo;

    public EjendomBeregnerService(ILejemaalRepository repo)
    {
        _repo = repo;
    }

    double IEjendomBeregnerService.BeregnKvadratmeter()
    {
        var lejemaalene = _repo.HentLejemaal();
        var kvadratmeter = 0.0;

        foreach (var lejemaal in lejemaalene)
            kvadratmeter += lejemaal.Kvadratmeter;

        return kvadratmeter;
    }
}
```

Constructor injection: ILejemaalRepository injiceres

BeregnKvadratmeter henter lejemål og summerer kvadratmeter

Eksplicit interface-implementering (IEjendomBeregnerService.)

For at teste: Mock ILejemaalRepository

Test 1: EjendomsberegnerServiceTest

Mock ILejemaalRepository → test at BeregnKvadratmeter summerer korrekt

```
[Fact]
public void Given_BeregnKvadratmeter_Faar_LejemaalListe__Then_Beregnes_Korrekt()
{
    // Arrange
    var lejemaal = new List<Lejemaal>();
    lejemaal.Add(new Lejemaal { Lejlighednummer = 1, Kvadratmeter = 50.0, AntalRum = 2 });
    lejemaal.Add(new Lejemaal { Lejlighednummer = 2, Kvadratmeter = 75.0, AntalRum = 3 });

    var expected = 125.0;

    var lejemaalRepo = new Mock<ILejemaalRepository>();
    lejemaalRepo.Setup(repo ⇒ repo.HentLejemaal()).Returns(lejemaal);
    var service = new EjendomBeregnerService(lejemaalRepo.Object)
                    as IEjendomBeregnerService;

    // Act
    var actual = service.BeregnKvadratmeter();

    // Assert
    Assert.Equal(expected, actual);
}
```

Analyse: EjendomsberegnerServiceTest

Arrange

1. Opret test-data
   (List<Lejemaal>)

2. Definer expected
   (125.0)

3. Mock ILejemaalRepository
   Setup HentLejemaal()

4. Opret SUT via
   constructor injection

Act

Kald BeregnKvadratmeter()

Bemærk cast:

```
service as
  IEjendomBeregnerService
```

Nødvendig pga. eksplicit
interface-implementering

Assert

```
Assert.Equal(
  expected, actual)
```

Sammenligner:
expected = 125.0
actual = 50.0 + 75.0
= 125.0 ✓

SUT = System Under Test = EjendomBeregnerService

LejemaalFraFilRepository

```
public class LejemaalFraFilRepository : ILejemaalRepository
{
    private readonly IFile _dataFile;

    public LejemaalFraFilRepository(IFile dataFile)
    {
        _dataFile = dataFile;
    }

    List<Lejemaal> ILejemaalRepository.HentLejemaal()
    {
        var raaData = _dataFile.ReadAllLines().Skip(1).ToArray();
        var lejemaal = Konverter(raaData);
        return lejemaal;
    }

    protected Lejemaal DanLejemaalObjekt(string lejemaalData)
    {
        var lejemaalParts = lejemaalData.Split(';');
        if (lejemaalParts.Length < 3)
            throw new Exception("Filformat forkert ... ");
        //  ... parsing med TryParse og RemoveQuotes
    }
}
```

Constructor injection: IFile (adapter) injiceres

HentLejemaal(): læser linjer, springer header over, konverterer

DanLejemaalObjekt(): parser en enkelt linje til Lejemaal

protected metoder - testes via Stub (se næste slides)

Test 2: LejemaalFraFilRepository - HentLejemaal

Mock IFile → test at HentLejemaal parser CSV-data korrekt

```
[Fact]
public void Given_HentLejemaal_Faar_Korrekte_Datalinjer__Then_Liste_Dannes_Korrekt()
{
    // Arrange
    var lejemaalData = new[]
    {
        new string("\"lejlighednummer\"; \"kvadratmeter\"; \"antal rum\""),
        new string("\"101\"; \"755\"; \"3\""),
        new string("\"102\"; \"600\"; \"2\"")
    };

    var expected = new List<Lejemaal>
    {
        new() { AntalRum = 3, Kvadratmeter = 755, Lejlighednummer = 101 },
        new() { AntalRum = 2, Kvadratmeter = 600, Lejlighednummer = 102 }
    };

    var file = new Mock<IFile>();
    file.Setup(f ⟹ f.ReadAllLines()).Returns(lejemaalData);
    var service = new LejemaalFraFilRepository(file.Object) as ILejemaalRepository;

    // Act
    var actual = service.HentLejemaal();

    // Assert
    Assert.Equal(expected, actual);  // Record comparison!
}
```

Test 3: Fejlhåndtering med [Theory]

Test at forkerte datatyper i CSV-linjer kaster Exception

```
[Theory]
[InlineData("\"10x1\"; \"755\"; \"3\"")]       // Forkert int
[InlineData("\"10x1\"; \"755,0\"; \"3\"")]     // Forkert double
[InlineData("\"10x1\"; \"755,0\"; \"3.2\"")]   // Forkert int
public void Given_DanLejemaalObjekt_Faar_Forkerte_Datatyper__Then_Throw_Exception(
    string lejemaalLinje)
{
    // Arrange
    var file = new Mock<IFile>();
    file.Setup(f ⇒ f.ReadAllLines()).Returns(new string[1]);
    var service = new LejemaalFraFilRepositoryStub(file.Object);

    // Act & Assert
    Assert.Throws<Exception>(() ⇒ service.DanLejemaalObjekt(lejemaalLinje));
}
```

[Theory] med [InlineData] tester flere fejlscenarier i én testmetode

Assert.Throws<Exception> verificerer at en exception kastes

Bemærk: bruger Stub i stedet for direkte kald (protected metode)

Test 4: Forkert antal elementer i en linje

```
[Theory]
[InlineData("\"101\"; \"755\"")]  // Kun 2 elementer i stedet for 3
public void Given_DanLejemaalObjekt_Faar_Forkerte_Antal_Elementer__Then_Throw_Exception(
    string lejemaalLinje)
{
    // Arrange
    var file = new Mock<IFile>();
    file.Setup(f ⇒ f.ReadAllLines()).Returns(new string[1]);
    var service = new LejemaalFraFilRepositoryStub(file.Object);

    // Act & Assert
    Assert.Throws<Exception>(() ⇒ service.DanLejemaalObjekt(lejemaalLinje));
}
```

Tester at linjer med færre end 3 semikolon-separerede felter kaster Exception

Validering sker i DanLejemaalObjekt: if (lejemaalParts.Length < 3)

Vigtig edge case - hvad sker der med korrupte data?

Stub Pattern - Test af protected metoder

Problem: DanLejemaalObjekt() og Konverter() er protected

Løsning: En Stub-klasse der arver og "åbner" metoderne med new

```
public class LejemaalFraFilRepositoryStub : LejemaalFraFilRepository
{
    public LejemaalFraFilRepositoryStub(IFile dataFile) : base(dataFile)
    {
    }

    public new string RemoveQuotes(string lejemaalPart)
    {
        return base.RemoveQuotes(lejemaalPart);
    }

    public new List<Lejemaal> Konverter(string[] lejemaalData)
    {
        return base.Konverter(lejemaalData);
    }

    public new Lejemaal DanLejemaalObjekt(string lejemaalLinje)
    {
        return base.DanLejemaalObjekt(lejemaalLinje);
    }
}
```

"new" keyword skjuler basisklassens protected medlem og gør det public

base.Method() kalder den originale implementering

Test projekt konfiguration (.csproj)

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <OutputType>Exe</OutputType>
    <TargetFramework>net10.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="xunit.runner.json" CopyToOutputDirectory="PreserveNewest" />
  </ItemGroup>

  <ItemGroup>
    <Using Include="Xunit" />    <!-- Global using for xUnit -->
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="18.0.1" />
    <PackageReference Include="Moq" Version="4.20.72" />
    <PackageReference Include="xunit.v3.mtp-v2" Version="3.2.2" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Ejendomsberegner.Core\Ejendomsberegner.Core.csproj" />
  </ItemGroup>
</Project>
```

OutputType: Exe (kræves af xUnit v3)

Global using Xunit - ingen using Xunit; i hver testfil

Tre NuGet pakker: Test.Sdk, Moq, xunit.v3

ProjectReference til Core projektet

Kør tests

Fra kommandolinjen:

```
dotnet test
```

Fra Visual Studio / Rider:

Test Explorer: View → Test Explorer

Højreklik på test → Run Test

Grøn = passed, Rød = failed

Forventet resultat:

```
Passed!  - Failed: 0, Passed: 4, Skipped: 0, Total: 4
```

4 tests: 1x [Fact] ServiceTest + 1x [Fact] RepoTest + 2x [Theory] fejltest

Oversigt: Hvad mockes hvor?

Test klasse	SUT (System Under Test)	Mocked dependency	Hvad testes?
EjendomsberegnerServiceTest	EjendomBeregnerService	ILejemaalRepository	BeregnKvadratmeter summerer
LejemaalFraFilRepositoryTest	LejemaalFraFilRepository	IFile	HentLejemaal parser CSV
(Theory) Forkerte datatyper	LejemaalFraFilRepositoryStub	IFile	Exception ved forkert input
(Theory) Forkert antal felter	LejemaalFraFilRepositoryStub	IFile	Exception ved < 3 felter

Hvert lag mockes separat - dette er kernen i unit testing

Mock ILejemaalRepository → test beregning isoleret fra dataadgang

Mock IFile → test parsing isoleret fra filsystem

Opsummering - Hvad lærte vi?

Interfaces + DIP gør koden testbar

- Dependency Injection via constructor

Adapter Pattern abstraherer eksterne afhængigheder

- IFile indkapsler File.ReadAllLines()

Moq opretter mocks af interfaces

- Setup() → Returns() definerer mock-adfærd

xUnit v3 med [Fact] og [Theory]

- [InlineData] til parameteriserede tests

Record types giver værdi-baseret sammenligning

- Assert.Equal() virker direkte på records

Stub pattern til test af protected metoder

Arkitektur og interfaces er fundamentet for god testbarhed!