Ejendomsberegner - Unittest - Vejledende Løsning

Denne guide fører dig trin for trin fra opgavens udgangspunkt (`EjendomsberegnerTestBefore`) til den færdige løsning. Guiden forudsætter at du har gennemført opgave 02 (Interface og IoC) og har et fungerende projekt med interfaces, constructor injection og IoC.

Overblik: Hvad skal ændres?

Core-projektet og IoC-projektet er allerede færdige - du skal **kun** ændre i test-projektet (`Ejendomsberegner.Test`).

Der er fire steder i testkoden hvor der står `TODO`. Dem skal du løse:

Fil	TODO	Hvad mangler?
`EjendomsberegnerServiceTest.cs`	Setup SUT	Mock af `ILejemaalRepository`
`EjendomsberegnerServiceTest.cs`	Set actual	Kald `BeregnKvadratmeter()`
`LejemaalFraFilRepositoryTest.cs`	Setup SUT (HentLejemaal)	Mock af `IFile`
`LejemaalFraFilRepositoryTest.cs`	Setup SUT (Forkerte datatyper)	Mock af `IFile` til stub
`LejemaalFraFilRepositoryTest.cs`	Write test method	Hele test-metoden for forkert antal elementer

---

Trin 1: EjendomsberegnerServiceTest - Mock af ILejemaalRepository

Problemet

`EjendomBeregnerService` kræver en `ILejemaalRepository` i sin constructor. I udgangspunktet ser koden sådan ud:

```
// TODO: setup sut (System Under Test) here - remember to mock dependencies
var sut = new EjendomBeregnerService() as IEjendomBeregnerService;
```

Denne kode **kompilerer ikke** fordi `EjendomBeregnerService` ikke har en parameterløs constructor. Den kræver en `ILejemaalRepository`.

Tankegang

Vi vil teste om `BeregnKvadratmeter()` summerer korrekt. Vi vil **ikke** teste om filen læses korrekt - det er en anden test. Derfor mocker vi `ILejemaalRepository` så den returnerer en foruddefineret liste af lejemål.

Løsning

Erstat TODO-kommentaren med:

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

Linje for linje:

1. `new Mock<ILejemaalRepository>()` - opretter en mock af interfacet
2. `.Setup(repo => repo.HentLejemaal()).Returns(lejemaal)` - når nogen kalder `HentLejemaal()` på mock'en, returnér den testdata-liste vi har defineret i Arrange
3. `lejemaalRepo.Object` - giver den mockede instans som vi sender ind i constructoren
4. `as IEjendomBeregnerService` - cast er nødvendig fordi `BeregnKvadratmeter()` er implementeret eksplicit på interfacet

---

Trin 2: EjendomsberegnerServiceTest - Kald metoden

Problemet

```
// Act
//TODO: set actual to the result of the method being tested
```

Løsning

Erstat TODO-kommentaren med:

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

Den færdige testmetode

```
[Fact]
public void
Given_BeregnKvadratmeter_Faar_LejemaalListe__Then_Kvadratmeter_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);
}
```

Hvad tester vi? At `BeregnKvadratmeter()` summerer `Kvadratmeter` fra alle lejemål korrekt: 50.0 + 75.0 = 125.0.

Trin 3: LejemaalFraFilRepositoryTest - Mock af IFile (HentLejemaal)

Problemet

`LejemaalFraFilRepository` kræver en `IFile` i sin constructor:

```
// TODO: setup sut (System Under Test) here - remember to mock dependencies
var sut = new LejemaalFraFilRepository() as ILejemaalRepository;
```

Igen **kompilerer dette ikke** - der mangler et `IFile` argument.

Tankegang

Vi vil teste om `HentLejemaal()` parser CSV-data korrekt til `Lejemaal`-objekter. Vi vil **ikke** læse fra en rigtig fil - det ville gøre testen langsom og afhængig af filsystemet. Derfor mocker vi `IFile` så den returnerer foruddefinerede CSV-linjer.

Løsning

Erstat TODO-kommentaren med:

```
var file = new Mock<IFile>();
file.Setup(f => f.ReadAllLines()).Returns(lejemaalData);

var service = new LejemaalFraFilRepository(file.Object) as ILejemaalRepository;
```

Den færdige testmetode

```
[Fact]
public void
Given_HentLejemaal_Faar_Korrekte_Datalinjer__Then_Lejemaal_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);
}
```

Hvad tester vi? At CSV-linjer med header parses korrekt til `Lejemaal`-objekter. Bemærk at `Assert.Equal` virker her fordi `Lejemaal` er en `record` type, som sammenlignes på værdi.

Trin 4: LejemaalFraFilRepositoryTest - Mock af IFile til Stub (Forkerte datatyper)

Problemet

`LejemaalFraFilRepositoryStub` arver fra `LejemaalFraFilRepository`, som kræver `IFile` i sin constructor:

```
// TODO: setup sut (System Under Test) here - remember to mock dependencies
var service = new LejemaalFraFilRepositoryStub();
```

Hvorfor bruger vi en Stub her?

`DanLejemaalObjekt()` er en `protected` metode. Vi kan ikke kalde den direkte udefra. Stub-klassen arver fra `LejemaalFraFilRepository` og "åbner" metoden med `new` keyword, så vi kan kalde den i testen.

Løsning

Erstat TODO-kommentaren med:

```
var file = new Mock<IFile>();
file.Setup(f => f.ReadAllLines()).Returns(new string[1]);
var service = new LejemaalFraFilRepositoryStub(file.Object);
```

Vi mock'er `IFile` med en minimal setup (returnerer et tomt array) fordi vi ikke bruger `ReadAllLines()` i denne test - vi kalder `DanLejemaalObjekt()` direkte.

Den færdige testmetode

```
[Theory]
[InlineData("\"10x1\"; \"755\"; \"3\"")]
[InlineData("\"10x1\"; \"755,0\"; \"3\"")]
[InlineData("\"10x1\"; \"755,0\"; \"3.2\"")]
public void Given_DanLejemaalObjekt_Faar_Forkerte_Datatyper__Then_Thow_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));
}
```

Hvad tester vi? At `DanLejemaalObjekt()` kaster en `Exception` når den modtager data med forkerte datatyper (fx `"10x1"` som ikke kan parses til `int`).

Trin 5: LejemaalFraFilRepositoryTest - Test af forkert antal elementer

Problemet

Testmetoden er helt tom:

```
[Theory]
[InlineData("\"101\"; \"755\"")]
public void
Given_DanLejemaalObjekt_Faar_Forkerte_Antal_Elementer_I_En_Linje__Then_Thow_Exception(
    string lejemaalLinje)
{
    // TODO: write test method
    // Arrange


    // Act & Assert
}
```

Tankegang

Denne test ligner meget Trin 4. Vi tester at `DanLejemaalObjekt()` kaster en `Exception` når en linje kun har 2 elementer i stedet for de forventede 3. Koden i `DanLejemaalObjekt()` checker dette: `if (lejemaalParts.Length < 3) throw new Exception(...)`.

Løsning

Udfyld Arrange og Act & Assert sektionerne:

```
[Theory]
[InlineData("\"101\"; \"755\"")]
public void
Given_DanLejemaalObjekt_Faar_Forkerte_Antal_Elementer_I_En_Linje__Then_Thow_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));
}
```

Hvad tester vi? At en CSV-linje med kun 2 felter (`"101"; "755"`) kaster en `Exception` fordi der forventes 3 felter (lejlighednummer, kvadratmeter, antal rum).

Kør tests

Når alle fem trin er gennemført, kan du køre tests:

```
dotnet test
```

Forventet resultat:

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

De 4 tests er:

1. `Given_BeregnKvadratmeter_Faar_LejemaalListe__Then_Kvadratmeter_Beregnes_Korrekt` (Fact)
2. `Given_HentLejemaal_Faar_Korrekte_Datalinjer__Then_Lejemaal_Liste_Dannes_Korrekt` (Fact)
3. `Given_DanLejemaalObjekt_Faar_Forkerte_Datatyper__Then_Thow_Exception` (Theory, 3 test cases)
4. `Given_DanLejemaalObjekt_Faar_Forkerte_Antal_Elementer_I_En_Linje__Then_Thow_Exception` (Theory, 1 test case)

Opsummering: Mønstret i alle tests

Alle tests følger det samme mønster:

1. **Identificér afhængigheden** - hvilken interface skal mock'es?
2. **Opret mock** - `new Mock<IInterface>()`
3. **Setup mock** - `.Setup(...).Returns(...)` med testdata
4. **Opret SUT** - send `mock.Object` til constructoren
5. **Act** - kald metoden
6. **Assert** - verificér resultatet

Test	SUT	Mock'et interface	Hvad returnerer mock'en?
BeregnKvadratmeter	`EjendomBeregnerService`	`ILejemaalRepository`	Liste af Lejemaal-objekter
HentLejemaal	`LejemaalFraFilRepository`	`IFile`	Array af CSV-strenge
Forkerte datatyper	`LejemaalFraFilRepositoryStub`	`IFile`	Tomt array (bruges ikke)
Forkert antal felter	`LejemaalFraFilRepositoryStub`	`IFile`	Tomt array (bruges ikke)

Det er Dependency Inversion Principle der gør dette muligt: fordi alle afhængigheder er interfaces, kan vi erstatte dem med mocks i tests.