

Module 26:

"Null Object"

(with Unit Testing)



TEKNOLOGISK
INSTITUT

Agenda

- ▶ Introductory Example: Animal Sounds
- ▶ Challenges
- ▶ Pattern: Null Object
- ▶ Implementing the Null Object Pattern
- ▶ Overview of Null Object
- ▶ Background: Unit Testing
- ▶ Null Object in Unit Testing

Introductory Example: Animals Sounds

```
class AnimalFactory : IAnimalFactory
{
    public IAnimal Create( string description )
    {
        ...

        if ( _animalTypes.TryGetValue(processedDescription,
            out Type animalType))
        {
            return Activator.CreateInstance(animalType) as IAnimal;
        }
        return null;
    }
    ...
}
```

```
interface IAnimal
{
    string Name { get; }
    void MakeSound();
}
```

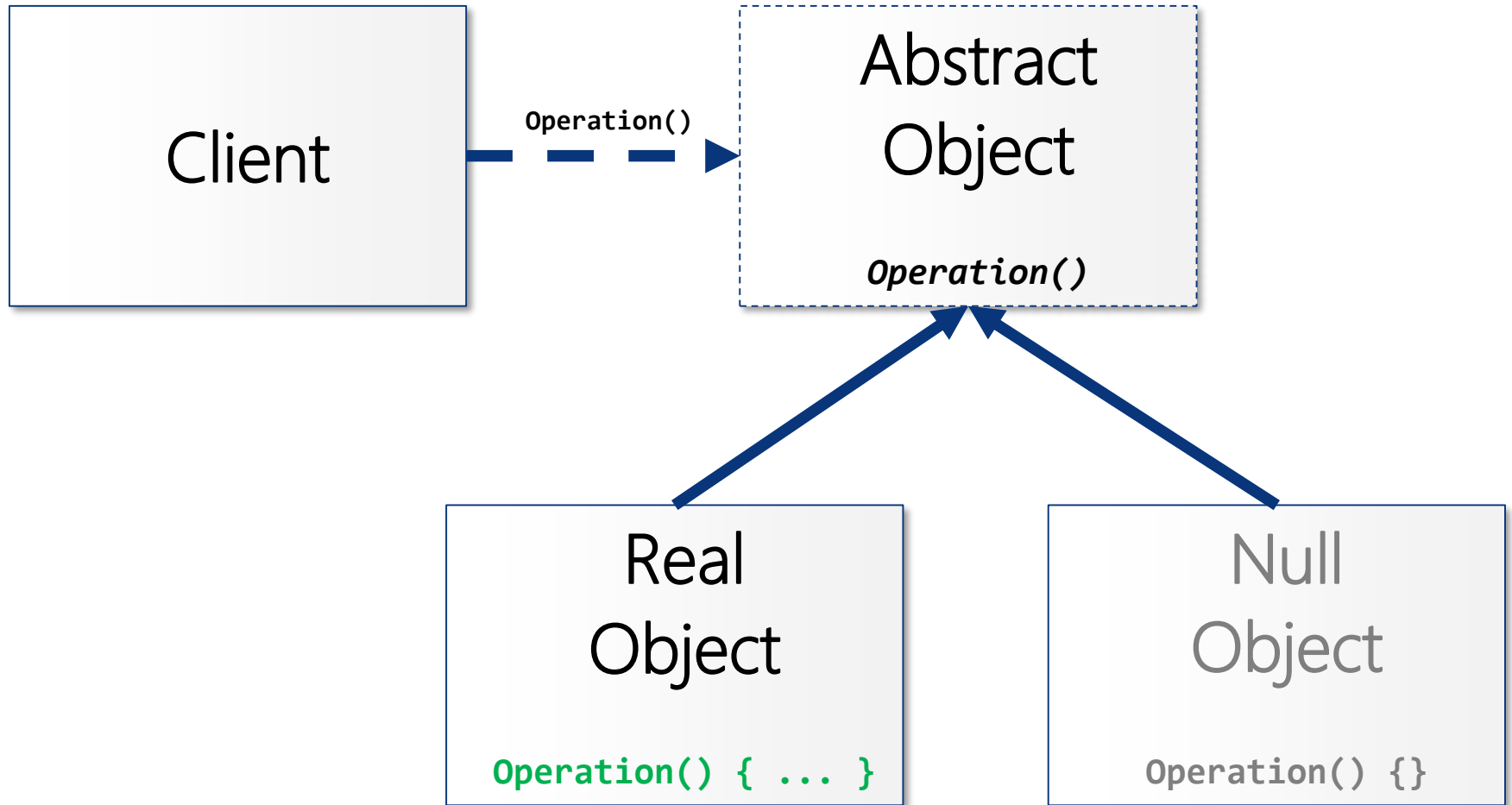
Challenges

- ▶ C# has specialized syntax for null-checks, but could we relieve the client of that burden?
- ▶ What if a component needs an object to compile and run, but during unit tests that object should be "inactive"?

Pattern: Null Object

- ▶ *Provide an object as a surrogate for the lack of an object of a given type. The Null Object provides intelligent "do-nothing" behavior, hiding the details from its collaborators.*
- ▶ Outline
 - Abstract the handling of null references away from the client
 - Create an object with do-nothing behavior in a well-defined interface expected by the client
- ▶ Origin: Bobby Woolf (1998)

Overview of Null Object Pattern



Overview of Null Object Pattern

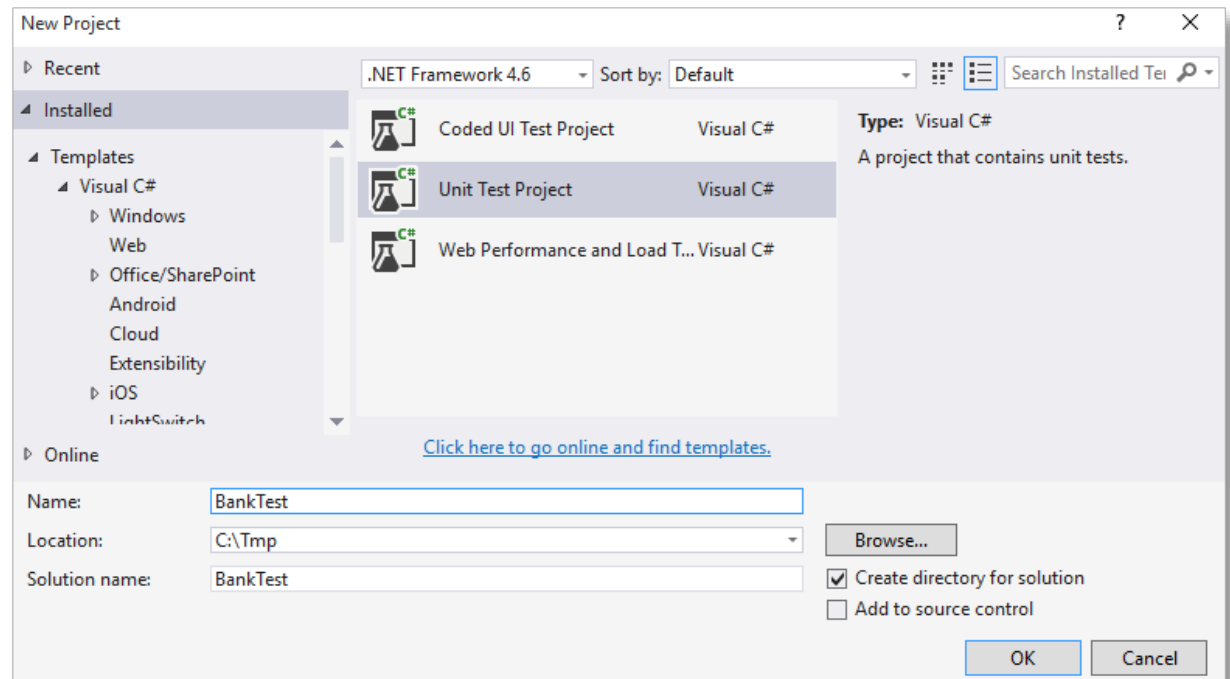
- ▶ Client
 - Needs a collaborator exposing **Operation()**
- ▶ Abstract Object
 - Interface or abstract class specifying the abstract **Operation()**
- ▶ Real Object
 - Concrete class implementing the Abstract Object interface
 - Supplied appropriate behavior in **Operation()** used by Client
- ▶ Null Object
 - Concrete class implementing the Abstract Object interface
 - Can be substituted for Real Object in the context of Client
 - Implements the **Operation()** to do nothing / neutral behavior
 - The exact neutral behavior depends on what Client expects

Background: Unit Testing

- ▶ Automatic “white-box” testing of classes
 - Developers can run tests in Visual Studio
 - Tests can run automatically when code is checked in
- ▶ Unit tests
 - Captures a code-based “specification” of functionality
 - Drives safe refactoring
 - Assists regression testing
- ▶ Unit testing frameworks for C# include
 - MSTest
 - NUnit
 - MbUnit
 - xUnit.net
 - ...

Unit Testing in Visual Studio

- ▶ Visual Studio includes MSTest
 - Unit Test Project



- ▶ Create business logic project(s) “as usual”
- ▶ Create Unit Test Project with a reference to business logic project(s)
- ▶ Author test classes and methods in test project



Test Classes and Test Methods

- ▶ Test methods must be marked with the `[TestMethod]` attribute
 - Cannot have parameters and returns `void`, `Task`, or `Task<T>`

```
[TestClass]
public class BankAccountTest
{
    [TestMethod]
    public void TestDeposit()
    {
        BankAccount account = new BankAccount();
        account.Deposit(87);
        Assert.AreEqual(87, account.Balance);
    }
}
```

- ▶ Test classes must be marked with the `[TestClass]` attribute



Using the **Assert** Class and Attributes

- ▶ The **Microsoft.VisualStudio.TestTools.UnitTesting** namespaces includes e.g.
 - **Assert.**
 - `AreEqual()`
 - `AreNotEqual()`
 - `Fail()` ...
 - `[ExpectedException]` attribute

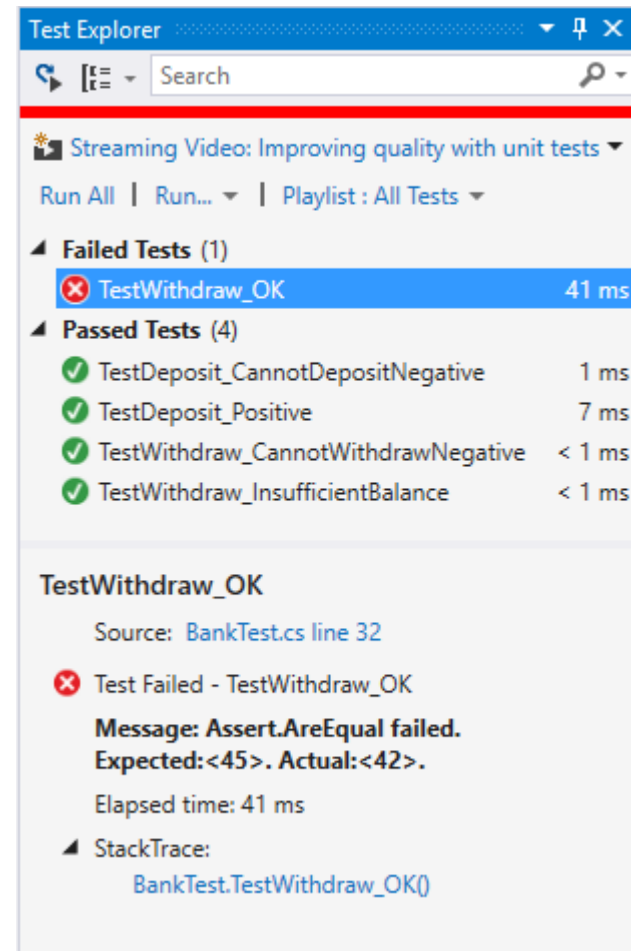
```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
public void TestWithdraw()
{
    BankAccount account = new BankAccount();
    account.Withdraw(87);
}
```

- `[TestInitialize]` + `[TestCleanup]` attributes



Running the Tests

- ▶ Test Explorer
 - Test -> Windows -> Test Explorer
- ▶ Status annotations in source code in editor



Code Coverage Analyzer

- ▶ Some versions of Visual Studio have additional testing tools
 - Test > Analyze Code Coverage > All Tests
 - ...

Code Coverage Results				
jespe_DESKTOP-IO4GN8M 2015-10-05 22_52				
Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
jespe_DESKTOP-IO4GN8M 2015-...	3	7,50 %	37	92,50 %
bank.dll	0	0,00 %	18	100,00 %
{ } Bank	0	0,00 %	18	100,00 %
BankAccount	0	0,00 %	18	100,00 %
Deposit(double)	0	0,00 %	6	100,00 %
Withdraw(double)	0	0,00 %	10	100,00 %
get_Balance()	0	0,00 %	1	100,00 %
set_Balance(double)	0	0,00 %	1	100,00 %
banktest.dll	3	13,64 %	19	86,36 %



Test-Driven Development (TDD)

- ▶ Test-Driven Development (TTD)
 - Write unit tests before class itself
 - Class is complete when all units tests pass
 - Additional features and/or bug fixes incur yet more unit tests etc.
- ▶ Visual Studio has “TDD-friendly” IntelliSense mode
 - **CTRL-ALT-Space** toggles between modes



Null Object in Unit Testing

- ▶ Null Objects are extremely useful in unit testing
 - "mock", "stub", ...

```
private class NullLogger : ILogger
{
    public void Enter( string callerMemberName ) { }
    public void Error( string message ) { }
    public void Error( Exception exception ) { }
    public void Exit( string callerMemberName = null ) { }
    public void Info( string message ) { }
    public void Info( Exception exception ) { }
}
```

- ▶ Null objects and factories can be set up in `[TestInitialize]`



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Hasselvangel 243

8355 Solbjerg

Denmark