Module 3:
"Managing Dependencies"

TEKNOLOGISK
INSTITUT

Agenda

- **Classifying Dependencies**
- Dependency Injection Patterns
- Workshop A.4: A Test of Time in the Domain Layer
- Anti-Pattern: Ambient Context
- *Pattern: Null Object*
- Test Spies
- (Optional) Workshop A.5: Create a Null User Context

Volatile Dependencies

- Out-of-process or unmanaged resources
- Nondeterministic resources
- Resources to be
  - Replaced
  - Intercepted
  - Decorated
  - Mocked

Examples of Volatile Dependencies

- Databases
- File system
- Web services
- Security contexts
- Message Queues
- **`System.Random`** (or similar)

Stable Dependencies

- A dependency is *stable* if it's not volatile...!

```
interface IUserRoleParser
{
    bool Parse(string role);
}
```

```
class MovieViewModel
{
    public string Name { get; }
    public string DisplayText { get; }

    public MovieViewModel(MovieShowing movie) { ... }
}
```

TEKNOLOGISK
INSTITUT

To Inject or Not To Inject?

Dependency Injection applies exclusively to Volatile Dependencies.

Don't inject Stable Dependencies!

Never Make Dependencies Optional!

- You should usually always require a dependency
  - Only exception is if it has a Local Default
  - If it is not present (yet?) then use a Null Object
- Local Default
  - A default implementation of a dependency which resides in the same module or layer of the application
  - (As opposed to Foreign Default)
- Note:
  - A Local Default cannot have any Foreign Defaults

Agenda

- Classifying Dependencies
- **Dependency Injection Patterns**
- Workshop A.4: A Test of Time in the Domain Layer
- Anti-Pattern: Ambient Context
- *Pattern: Null Object*
- Test Spies
- (Optional) Workshop A.5: Create a Null User Context

Pattern: Composition Root

- *The Composition Root is the single, logical location in your application where the modules and dependencies are composed together.*

- Outline
  - This is as near as possible to the application's entry point
  - If using a DI container this is the only place where you reference the container

- See:
  "Dependency Injection Principles, Practices, and Patterns"
  Steven van Deursen and Mark Seemann (2019)

Examples of Composition Root

- Console applications
  - `Main()` in `Program.cs`
- WPF / UWP applications
  - `App.xaml.cs`
- ASP.NET applications
  - `Application_Start()` in `Global.asax.cs`

  + Setting the `DependencyResolver` for controllers

Pattern: Constructor Injection

- *Constructor Injection is the act of statically defining the list of required dependencies by specifying them as parameters to the class' constructor.*

- Outline
  - Explicitly list the dependencies needed by the component
  - Keep the constructor free of any other logic
  - Note: This is always the preferred choice of injection

- See:
  "Dependency Injection Principles, Practices, and Patterns"
  Steven van Deursen and Mark Seemann (2019)

Constructor Injection

- Advantages
  - Dependencies are always present when object constructed.
  - Make the dependencies needed by the component obvious in a declarative manner
  - Very easy to implement

- Disadvantages
  - Does not work well with late-binding frameworks with Constrained Constructions

Pattern: Method Injection

- *Method Injection supplies a consumer with a dependency by passing it as method argument on a method called outside the Composition Root.*

- Outline
  - Used when the dependency is short-lived or varies

- See:
  "Dependency Injection Principles, Practices, and Patterns"
  Steven van Deursen and Mark Seemann (2019)

Method Injection

- Advantages
  - Allows dependencies to be injected into data objects which are not created in the Composition Root
  - Allows the caller to provide operation-specific context
- Disadvantages
  - Rarely needed or used
  - Passing dependencies in methods can make dependency part of API

Pattern: Property Injection

- *Property Injection allows a Local Default to be replaced via a public settable property. Property Injection is also known as Setter Injection.*

- Outline
  - Best used with optional dependencies
  - Allows Local Default to be overridden in a simple way

- See:
  "Dependency Injection Principles, Practices, and Patterns"
  Steven van Deursen and Mark Seemann (2019)

Example of Property Injection

```
public class Consumer
{
    public IDependency Dependency { get; set; }
    public void UseDependency()
    {
        Dependency.DoStuff();
    }
}
```

```
Consumer consumer = new Consumer();
consumer.Dependency = new Implementation1();
consumer.UseDependency();
consumer.Dependency = new Implementation2();
consumer.UseDependency();
```

Property Injection

- Advantages
  - Easy to understand..! ☺
- Disadvantages
  - Causes temporal coupling
  - Can change during lifetime (unless subtle logic is implemented)
  - Very limited applicability, e.g. reusable libraries

Agenda

- Classifying Dependencies
- Dependency Injection Patterns
- **Workshop A.4: A Test of Time in the Domain Layer**
- Anti-Pattern: Ambient Context
- *Pattern: Null Object*
- Test Spies
- (Optional) Workshop A.5: Create a Null User Context

TEKNOLOGISK
INSTITUT

Workshop A.4:
A Test of Time in the Domain Layer

TEKNOLOGISK INSTITUT

Discussion: Possible Approaches?

- Time is one of the most common sources of errors

- Which solution did you implement?
- Why?

Agenda

- Classifying Dependencies
- Dependency Injection Patterns
- Workshop A.4: A Test of Time in the Domain Layer
- **Anti-Pattern: Ambient Context**
- *Pattern: Null Object*
- Test Spies
- (Optional) Workshop A.5: Create a Null User Context

Anti-Pattern: Ambient Context

- *An Ambient Context supplies application code outside the Composition Root with global access to a Volatile Dependency by means of static members.*

```
ITimeProvider timeProvider = TimeProvider.Current;
DateTime now = timeProvider.Now;
```

- See:
  "Dependency Injection Principles, Practices, and Patterns"
  Steven van Deursen and Mark Seemann (2019)

Agenda

- Classifying Dependencies
- Dependency Injection Patterns
- Workshop A.4: A Test of Time in the Domain Layer
- Anti-Pattern: Ambient Context
- ***Pattern: Null Object***
- Test Spies
- (Optional) Workshop A.5: Create a Null User Context

Agenda

- Classifying Dependencies
- Dependency Injection Patterns
- Workshop A.4: A Test of Time in the Domain Layer
- Anti-Pattern: Ambient Context
- *Pattern: Null Object*
- **Test Spies**
- (Optional) Workshop A.5: Create a Null User Context

Test Spies

- Test Spy
  - An object that records its interaction with other objects
  - When deciding if a test was successful based on the state of available objects alone is not sufficient
  - Can be useful for making assertions on things e.g.
    - the number of calls
    - elements sent
    - arguments passed to specific functions, and return values
- Related to Null Object, but different!

Agenda

- Classifying Dependencies
- Dependency Injection Patterns
- Workshop A.4: A Test of Time in the Domain Layer
- Anti-Pattern: Ambient Context
- *Pattern: Null Object*
- Test Spies
- **(Optional) Workshop A.5: Create a Null User Context**

TEKNOLOGISK INSTITUT

(Optional) Workshop A.5: Create a Null User Context

Discussion: What Have We Learned?

- Cascading change adding NullUserContext instance?
- What was unit tested? And how?
- What kinds of DI Patterns were applied?

- Is the new design still maintainable?
- Does SOLID enhance team collaboration?

- How would realistic user contexts be implemented?

Summary

- Classifying Dependencies
- Dependency Injection Patterns
- Workshop A.4: A Test of Time in the Domain Layer
- Anti-Pattern: Ambient Context
- *Pattern: Null Object*
- Test Spies
- (Optional) Workshop A.5: Create a Null User Context


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Hasselvangen 243
8355 Solbjerg
Denmark