Leveling Up Dependency Injection in .NET

4: Common Stumbling Blocks

Jeremy Clark
www.jeremybytes.com
@jeremybytes

Common Stumbling Blocks

- Constructor Over-Injection
- Static Dependencies
- Dealing with IDisposable (and other lifetime concerns)
- Using Factory Methods
- Configuration Strings

Constructor Over-Injection

- Symptom: a constructor contains a large number of parameters.
- Code Smell: this often points to a violation of the Single Responsibility Principle

Constructor Over-Injection

- Possible Solution:
  Break up the class along the functionality lines. This generally results in object groupings and dependencies that are more manageable in size.

Constructor Over-Injection

- Possible Solution:
  Create Parameter Objects.
- A parameter object can combine multiple dependencies into a single parameter. This allows grouping of parameters along functional lines.

Static Dependencies

- Symptom: A class relies on a static object as a dependency.
- Problem: This makes it difficult to swap out functionality for testing.
- Example: DateTime.Now()

Static Dependencies

- Possible Solution:
  Instead of relying on the static object directly, a class can wrap that dependency in a property. By default, the static dependency will be used, but it's possible to provide a different implementation for testing or other purposes.

Dealing with IDisposable

- Symptom: a dependency implements IDisposable.
- Code Smell: This is a leaky abstraction. The requirement to dispose of the object "leaks" out; the consuming class needs to know this about the dependency.

Dealing with IDisposable

- Possible Solution:
  Create a proxy class to wrap the functionality.
- Each method call creates the underlying object inside a “using”, the makes the call.
- The object is disposed and resources released.

- Example: SQL Repository

Factory Methods

- Symptom: A class uses a factory method and has a private constructor.
- Problem: This breaks auto-wiring in DI containers.

- Solution: We'll take a closer look after exploring DI containers further.

Factory Methods

- Symptom: A class uses a factory method and has a private constructor.
- Problem: This breaks auto-wiring in DI containers.

- Solution: We'll take a closer look after exploring DI containers further.

Configuration Strings

- Symptom: A class constructor needs a string as a parameter, such as a connection string.
- Problem: This breaks auto-wiring in DI containers.

- Solution: We'll take a closer look after exploring DI containers further.

Primary Benefits

- Late Binding
- Extensibility
- Parallel Development
- Maintainability
- Testability

- Adherence to S.O.L.I.D. Design Principles.

Dependency Injection Concepts

- DI Design Patterns
  - Constructor Injection
  - Property Injection
  - Method Injection
  - Ambient Context
  - Service Locator
- Dimensions of DI
  - Object Composition
  - Interception
  - Lifetime Management

More Information

https://github.com/jeremybytes/di-dotnet-workshop