Design Pattern & Clean Code

Dependency Injection

Thomas Ley | @CleanCodeCoach

[DIP]

❗ The Dependency Inversion Principle (DIP) states that high level modules should not depend on low level modules; both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions. ❗

Goals

- DI & SL differences
- Lifetime scopes
- Factory Pattern

DI vs. SL

- [Dependency injection] (DI)
  - Injects dependencies
  - Contructor/property injection
- [Service Locator] (SL)
  - Single point of contact
  - Static dependency resolver.

Demo

- Project EUMEL Dj
- Mobile app uses service locator
- Desktop app uses dependency injection.

DI as SL

- Inject DI container
- Resolve dependency from container.

Lifetime Scopes

- Unique-Instance context or scope
- Example
  - Per process
  - Per thread
  - Per HTTP request
  - Any customer defined
- DI framework has already implementations.

[Singleton]

- Instance is created once
- Instance is created on first use
- [Double-Check Locking]
- MSDN Documentation
- Implementation see Lazy<T>.

Demo

- Project src/Zapfenstreich.sln

AD: `DI Frameworks`

- Reduces "hard dependencies"
- Delegates creation
- Simplifies injecting of code
- Simplifies changing of implementation

☝ A DI container makes you write cleaner software
👍 A DI container helps refactoring code.

Factories

- Creates an *implemenentation*
- Returns an *interface*.

[Factory] Implementations

- Class with `Create()` method(s)
- Interface with `Create()` method(s)
- Func<T> / Lambda.

Lazy<T> vs. Func<T>

- Func<T> is a method which creates a T
- Lazy<T> implements a singleton
- Lazy gets a Func as constructor parameter
- Lazy can solve circular (DI) dependencies.

Demo

- Project