Leveling Up Dependency Injection in .NET

2: A Closer Look at DI

Jeremy Clark
www.jeremybytes.com
@jeremybytes

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

Deeper Dive

- Dimensions of Dependency Injection
- Programming to an Abstraction
- SOLID Principles

Tips / Techniques

- Read-Only Properties (for Constructor Injection)
- Guard Clauses (prevent unintended nulls)

Dimensions of Dependency Injection

- Object Composition
  - Snapping loosely coupled pieces together
- Lifetime Management
  - Managing creation and re-use of objects.
  - Transient, Singleton, Scoped, Thread
- Interception
  - Adding or replacing functionality in method calls

Programming to an Abstraction

- DI is made possible by programming against abstractions. When we program against an interface or base-class, we can swap out functionality by providing objects that implement the methods and properties of the base object.
- Abstractions provide the seams that make DI possible.

Benefits – SOLID Principles

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Single Responsibility Principle

Each class should have only
one reason to change.

Single Responsibility Principle

Cohesion:

The functional relatedness of elements in a class or module. The lower the amount of cohesion, the higher the chance a class violates the Single Responsibility Principle.

Open/Closed Principle

A class should be open for extension,
but closed for modification.

Open/Closed Principle

Application design prevents us from having to make sweeping changes throughout the code base.

There is strong relationship between the Open/Closed Principle and the DRY principle (Don't Repeat Yourself).

Liskov Substitution Principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of the program.

Liskov Substitution Principle

Every Dependency should behave as defined by its Abstraction.

Interface Segregation Principle

Clients should not be forced
to depend upon methods that
they do not use. Interfaces belong
to clients, not hierarchies.

Interface Segregation Principle

Many client-specific interfaces are better than one general-purpose interface.

Any time a consumer depends on an Abstraction where some of the members stay unused, this principle is violated.

Dependency Inversion Principle

Abstractions should not depend upon details; details should depend upon abstractions.

Dependency Inversion Principle

We should program against abstractions, and the consuming layer should be in control over the shape of a consumed abstraction.

Tips / Techniques

- Read-Only Properties (for Constructor Injection)
- Guard Clauses (prevent unintended nulls)

Read-Only Properties

- Properties marked as “readonly” are settable only in the constructor. This prevents the property from being inadvertently changed during the lifetime of the object.
- This is applicable to Constructor Injection; for obvious reasons, this would be a problem for Property Injection.

Guard Clauses

- Guard clauses (null checks) should be used in constructors, methods, and property setters to ensure that dependencies are not set to null.
- If a “null behavior” is required, consider using the Null Object pattern. This provides a valid implementation with no actual behavior.

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