

SOLID DESIGN PATTERNS FOR MERE MORTALS

Philip Japikse (@skimedic)

skimedic@outlook.com

www.skimedic.com/blog

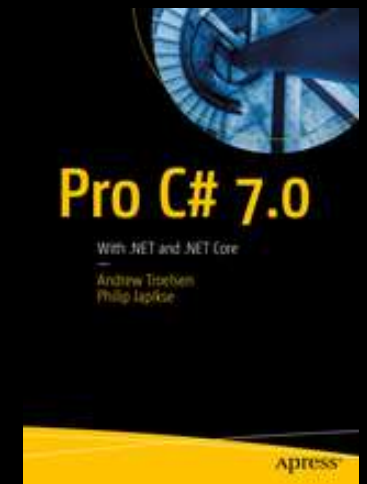
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM II, PSD

Consultant, Teacher, Writer



Phil.About()

- Director of Consulting/Chief Architect
- Author: Apress.com (<http://bit.ly/apressbooks>)
- Speaker: <http://www.skimedic.com/blog/page/Abstracts.aspx>
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM II, PSD
- Founder, Agile Conferences, Inc.
 - <http://www.cincydeliver.org>
- President, Cincinnati .NET User's Group



A LOOK AT SOLID

SINGLE RESPONSIBILITY PRINCIPLE

➤ Do one thing and do it well!



<http://joshlinkner.com/images/2012/05/SAN.jpg>

All slides copyright Philip Japikse <http://www.skimedic.com>

OPEN CLOSED PRINCIPLE

- Be Open for Extension,
Closed for Modification



http://www.wellgolly.com/images/WWTT_house.jpg

All slides copyright Philip Japikse <http://www.skimedic.com>

LISKOV SUBSTITUTION PRINCIPLE

- Derived Classes Can Stand In for Base Classes



<http://beerimages.com/wp-content/uploads/2011/03/beer-collection.jpg>

INTERFACE SEGREGATION PRINCIPLE

- Make Interfaces
 - Fine Grained and
 - Client Specific



DEPENDENCY INVERSION

- Depend On Abstractions,
Not Concrete Implementations



ADDITIONAL CONSIDERATIONS

DON'T REPEAT YOURSELF (DRY)

- Clip-board Inheritance is an anti-pattern!



THE BOY SCOUT PRINCIPLE

- Clean up after yourself
- Clean up after others



YAGNI

➤ You Ain't Gonna Need It



<http://www.k-photography.info/srvgdata-gold-plated-toilets.asp>

SEPARATION OF CONCERNS

- Focusing one's attention upon some aspect – Edsger Dijkstra



<https://sf.curbed.com/2017/3/10/14889950/kitchen-bathroom-sf-combined-toilet>

DESIGN PATTERNS

MOTIVATION FOR DESIGN PATTERNS

“The goal is not to bend developers to the will of some specific patterns, but to get them to think about their work and what they are doing”

--Phil Haack

WHAT ARE DESIGN PATTERNS?

- General Reusable Solutions To A Common Problem
- Conceptual
- Defined by Purpose and Structure
- Method of Communication
- Support SOLID development
- NOT CODE!

TYPES OF DESIGN PATTERNS

➤ Creational

- Deal with instantiation of objects (Singleton, Factories, Prototype)

➤ Structural

- Deal with Composition and Relations (Adapter, Façade, Decorator)

➤ Behavioral

- Deal with responsibilities and communication between objects (Command, Strategy, Observer, Pub-Sub, Memento, Template Method)

CREATIONAL DESIGN PATTERNS

CREATIONAL

➤ Singleton

- Ensures class has only one instance with a single access point

➤ Simple Factory (Not a “true” pattern)

- Encapsulates object creation in one place

➤ Factory Method

- Uses methods to create objects without specifying the exact class

➤ Abstract Factory

- Encapsulates a group of individual factories with a common theme without specifying their concrete class

THE SINGLETON PATTERN

- The **singleton pattern** is a software design pattern that restricts the instantiation of a class to one instance and provides global access to that instance.
- Commonly used to implement many other patterns:
 - Factories (abstract and simple), façade, state, builder, and prototype
- Many DI frameworks provide singleton support
 - You need to understand how the DI f/w works to prevent issues

THE SIMPLE FACTORY PATTERN

- Not a “true” pattern
- Encapsulates object creation in one place
 - Should be the only part of the application that refers to concrete classes
- Reduces duplicate code by enforcing DRY

THE FACTORY METHOD PATTERN

- The *factory method pattern* uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.
- This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

THE ABSTRACT FACTORY PATTERN

- The **abstract factory pattern** provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes.
- In normal usage, the client software creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create the concrete objects that are part of the theme.
- This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface.

STRUCTURAL DESIGN PATTERNS

STRUCTURAL

➤ Adapter

- Converts the interface of a class into another interface the client expects

➤ Façade

- Provides a simplified interface to a larger body of code

➤ Decorator

- Attaches additional responsibilities to an object at runtime without effecting othe objects of the same class

THE ADAPTER AND FAÇADE PATTERNS

- The **adapter pattern** allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying code.
- A **facade** is an object that provides a simplified interface to a larger body of code, such as a class library. A facade can:
 - Make a software library easier to use, understand, and test, since the facade has convenient methods for common tasks,
 - Reduce dependencies of outside code on the inner workings of a library
 - Wrap a poorly designed collection of APIs with a single well-designed API.

THE DECORATOR PATTERN

- The **decorator pattern** that allows behavior to be added to an individual object, either statically or dynamically, without affecting the behavior of other objects from the same class.
- Provides an alternative to subclassing for extending functionality.
- Supports the Single Responsibility Principle, as it allows functionality to be divided between classes with unique areas of concern.

BEHAVIORAL DESIGN PATTERNS

BEHAVIORAL

- Command
 - Encapsulates a request as an object
- Strategy
 - Encapsulates an algorithm inside a class

THE COMMAND PATTERN

- The **command pattern** is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time.
- Can optionally add Undo
 - Tracked by either the command or the controller
- Used in conjunction with many other patterns:
 - Factory method, Template method

THE STRATEGY PATTERN

- The **strategy pattern** (also known as the **policy pattern**) is a behavioral software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern:
 - defines a family of algorithms,
 - encapsulates each algorithm, and
 - makes the algorithms interchangeable within that family.

- Promotes the Open/Closed principle by using Composition over Inheritance

- Examples:
 - Sorting (with custom comparer)
 - Log4Net Fallback Appender

RESOURCES

- “Design Patterns: Elements of Reusable Object Oriented Design”
 - Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides
- “Head First Design Patterns”
 - Freeman, Robson, Bates, Sierra
- Eight part series with Robert Green on Visual Studio Toolbox
 - <https://channel9.msdn.com/Shows/Visual-Studio-Toolbox/Design-Patterns-CommandMemento> (first of the series)

DEMO DEWNO

Patterns in C#

Contact Me

skimedic@outlook.com

www.skimedic.com/blog

www.twitter.com/skimedic

<http://bit.ly/skimediclyndacourses>

<http://bit.ly/apressbooks>

www.hallwayconversations.com

Questions?



Thank You!

Questions?