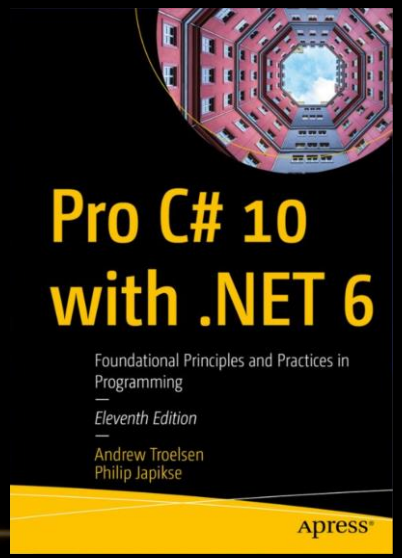
SOLID DESIGN PATTERNS FOR MERE MORTALS

Philip Japikse (@skimedic)
skimedic@outlook.com
CTO, Author, Teacher
Microsoft MVP, ASPInsider,
PST, PSM II, PSD, PSPO, PSK

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

Phil.About()

- CTO/Chief Architect, Pintas & Mullins
- Author: Apress.com (http://bit.ly/apressbooks)
- Professional Scrum Trainer (PSF, PSD)
- Speaker: http://www.skimedic.com/blog/page/Abstracts.aspx
- Microsoft MVP, ASPInsider, PST, PSM II, PSD
- Founder, Agile Conferences, Inc.
  - http://www.cincydeliver.org
- President, Cincinnati .NET User's Group



https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

A LOOK AT SOLID

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

SINGLE RESPONSIBILITY PRINCIPLE

- Do one thing and do it well!

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

OPEN CLOSED PRINCIPLE

- Be Open for Extension, Closed for Modification

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

LISKOV SUBSTITUTION PRINCIPLE

- Derived Classes Can Stand In for Base Classes

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

INTERFACE SEGREGATION PRINCIPLE

- Make Interfaces
  - Fine Grained and
  - Client Specific

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

DEPENDENCY INVERSION

- Depend On Abstractions, Not Concrete Implementations

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

ADDITIONAL CONSIDERATIONS

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

DON'T REPEAT YOURSELF (DRY)

- Clip-board Inheritance is an anti-pattern!

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

THE BOY SCOUT PRINCIPLE

- Clean up after yourself
- Clean up after others

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

YAGNI

- You Ain’t Gonna Need It

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

SEPARATION OF CONCERNS

- It's time to unmask the computing community as a Secret Society for the Creation and Preservation of Artificial Complexity – Edsger Dijkstra

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

FIX THE WINDOWS

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

DESIGN PATTERNS

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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!

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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)

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

CREATIONAL DESIGN PATTERNS

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

https://en.wikipedia.org/wiki/Singleton_pattern

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

THE FACTORY METHOD PATTERN

- The *factory method pattern* defines an interface for creating an object, but lets derived classes decide what to instantiate.
- Derived classes can employ a simple factory to instantiate the class specific types

https://en.wikipedia.org/wiki/Factory_method_pattern
https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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.
- 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.

https://en.wikipedia.org/wiki/Abstract_factory_pattern

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

STRUCTURAL DESIGN PATTERNS

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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 other objects of the same class

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

THE ADAPTER PATTERN

- 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.

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

https://en.wikipedia.org/wiki/Adapter_pattern

https://en.wikipedia.org/wiki/Facade_pattern

THE FAÇADE PATTERN

- 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.

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

https://en.wikipedia.org/wiki/Adapter_pattern

https://en.wikipedia.org/wiki/Facade_pattern

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.

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

https://en.wikipedia.org/wiki/Decorator_pattern

BEHAVIORAL DESIGN PATTERNS

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

BEHAVIORAL

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

https://en.wikipedia.org/wiki/Command_pattern

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

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

https://en.wikipedia.org/wiki/Strategy_pattern

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)

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns

Contact Me

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic

http://bit.ly/apressbooks

Questions?

Thank You!

https://github.com/skimedic/presentations/tree/main/Patterns/6.0/DesignPatterns