



# Design Patterns for Mere Mortals

By  
Philip Japikse  
Phil.japikse@pinnsng.com  
MVP, MCSD.Net, MCDBA, CSM, CSP  
Principal Consultant  
Pinnacle Solutions Group



# Who am I?

- ▶ Principal Consultant, Pinnacle Solutions Group
- ▶ Microsoft MVP
- ▶ MCSD, MCDBA, CSM, CSP
- ▶ Enterprise Application Architect
- ▶ Trainer/Mentor/Speaker
- ▶ Lead Director, Cincinnati .NET User's Group
- ▶ Contributing Author – [www.nplus1.org](http://www.nplus1.org)

**“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**

- ▶ **S – Single Responsibility Principle**
  - There should never be more than one reason for a class to change
- ▶ **O – Open/Closed Principle**
  - Classes should be Open for extension, Closed for modification
- ▶ **L – Liskov Substitution Principle**
  - Derived classes should be substitutable for their base classes
- ▶ **I – Interface Segregation Principle**
  - Make fine grained interfaces that are client specific
- ▶ **D – Dependency Inversion Principle**
  - Depend on abstractions, not concrete implementations

# What are Design Patterns?

- ▶ General Reusable Solutions To A Common Problem
  - Conceptual
  - Templates
  - 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
- ▶ **Structural**
  - Deal with Composition and Relations
    - Adapter, Decorator, Facade
- ▶ **Behavioral**
  - Deal with responsibilities and communication between objects
    - Command, Strategy

# Creational

- ▶ Singleton
  - Ensures class has only one instance with a single access point
- ▶ Factory Types
  - Simple Factory (Not a “true” pattern)
    - Encapsulates Object Creation in one place
  - Factory Method
    - Provides an interface to allow subclasses to determine what is instantiated
  - Abstract Factory
    - Provides an interface for creating families of related objects without specifying their concrete class
- ▶ Demo

# Structural

- ▶ **Adapter**
  - Converts the interface of a class into another interface the client expects
- ▶ **Decorator**
  - Attaches additional responsibilities to an object at runtime
- ▶ **Façade**
  - Provides a unified interface to a set of interfaces
- ▶ **Demo**

# Behavioral

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

# Resources

- ▶ “Design Patterns: Elements of Reusable Object Oriented Design”
  - Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides
- ▶ “Head First Design Patterns”
- ▶ [www.dofactory.com](http://www.dofactory.com)

# Contact Me

- ▶ [phil.japikse@pinnsng.com](mailto:phil.japikse@pinnsng.com)
- ▶ [www.pinnsng.com](http://www.pinnsng.com)
- ▶ [www.skimedic.com/blog](http://www.skimedic.com/blog)
- ▶ [www.twitter.com/skimedic](http://www.twitter.com/skimedic)
- ▶ (513) 619-6323

# Questions?

