

Design Patterns

Elements of Reusable
Object-Oriented Software

By

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Presented by:
Steve Brinkley
Joe Holicky

1

Aims of the Book

1. Describe what design patterns are.
2. Explain how they help with design of object-oriented software.

Relates to FP (Family Pattern) design from DeMarco article.

2

WHAT IS A PATTERN?

Christopher Alexander:

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use that solution a million times over, without ever doing it the same way twice.”

3

WHAT IS A PATTERN?

Point of view influences how one defines a pattern.

We aren't concerned with designs that can be encoded and reused as-is.

Neither are we talking about complex, specialized designs.

4

For our purposes, we define pattern as:

A description of communicating objects and classes that is customized to solve a general design problem in a particular context.

5

Essential Pattern Elements

- 1. Pattern Name:** one- or two-word handle we can use to describe a design problem, its solution, and consequences.
- 2. Problem:** advises when to apply the pattern.
- 3. Solution:** details the elements that make up the design, their relationships, responsibilities, and collaborations.
- 4. Consequences:** results and trade-offs of applying the pattern.

6

TYPES OF PATTERNS

A. Purpose - what the pattern does

1. **Creational:** deal with the process of object creation.
2. **Structural:** cover the composition of classes or objects.
3. **Behavioral:** concerned with the ways in which classes or objects interact and distribute responsibility.

B. Scope - what pattern applies to

1. **Class:** class/subclass relationships
2. **Object:** object relationships

7

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

DESIGN PATTERN SPACE

Format of Design Pattern Description

Pattern Name and Classification

Intent

- What does the pattern do?
- Rationale?
- What specific design issue or problem does the pattern address?

Motivation

- Presents scenario that illustrates how the pattern's class and object structures solve the problem.

9

Format of Design Pattern Description

Structure

- Graphical representation of classes in the pattern.
- Interaction diagrams illustrate sequences of requests and collaboration between objects.

Consequences

- How does the pattern support its objectives?
- Results?
- Tradeoffs?
- What aspect of system structure does the pattern let user vary independently?

10

Format of Design Pattern Description

Implementation

- Pitfalls?
- Hints on techniques for using pattern.
- Programming language limitations and issues.

Sample Code

Known Uses in Existing Systems

Related Patterns

11

Pattern Example - Abstract Factory

Pattern Name and Classification

Abstract Factory, Object Creational

Intent

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Definitions (Brown)

Class: Group of objects with similar properties, common behavior, common relationships to other objects, and common meaning.

Abstract Class: Class that has no direct instances.

Concrete Class: Class that does have direct instances.

12

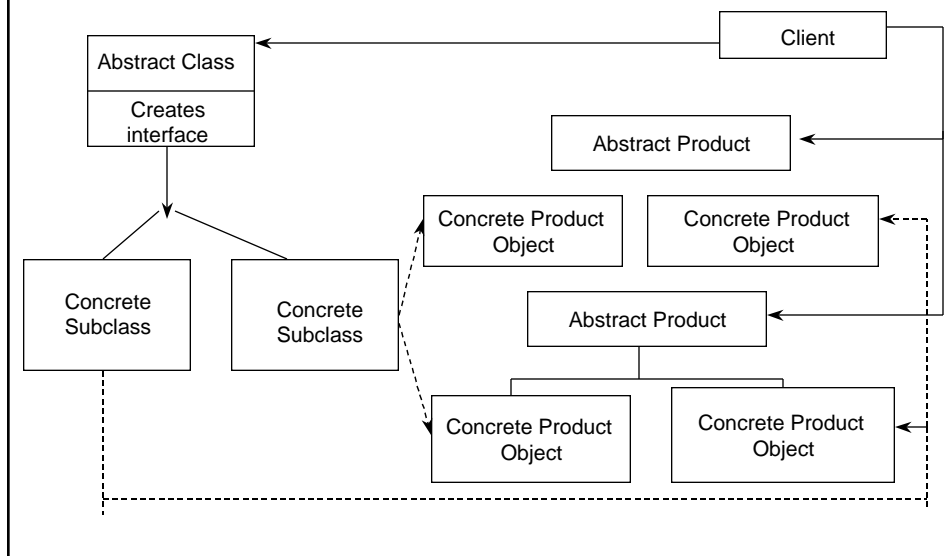
Pattern Example - Abstract Factory

Motivation

Avoid hard-coding classes, so that user standards can easily be changed if necessary.

13

STRUCTURE



Pattern Example - Abstract Factory

Participants

Abstract Class

- declares an interface for operations that create abstract product objects.

Concrete Subclass

- implements the operations to create concrete products objects.

Abstract Product

- declares a interface for a new type of product object.

Concrete Product

- defines a product object to be created by the corresponding concrete factory.

- implements the Abstract Product interface.

Client

- uses only interfaces declared by Abstract Class and Abstract Product.

15

Pattern Example - Abstract Factory

Consequences

1. Isolates concrete classes. Isolates clients from implementation classes (Encapsulation) Clients manipulate instances through their abstract interfaces.
2. Makes exchanging product families easy. Simply change the concrete class.
3. Promotes consistency among products.
4. Supporting new kinds of products is difficult. Requires changing the abstract class and all its subclasses.

16

Pattern Example - Abstract Factory

Implementation

1. Abstract class only declares an interface for creating products. Concrete subclasses must actually create them. Factory Method is the most common pattern for creating the products.
2. Adding a parameter to the abstract class that specifies the kind of product (object) to be created can be a workaround solution to the problem of creating new kinds of products. Tradeoffs are involved, however. Main tradeoff is that the client will not be able to differentiate the class of a product.

17

Pattern Example - Abstract Factory

Known Uses

- Interviews
- ET++

Related Patterns

- Factory Method
- Prototype
- Singleton

18

Pattern Example - Adapter

Pattern Name and Classification

Adapter, Object Structural

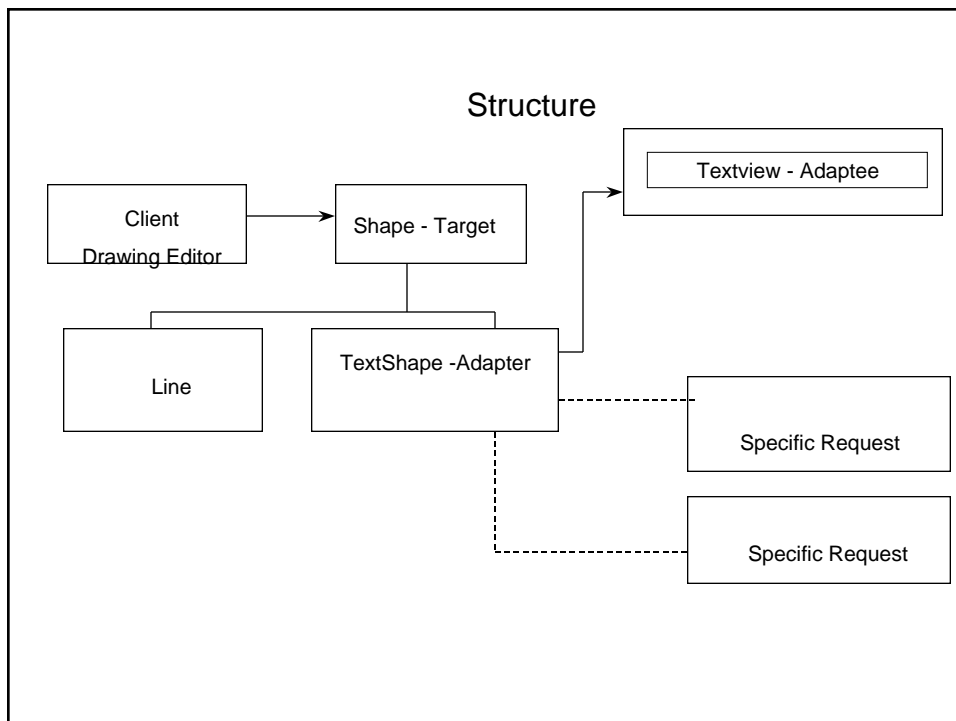
Intent

Convert the interface of a class into another interface clients expect. Allow classes to work together that couldn't otherwise because of incompatible interfaces.

Motivation

Drawing editor example - TextShape adapter adapts TextView interface to Shape's

19



Pattern Example - Adapter

Participants

Target (Shape)

- defines the domain-specific interface that Client uses.

Client (DrawingEditor)

- collaborates with objects performing the Target interface.

Adaptee (TextView)

- defines an existing interface that needs adapting.

Adapter (TextShape)

- adapts the interface of Adaptee to the Target interface.

21

Pattern Example - Adapter

Consequences

A class adapter

- adapts Adaptee to Target by committing to a concrete Adaptee class.
- lets Adapter override some of Adaptee's behavior
- introduces only one object.

An object adapter

- lets a single Adapter work with Adaptees.
- makes it harder to override Adaptee behavior.

22

Pattern Example - Adapter

Implementation

- C++ Issues
- Use of pluggable adapters

Known Uses

- ET++Draw
- InterViews 2.6

Related Patterns

- Bridge
- Decorator
- Proxy

23

THE END

24