Design Patterns and AntiPatterns

Object-Oriented Analysis and Design

CSCI 6448 - Fall 1998

Kenneth M. Anderson

Goals of the Lecture

- Present the ideas behind Design Patterns and AntiPatterns
  - Present two examples of each
- Assign Patterns to Volunteers
  - These will be presented next Tuesday

Design Patterns

- Addison-Wesley book published in 1995
  - Erich Gamma
  - Richard Helm
  - Ralph Johnson
  - John Vlissides
- Known as “The Gang of Four”
- Presents 23 Design Patterns
- ISBN 0-201-63361-2

AntiPatterns

- John Wiley & Sons book published in 1998
  - William J. Brown
  - Raphael C. Malveau
  - Hays W. “Skip” McCormick III
  - Thomas J. Mowbray
- Presents a variety of AntiPatterns
  - Development, Architecture, & Management
- ISBN 0-471-19713-0

Pattern Resources

- Pattern Languages of Programming
  - Technical conference on Patterns
- The Portland Pattern Repository
  - http://c2.com/ppr/
- Patterns Homepage
  - http://hillside.net/patterns/patterns.html

What are Patterns?

- Christopher Alexander talking about buildings and towns
  - "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 this solution a million times over, without ever doing it the same way twice"
  - Alexander, et al., A Pattern Language. Oxford University Press, 1977

Patterns, continued

- Patterns can have different levels of abstraction
- In Design Patterns (the book),
  - Patterns are not classes
  - Patterns are not frameworks
  - Instead, Patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context

Patterns, continued

- So, patterns are formalized solutions to design problems
  - They describe techniques for maximizing flexibility, extensibility, abstraction, etc.
- These solutions can typically be translated to code in a straightforward manner

Elements of a Pattern

- Pattern Name
  - More than just a handle for referring to the pattern
  - Each name adds to a designer's vocabulary
    - Enables the discussion of design at a higher abstraction
- The Problem
  - Gives a detailed description of the problem addressed by the pattern
  - Describes when to apply a pattern
    - Often with a list of preconditions

Elements of a Pattern, continued

- The Solution
  - Describes the elements that make up the design, their relationships, responsibilities, and collaborations
  - Does not describe a concrete solution
    - Instead a template to be applied in many situations

Elements of a Pattern, continued

- The consequences
  - Describes the results and tradeoffs of applying the pattern
    - Critical for evaluating design alternatives
  - Typically include
    - Impact on flexibility, extensibility, or portability
    - Space and Time tradeoffs
    - Language and Implementation issues

Design Pattern Template

- Pattern Name and Classification
  - Creational
  - Structural
  - Behavioral
- Intent
- Also Known As
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns

What is an AntiPattern?

- An AntiPattern
  - describes a commonly occurring solution to a problem that generates decidedly negative consequences
  - A design pattern can thus become part of an AntiPattern, especially when the design problem is used in the wrong way
    - Popular patterns of today can become the AntiPatterns of tomorrow

Parts of an AntiPattern

- An AntiPattern consists of two solutions
  - The AntiPattern Solution
  - The Refactored Solution
- The pattern describes a mechanism of moving from the former to the latter
- In addition, the pattern identifies the context of the former and the benefits of the latter

AntiPatterns Research

- The study of AntiPatterns is an important research activity. The presence of 'good' patterns in a successful system is not enough; you also must show that those patterns are absent in unsuccessful systems. Likewise, it is useful to show the presence of certain patterns (AntiPatterns) in unsuccessful systems, and their absence in successful systems.
  - Attributed to Jim Coplien
- Further motivation
  - A third of all software projects are cancelled, five out of six software projects are unsuccessful [Johnson, 1995]

AntiPattern Viewpoints

- Software Developer
  - technical problems and solutions
- Software Architect
  - problems with structuring systems
- Software Manager
  - problems with software processes and development organizations

Root Causes of AntiPatterns

- Haste
  - Projects are subject to severe schedule-related stress
    - Testing is often the victim
- Apathy
  - Refers to a developer or organization being unwilling to solve known problems
    - maintain the status quo, don't rock the boat!

Root Causes, continued

- Narrow-Mindedness
  - Refusal to practice solutions otherwise known to be effective
- Sloth
  - "Easy technology" promotes poor decisions and poor discipline
    - Lack of configuration control
    - Unstable interfaces

Root Causes, continued

- Avarice
  - Too many details specified for a system
    - Can lead to unnecessary or excessive complexity
- Ignorance
  - Failure to seek understanding of problems and/or solutions
- Pride - "Not Invented Here" Syndrome

Examples

- Design Patterns
  - Factory Method
    - Creational
- AntiPatterns
  - The Blob
    - Development

Factory Method

- Intent
  - Define an interface for creating an object, but let subclasses decide which class to instantiate
- Also Known As
  - Virtual Constructor
- Motivation
  - Frameworks define abstract classes, but any particular domain needs to use specific subclasses; how can the framework create these subclasses?

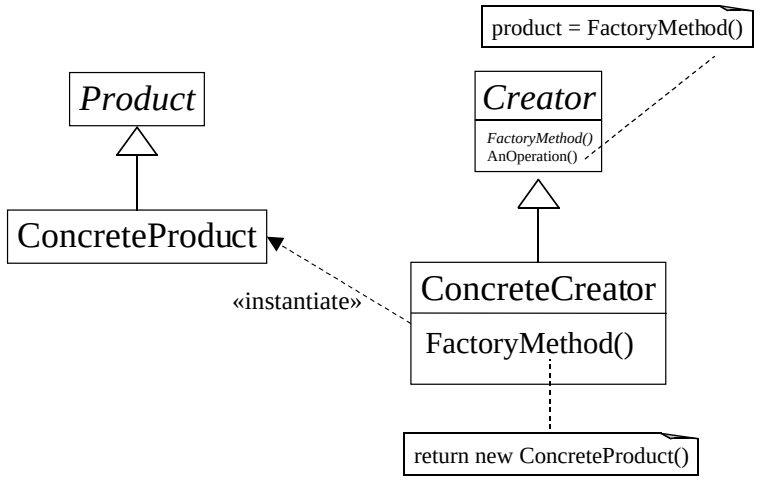
Factory Method, continued

- Applicability
  - Use the Factory Method pattern when
    - a class can't anticipate the class of objects it must create
    - a class wants its subclasses to specify the objects it creates
    - classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate

Factory Method, continued

- Participants
  - Product
    - Defines the interface of objects the factory method creates
  - Concrete Product
    - Implements the Product Interface
  - Creator
    - declares the Factory method which returns an object of type Product
  - Concrete Creator
    - overrides the factory method to return an instance of a Concrete Product

Factory Method Structure



Factory Method Consequences

- Factory methods eliminate the need to bind application-specific classes into your code
- Potential disadvantage is that clients must use subclassing in order to create a particular ConcreteProduct
  - In single-inherited systems, this constrains your partitioning choices
- Provides hooks for subclasses
- Connects parallel class hierarchies

The Blob

- The Blob is found in designs where one class monopolizes the processing, and other classes primarily encapsulate data
  - Like its movie counterpart, the Blob AntiPattern consumes entire object-oriented designs!
- The key problem is that all the responsibilities of a system are assigned to one class
- Its typically the result of a procedural design (wolf) of an OO system (sheep's clothing)

Symptoms of the Blob

- A class with a large number of attributes, operations, or both
- A large class with low cohesion
- A single controller class with simple, data classes

Consequences of the Blob

- Limits OO advantages
  - Low abstraction
  - No extensibility
  - etc.
- Blob class is too complex for reuse
- Blob class is too expensive in terms of time and space

Causes

- Root Causes
  - Sloth, Haste
- More specifically
  - Lack of OO architecture
  - Lack of (any) architecture
  - Lack of architecture enforcement
  - Unwillingness to refactor existing class hierarchy when requirements change

Refactored Solution

- Refactor the Blob
  - 1. Identify related attributes and operations
  - 2. Move these groups to their own class
  - 3. Reduce "far-coupled" classes
    - Remove indirection when there is no need for it
  - 4. Migrate associations between derived classes to common base class
  - 5. Remove transient behavior and attributes to separate classes

Applicability to other Viewpoints

- Manager should insist on up-front investment on architecture design
  - Includes training people who have limited architecture experience
- Architects must have the power to enforce their designs
  - Within reason, of course, good architects must also be willing to acknowledge flaws and redesign!

Time to Volunteer!

- Photocopy a pattern from one of these books
  - Present the pattern in class next Tuesday
  - Plan on at least a five minute presentation
- We have plenty of patterns to choose from!
  - Need at least 6 volunteers, max 10
    - If we don't have enough time on Tuesday, the rest will be presented next Thursday
  - Walt Manaker has already volunteered to present the Bridge Pattern