Producing Production Quality Software

Lecture 11: Design Patterns
Prof. Arthur P. Goldberg
Fall, 2004
Design Patterns

  – Design Patterns “describe simple and elegant solutions to specific problems in OO design”
  – “Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context”.
Objectives

- Reuse
- Flexibility
- Modularity
- Comprehensibility
OO Design is Difficult

• Systems usually require some redesign
• Reuse helps
• Design Patterns facilitate
An example:
Model/View/Controller (MVC)

• History: Smalltalk-80 interfaces
• MVC
  – Model: the application
  – View: screen presentation
  – Controller: controls the UI
• Figure
Another Example: Composite
The Catalog

• All patterns have been used multiple times
• Part of OO ‘folklore’ or elements of some successful systems
• Incomplete: there are hundreds of patterns
A Pattern Summary:
A Catalog Entry

• Pattern name
  – Problem the pattern solves

• An abstract description
  – The solution the pattern offers

• The pattern's general arrangement of elements (classes and objects)
  – The consequences of using the pattern

• Impact on reuse, portability and flexibility
A Pattern’s Details:
A Full Catalog Entry

- Pattern name and classification
- Intent
- AKA
- Motivation
- Applicability
- Structure
- Participants
- Collaborations

- Consequences
- Implementation
- Sample code
- Known uses
- Related patterns
- An example pattern description
# Design Pattern Space

(GoF Table 1.1, simplest and most common patterns in *italics*)

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Creational</th>
<th>Structural</th>
<th>Behavioral</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope</strong></td>
<td><strong>Class</strong></td>
<td><em>Factory Method</em></td>
<td>Adapter (class)</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td><em>Abstract Factory</em></td>
<td>Adapter (object)</td>
<td>Chain of Responsibility</td>
</tr>
<tr>
<td></td>
<td>Builder</td>
<td>Bridge</td>
<td>Command</td>
</tr>
<tr>
<td></td>
<td>Prototype</td>
<td>Composite</td>
<td>Iterator</td>
</tr>
<tr>
<td></td>
<td>Singleton</td>
<td>Decorator</td>
<td>Mediator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Façade</td>
<td>Memento</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flyweight</td>
<td><em>Observer</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proxy</td>
<td><em>State</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>Strategy</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Visitor</td>
</tr>
</tbody>
</table>
Pattern Purposes

• Creational: create objects
• Structural: compose classes or objects into larger structures
• Behavioral: help define the communication between objects in the system and how the flow is controlled in a complex program
References

– Books


– Articles

• Gamma, E., *Applying Design Patterns in Java*, in Java Gems, SIGS Reference Library, 1997

– Lists of patterns

• http://hillside.net/patterns/onlinepatterncatalog.htm

• http://patterndigest.com/
Pattern Intent (from Metsker)

Interfaces
  Adapter, façade, composite, bridge

Responsibility
  Singleton, Observer, Mediator, Proxy, Chain of Responsibility, Flyweight

Construction
  Builder, Factory Method, Abstract Factory, Prototype, Memento

Operations
  Template Method, State, Strategy, Command, Interpreter

Extensions
  Decorator, Iterator, Visitor
Advice

Program to an interface and not to an implementation.

Favor object composition over inheritance.