Introduction To Design Patterns
What Is A Pattern?

- Current use comes from the work of the architect Christopher Alexander
- Alexander studied ways to improve the process of designing buildings and urban areas
- “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution.”
- Hence, the common definition of a pattern: “A solution to a problem in a context.”
- Patterns can be applied to many different areas of human endeavor, including software development
Why Patterns?

- "Designing object-oriented software is hard and designing reusable object-oriented software is even harder." - Erich Gamma
- Experienced designers reuse solutions that have worked in the past
- Well-structured object-oriented systems have recurring patterns of classes and objects
- Knowledge of the patterns that have worked in the past allows a designer to be more productive and the resulting designs to be more flexible and reusable
Software Patterns History

- 1987 - Cunningham and Beck used Alexander’s ideas to develop a small pattern language for Smalltalk
- 1990 - The Gang of Four (Gamma, Helm, Johnson and Vlissides) begin work compiling a catalog of design patterns
- 1991 - Bruce Anderson gives first Patterns Workshop at OOPSLA
- 1993 - Kent Beck and Grady Booch sponsor the first meeting of what is now known as the Hillside Group
- 1994 - First Pattern Languages of Programs (PLoP) conference
- 1995 - The Gang of Four (GoF) publish the Design Patterns book
Types Of Software Patterns

- Analysis
- Design
- Organizational
- Process
- Project Planning
- Configuration Management
Types Of Software Patterns

- Riehle and Zullighoven in “Understanding and Using Patterns in Software Development” mention three types of software patterns

  - Conceptual Pattern
    - Pattern whose form is described by means of terms and concepts from the application domain

  - Design Pattern
    - Pattern whose form is described by means of software design constructs, such as objects, classes, inheritance and aggregation

  - Programming Pattern (Programming Idiom)
    - Pattern whose form is described by means of programming language constructs
Design Pattern Levels Of Abstraction

- Complex design for an entire application or subsystem

- Solution to a general design problem in a particular context

- Simple reusable design class such as a linked list, hash table, etc.
GoF Design Patterns

- The GoF design patterns are in the middle of these levels of abstraction
- “A design pattern names, abstracts, and identifies key aspects of a common design structure that makes it useful for creating a reusable object-oriented design.”
- The GoF design patterns are “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.”
GoF Classification Of Design Patterns

- **Purpose - what a pattern does**
  - Creational Patterns
    - Concern the process of object creation
  - Structural Patterns
    - Deal with the composition of classes and objects
  - Behavioral Patterns
    - Deal with the interaction of classes and objects

- **Scope - what the pattern applies to**
  - Class Patterns
    - Focus on the relationships between classes and their subclasses
      - Involve inheritance reuse
  - Object Patterns
    - Focus on the relationships between objects
      - Involve composition reuse
GoF Essential Elements Of Design Patterns

- Pattern Name
  - Having a concise, meaningful name for a pattern improves communication among developers

- Problem
  - What is the problem and context where we would use this pattern?
  - What are the conditions that must be met before this pattern should be used?

- Solution
  - A description of the elements that make up the design pattern
  - Emphasizes their relationships, responsibilities and collaborations
  - Not a concrete design or implementation; rather an abstract description

- Consequences
  - The pros and cons of using the pattern
  - Includes impacts on reusability, portability, extensibility
GoF Pattern Template

- Pattern Name and Classification
  - A good, concise name for the pattern and the pattern's type

- Intent
  - Short statement about what the pattern does

- Also Known As
  - Other names for the pattern

- Motivation
  - A scenario that illustrates where the pattern would be useful

- Applicability
  - Situations where the pattern can be used
GoF Pattern Template (Continued)

- **Structure**
  - A graphical representation of the pattern

- **Participants**
  - The classes and objects participating in the pattern

- **Collaborations**
  - How to do the participants interact to carry out their responsibilities?

- **Consequences**
  - What are the pros and cons of using the pattern?

- **Implementation**
  - Hints and techniques for implementing the pattern
GoF Pattern Template (Continued)

- Sample Code
  - Code fragments for a sample implementation

- Known Uses
  - Examples of the pattern in real systems

- Related Patterns
  - Other patterns that are closely related to the pattern
GoF Notation

- The GoF book uses the Object Modeling Technique (OMT) notation for class and object diagrams:

- **(a) Abstract and concrete classes**

- **(b) Participant Client class (left) and implicit Client class (right)**

- **(c) Class relationships**

- **(d) Pseudocode annotation**
• We will also use the Unified Modeling Language (UML)
### UML Notation (Continued)

#### Association Symbols

<table>
<thead>
<tr>
<th>Association Type</th>
<th>Class</th>
<th>Association Symbol</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance/Generalisation</td>
<td>Subclass</td>
<td></td>
<td>Superclass</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Whole</td>
<td></td>
<td>Part</td>
</tr>
<tr>
<td>Composition</td>
<td>Whole</td>
<td>←</td>
<td>Part</td>
</tr>
<tr>
<td>Uni-Directional Association</td>
<td>Client</td>
<td>←</td>
<td>Supplier</td>
</tr>
<tr>
<td>Bi-Directional Association</td>
<td>A</td>
<td>←</td>
<td>B</td>
</tr>
<tr>
<td>Dependency</td>
<td>Client</td>
<td>←</td>
<td>Supplier</td>
</tr>
<tr>
<td>Template Instantiation</td>
<td>Template&lt;float&gt;</td>
<td>←</td>
<td>Template</td>
</tr>
</tbody>
</table>
Benefits Of Design Patterns

- Capture expertise and make it accessible to non-experts in a standard form
- Facilitate communication among developers by providing a common language
- Make it easier to reuse successful designs and avoid alternatives that diminish reusability
- Facilitate design modifications
- Improve design documentation
- Improve design understandability
Design Patterns Books

- *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson and Vlissides, Addison-Wesley, 1995
- *Design Patterns for Object-Oriented Software Development*, Wolfgang Pree, Addison-Wesley/ACM Press, 1995
- *Patterns of Software: Tales From The Software Community*, Richard P. Gabriel, Oxford University Press, 1996
- *Pattern Oriented Software Architecture: A System of Patterns*, Frank Buschmann (Editor), Wiley, 1996
- *Analysis Patterns: Reusable Object Models*, Martin Fowler, Addison-Wesley, 1997
Design Patterns Books

- *Pattern Hatching: Design Patterns Applied*, John Vlissides, Addison-Wesley, 1998
- *Patterns in Java Volume 1*, Mark Grand, Wiley, 2nd Ed., 2002
- *Patterns in Java Volume 2*, Mark Grand, Wiley, 1999
- *Java Design Patterns - A Tutorial*, James W. Cooper, Addison-Wesley, 2000
Design Patterns Books

- Design Patterns Explained, Alan Shalloway and James R. Trott, Addison-Wesley, 2001
- Core J2EE Patterns: Best Practices and Design Strategies, Alur, Crupi and Malks, 2001
- Design Patterns Java Workbook, Steven John Metsker, Addison-Wesley, 2002
- Applied Java Patterns, Stephen Stelting and Olav Maassen, Prentice Hall, 2002
- EJB Design Patterns: Advanced Patterns, Processes, and Idioms, Floyd Marinescu, Wiley, 2002
- Patterns Of Enterprise Application Architecture, Martin Fowler, Addison-Wesley, 2002
Design Patterns Books

- *C# Design Patterns - A Tutorial*, James W. Cooper, Addison-Wesley, 2002
- *Design Patterns In C#*, Steven John Metsker, Addison-Wesley, 2004
- *Head First Design Patterns*, Freeman and Freeman, O'Reilly, 2004
- *Refactoring To Patterns*, Joshua Kerievsky, Addison-Wesley, 2005