Design Patterns

CSCE 315 – Programming Studio

adpted from John Keyser's 315 slides

Design Patterns in General

- When designing in some field, often the same general type of problems are encountered
- Usually, there are a set of ways that are “good” for handling such design problems
- Rather than *reinventing* these good solutions, it would be helpful to have a way to recognize the design problem, and know what good solutions to it would tend to be (or already exist!).
- In architecture, a 1977 book, *A Pattern Language*, Christopher Alexander et al. introduced the idea of a way of describing design solutions

Pattern Language

- The idea is to describe how good design is achieved for a field
  - Ideas that are “settled” and well understood are good
- Key aspects of a pattern language include:
  - Identifying common, general (somewhat abstract) problems.
  - Finding common “good” ways of addressing these problems
  - Giving names to these solutions (patterns)
    - Identification, understanding, communication
  - Giving description of the patterns:
    - When and how to apply it
    - What the effects of applying it are
    - How it interacts with other patterns

Design Patterns in Computer Science

- The idea developed over time, but became popular with *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, published in 1995.
- Authors often called the “Gang of Four”, and the book sometimes called the GOF book
- Closely tied to Object-Oriented Programming, although the principles are not limited to OOP

Design Pattern Elements

- Pattern name
  - Name to describe it concisely
- Problem
  - When to apply the pattern
- Solution
  - What is involved in the pattern
- Consequences
  - Results and tradeoffs

Design Pattern Descriptions (GOF)

- Intent
- Also Known As
- Motivation (scenario)
- Applicability (when to use)
- Structure (diagram of how it works)
- Participants (other things it uses)
- Collaborations (how it interacts with other stuff)
- Consequences (results and tradeoffs)
- Implementation (Pitfalls/hints/techniques)
- Sample Code
- Known Uses (examples in real systems)
- Related Patterns (closely related design patterns)

Organizing Patterns

- Several Classification Schemes
  - Purpose
  - Scope (objects vs. classes)
  - Relationships
  - Functional (grouping similar ones)
  - etc.

Purposes of Design Patterns

- Creational
  - Deal with object creation
- Structural
  - Deal with how objects/classes are composed
- Behavioral
  - Deal with how classes/objects interact
- Others for specific domains
  - e.g. Concurrency
  - e.g. User interface

Example Creational: Factory Method

- Intent: Define an interface for creating an object, but let subclasses decide which class to instantiate. Defers instantiation to subclasses.
- Allows code to work with an interface, not the underlying concrete product
- Can be abstract (no default), or provide a default that is overridden by subclasses
- Allows subclasses to specialize and replace the default implementation

Example

- Instead of:

```
    Book* Publisher::CreateBook() {
  Book* aBook = new Book();
  Chapter* c1 = new Chapter(1);
  Chapter* c2 = new Chapter(2);

  aBook->addChapter(c1);
  aBook->addChapter(c2);
}
```

Example

- Use:

```
    Book* Publisher::CreateBook() {
  Book* aBook = makeBook();
  Chapter* c1 = makeChapter(1);
  Chapter* c2 = makeChapter(2);

  aBook->addChapter(c1);
  aBook->addChapter(c2);
}
```

Example Structural: Adapter

- aka Wrapper
- Intent: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- Example: Interface to game AI program

Example Structural: Composite

- Intent: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Composite could be a “leaf” (basic object), in which case, it behaves just like that object
- Composite could be a combination of other objects in a hierarchy. Performs some general operations, then usually calls children
- Example: translating an object (or group of objects) in computer graphics

Example Behavioral: Iterator

- Intent: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation
- Note: does not assume that there is a “true” sequential ordering
- Examples: tree traversal in preorder, postorder, inorder; records returned by DB query.

Example Behavioral: Observer

- Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Example: multiple graph view of same data set (as bar chart, pie chart, etc.); graphs update when base data changes
- Usually attach/detach observers from a subject
  - Observers get called whenever subject changes
  - Subject does not have to worry about how the observers work, it just calls a “notify” to each of them.

Patterns (GOF book)

- Creational:
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

Patterns (GOF book)

- Structural:
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

Patterns (GOF book)

- Behavioral:
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

Summary

- Many patterns out there
  - But, a key to usefulness is being commonly recognized
- Takes experience and practice to get used to identifying/using them