Design Patterns

Idioms, Patterns, Frameworks

- **Idiom**: a small language-specific pattern or technique
  - A more primitive building block
- **Design pattern**: a description of a problem that reoccurs and an outline of an approach to solving that problem
  - Generally domain, language independent
  - Also, analysis patterns
- **Framework**:  
  - A partially completed design that can be extended to solve a problem in a domain
  - Horizontal vs. vertical

Examples of C++ Idioms

- Use of an Init() function in constructors
  - If there are many constructors, make each one call a private function Init()
  - Init() guarantees all possible attributes are initialized
  - Initialization code in one place despite multiple constructors
- Don’t do real work in a constructor
  - Define an Open() member function
  - Open() called immediately after construction
  - Constructors can’t return errors
  - They can throw exceptions

Design Patterns: Essential Elements

- **Pattern name**  
  - A vocabulary of patterns is beneficial
- **Problem**  
  - When to apply the pattern, what context.
  - How to represent, organize components
  - Conditions to be met before using
- **Solution**  
  - Design elements: relationships, responsibilities, collaborations
  - A template for a solution that you implement
- **Consequences**  
  - Results and trade-offs that result from using the pattern
  - Needed to evaluate design alternatives

Patterns Are (and Aren’t)

- **Name and description of a proven solution to a problem**
- **Documentation of a design decision**
- **They’re not:**  
  - Reusable code, class libraries, etc. (At a higher level)
  - Do not require complex implementations
  - Always the best solution to a given situation
  - Simply “a good thing to do”

Readings

- Chapter 1 of GoF book  
  - Especially pp. 1-10, 24-26
  - I’ll get this to you (toolkit, reserve, Web?)
- Eckel’s *Thinking in Patterns*, on Web
  - Chap. 1, “The pattern concept”
  - Chap. 5, “Factories”
- Handouts on various patterns
Describing Design Patterns

- The GoF defined a standard format
  - Generally followed
  - Not just a UML diagram!
- Pattern Format (13 sections):
  - Pattern name and classification
  - Intent: what's it do? Rationale?
  - Also known as
  - Motivation
    - A scenario that illustrates a sample problem and how this pattern helps solve it.
  - Applicability
    - For which situations can this be applied?
  - Structure
    - Graphical representation (e.g. UML)

Example 1: Singleton Pattern

- Context: Only one instance of a class is created. Everything in the system that needs this class interacts with that one object.
- Controlling access: Make this instance accessible to all clients
- Solution:
  - The class has a static variable called `theInstance` (etc)
  - The constructor is made private
    (or protected)
  - Clients call a public operation `getInstance()` that returns
    the one instance
  - This may construct the instance the very first time or
    be given an initializer

Singleton: Java implementation

```java
public class MySingleton {
    private static MySingleton theInstance =
        new MySingleton();
    private MySingleton() { // constructor ...
    }
    public static MySingleton getInstance() {
        return theInstance;
    }
}
```

Static Factory Methods

- Singleton patterns uses a static factory method
  - Factory: something that creates an instance
  - Advantages over a public constructor
    - They have names. Example: `BigInteger(int, int, random)` vs. `BigInteger.probablePrime()`
    - Might need more than one constructor with same/similar signatures
    - Can return objects of a subtype (if needed)
- Wrapper class example:
  - `Double d1 = Double.valueOf("3.14");`;
  - `Double d2 = new Double("3.14");`
- More info: Bloch’s Effective Java

The State Design Pattern

- A connection can be in various states
  - Handles requests differently depending on state
- Connection delegates requests to its state object
  - Which changes dynamically

```
state -> (Open) -> (Open/Closed) -> (Closed/Acknowledged) -> (Closed/Open)...
```