Design Patterns

Based on Head First Design Patterns, available for free from the Purdue Library under Safari Books

Design Patterns

- They are not algorithms
- Collections of patterns exist, but they are not libraries of patterns in the stdlib sense
- Patterns are distillations of OO best practices
- The classic book is the “Gang of Four” book (GOF). There are other books and conferences on patterns, with the Head First book we will use being one
- Patterns can introduce complexity. Complexity makes maintenance debugging and development hard. Use patterns when appropriate.
- Patterns allow code to be modified and functionality to be added with little code outside the enhanced class needing to be changed.

Our OO Design toolbox (1)

- Basics
  - Abstraction – allows us to avoid low-level details
  - Encapsulation – enables change without breaking other code
  - Polymorphism – allows objects of a base (superclass) type to behave like the object they really are (usually a derived object)
- UML to give us visual documentation of our program, and a compact way to represent the relationship between classes

Our OO design toolbox (2)

- Principles
  - Encapsulate what varies
  - Favor composition (HASA) over inheritance (ISA)
  - Program to interfaces, not implementations, i.e., to base (superclasses) not derived or subclasses
- OO patterns
  - We will learn several of these in the rest of the semester

Examples of design patterns

- We will now look at several design patterns, with the first being the Singleton pattern
- We'll pick up some C++ features along the way
- Caveats:
  - Style is a controversial subject
  - Some people like patterns, others don't (too much complexity)
  - The same is true for every style strategy we discuss
  - In programming, we must always use intelligence in applying languages, algorithms and style guidelines – blindly applying anything will lead to bad code in some situation.