Agenda

- Lecture
  - Design Patterns
  - UML

Design Patterns

Design Patterns

- Reusable design component
- First codified by the Gang of Four in 1995
  - Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
- Concept taken from architecture
  - "A Pattern Language" by Christopher Alexander
  - "…a three-part rule, which expresses a relation between a certain context, a problem, and a solution."
- Original Gang of Four book described 23 patterns
  - More have been added
  - Other authors have written books

Design Patterns Template

- Context
  - General situation in which the pattern applies
- Problem
  - The main difficulty being tackled
- Forces
  - Issues or concerns that need to be considered. Includes criteria for evaluating a good solution.
- Solution
  - Recommended way to solve the problem in the context. The solution "balances the forces"
- The following are optional
- Antipatterns
  - Common mistakes to avoid
- Related Patterns
  - Similar patterns; could be alternated solutions or work with the pattern
- References
  - Source of pattern
  - Who developed or inspired the pattern

Gang of Four Design Patterns

- Creational Patterns
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Behavioral Patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

Patterns in Java

- Chain of Responsibility
  - Exception handling
  - Try/catch/throw blocks
- Iterator
  - Container classes
- Observer
  - Listeners in GUIs

Gang of Four Design Patterns

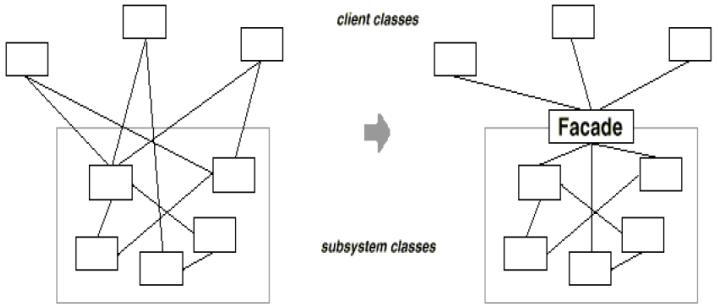
- Creational Patterns
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Behavioral Patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

The Façade Pattern

- Context
  - Often, an application contains several complex packages.
  - A programmer working with such packages has to manipulate many different classes
- Problem
  - How do you simplify the view that programmers have of a complex package?
- Forces
  - It is hard for a programmer to understand and use an entire subsystem
  - If several different application classes call methods of the complex package, then any modifications made to the package will necessitate a complete review of all these classes.

The Façade Pattern

•Solution



The Façade Pattern

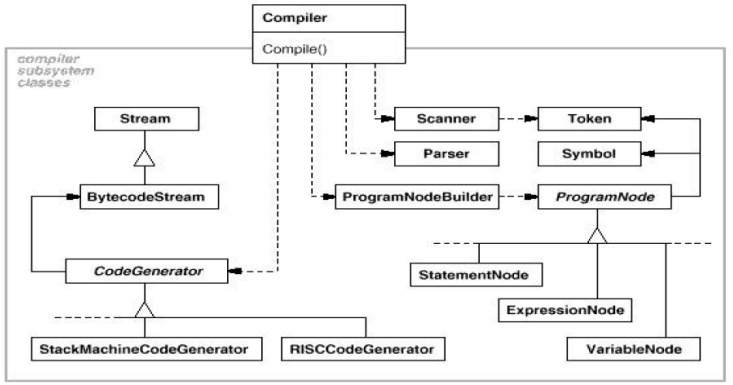
- Solution
  - Provide a simple interface to a complex subsystem.
  - Decouple the classes of the subsystem from its clients and other subsystems, thereby promoting subsystem independence and portability

Using the Façade Pattern

- Hides implementation details
- Promotes weak coupling between the subsystem and its clients.
- Reduces compilation dependencies in large software systems

- Does not add any functionality, it just simplifies interfaces
- Does not prevent clients from accessing the underlying classes.

Façade Example



Package Diagrams

- Package is a grouping construct
  - Most commonly used for class diagrams, but can be used with any UML diagram or elements
  - Used to create a hierarchy or higher level of abstraction
  - Corresponds to package in Java
- Each package represents a namespace
  - Like Java, can have classes with same name in different packages

Representing Packages