Agenda

- Lecture
  - Design Patterns
  - UML

Class Diagrams

- Class diagrams define the structure of the classes in a system, the relationship between all classes, and the components of each class.

Class

A class is a general concept (represented as a square box). A class defines the structural attributes and behavioural characteristics of that concept. Shown as a rectangle labeled with the class name.

Association

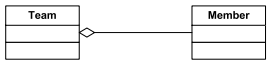
A (semantic) relationship between classes. A line that joins two classes.

Class Diagrams (cont)

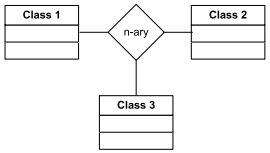
- Types of associations

Binary

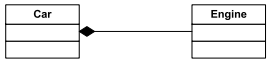
Aggregation (has-a)



n-ary



Composition (is-composed-of)

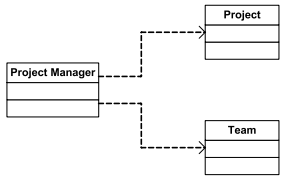


Generalization (is-a-kind-of)

Class Diagrams (cont)

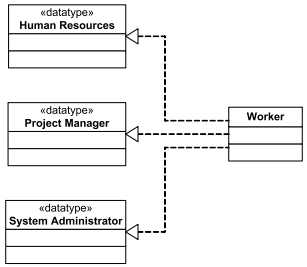
- Types of associations (cont)

Dependency



The source class depends on (uses) the target class

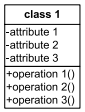
Realization



Class supports all operations of target class but not all attributes or associations.

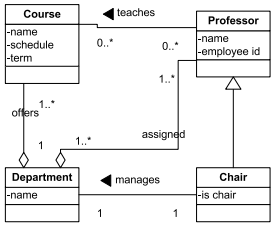
Class Diagrams (cont)

- Attributes and operations



Attributes are what is known about each object of this class type. Operations are what objects of this class type do.

- Multiplicity
  - *n*, where *n* = {0, 1, *x*, *}
  - *m..n*, where *m,n* = {0, 1, *x*, *}



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

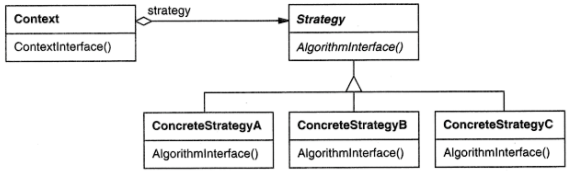
- Creational Patterns
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Behavioral Patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator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

Strategy Design Pattern

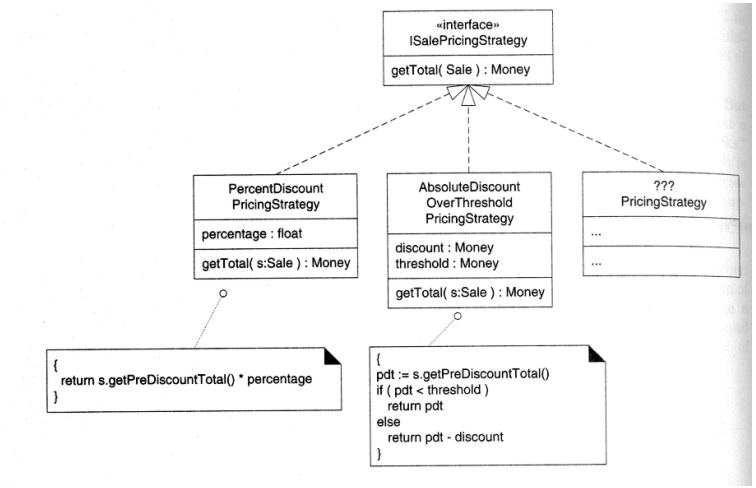
- Context
  - Define a family of algorithms, so they are interchangeable.
- Also Known As
  - Policy
- Problem
  - How to design for varying, but related algorithms or policies? How to design for the ability to change the algorithms or policies?
- Solution
  - Define each algorithm/policy/strategy in a separate class with a common interface

Strategy Design Pattern

- Structure



Example



Strategy Design Pattern

- Participants
  - Strategy interface, concrete Strategy, and Context/client
- Consequences
  - Provides an alternative to subclassing the Context class to get a variety of algorithms or behaviors
  - Eliminates large conditional statements
  - Provides a choice of implementations for the same behavior
  - Increases the number of objects
  - All algorithms must use the same Strategy interface
- Implementation
  - Can use an Abstract Factory to create a Strategy

The Observer Pattern

- Context
  - When an association is created between two classes, the code for the classes becomes inseparable.
  - If you want to reuse one class, then you also have to reuse the other.
- Problem
  - How do you reduce the interconnection between classes, especially between classes that belong to different modules or subsystems?
- Forces
  - You want to maximize the flexibility of the system to the greatest extent possible

The Observer Pattern

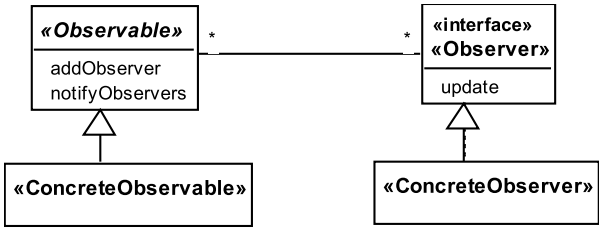
- Solution

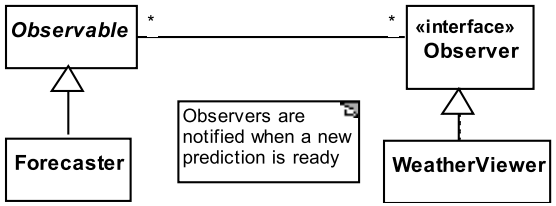
Observer

- Antipatterns (Don't do this)
  - Connect an observer directly to an observable so that they both have references to each other.
  - Make the observers *subclasses* of the observable.
- Reference
  - Gang of Four

The Observer Pattern

- Solution





Observer in Java

- Observer interface and Observable class exist
  - java.util.Observer and java.util.Observable
- But people usually implement their own
  - Usually can't or don't want to sub-class from Observable
  - Can't have your own class hierarchy and multiple inheritance is not available
  - Has been replaced by the Java Delegation Event Model (DEM)
    - Passes event objects instead of update/notify
- Listener is specific to GUI classes