Design Class Diagrams

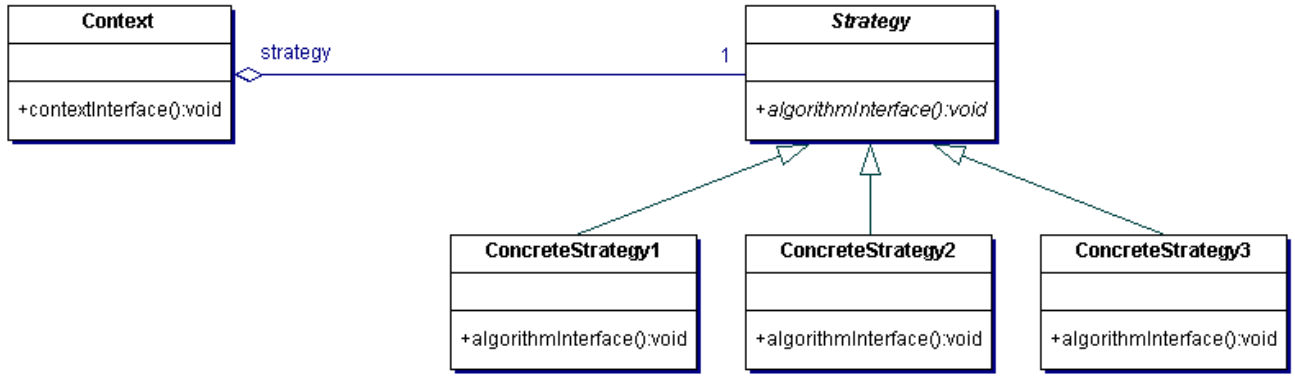
Originally from www.javacoder.net*

Strategy

"Strategy, State, Bridge (and to some degree Adapter) have similar solution structures. They all share elements of the "handle/body" idiom. They differ in intent - that is, they solve different problems." [Coplien, *Advanced C++*, p58]

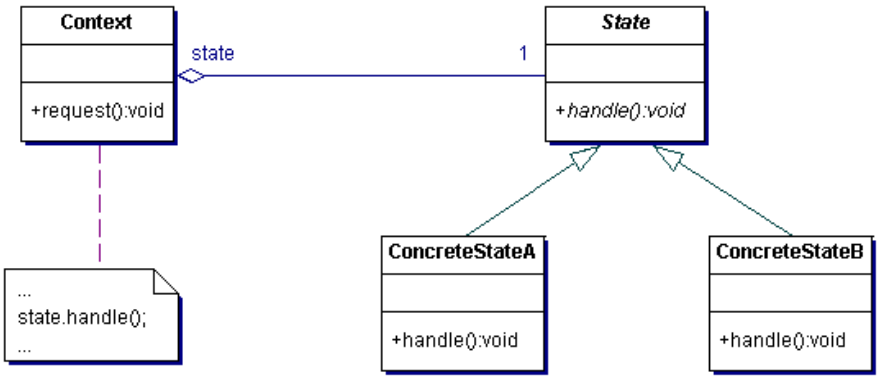
Most of the GoF patterns exercise the two levels of indirection demonstrated here.

1. Promote the "interface" of a method to an abstract base class or interface, and bury the many possible implementation choices in concrete derived classes.
2. Hide the implementation hierarchy behind a "wrapper" class that can perform responsibilities like: choosing the best implementation, caching, state management, remote access.



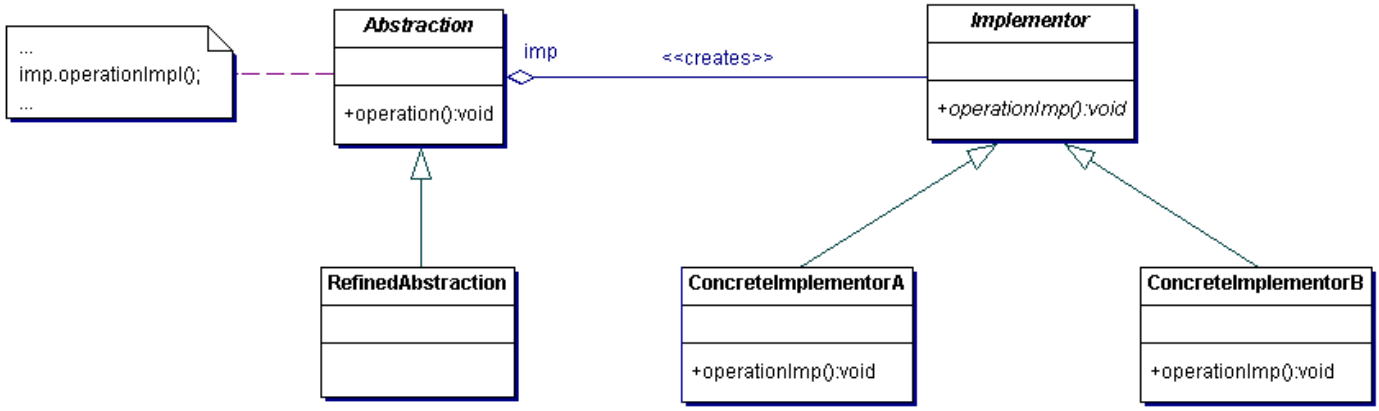
State

"Strategy is a bind-once pattern, whereas State is more dynamic." [Coplien, *C++ Report*, Mar96, p88]



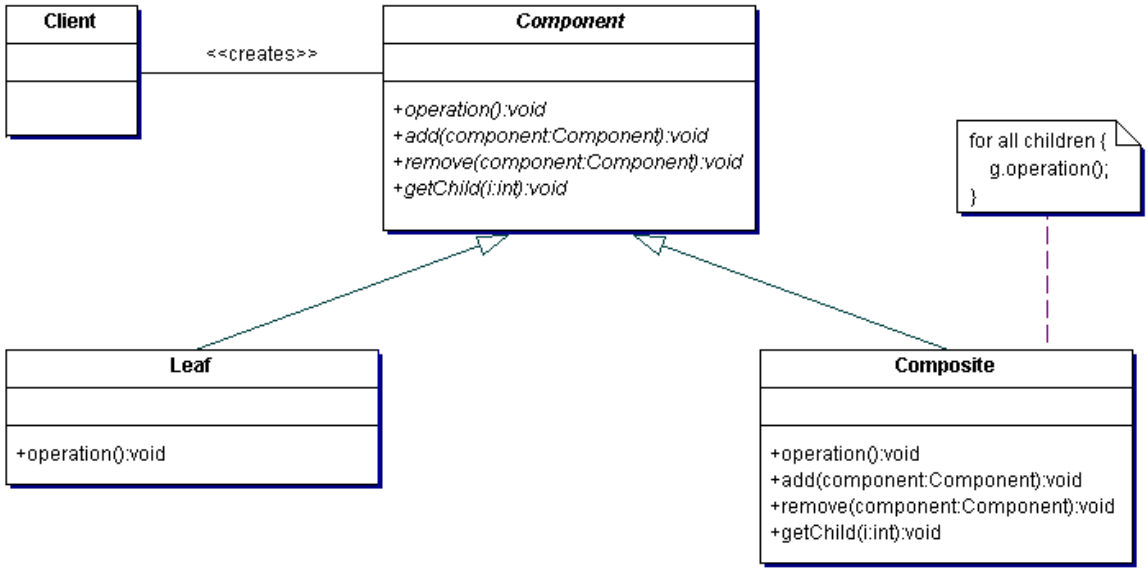
Bridge

The structure of State and Bridge are identical (except that Bridge admits hierarchies of envelope classes, whereas State allows only one). The two patterns use the same structure to solve different problems: State allows an object's behavior to change along with its state, while Bridge's intent is to decouple an abstraction from its implementation so that the two can vary independently. [Coplien, *C++ Report*, May 95, p58]



Composite

Three GoF patterns rely on recursive composition: Composite, Decorator, and Chain of Responsibility.

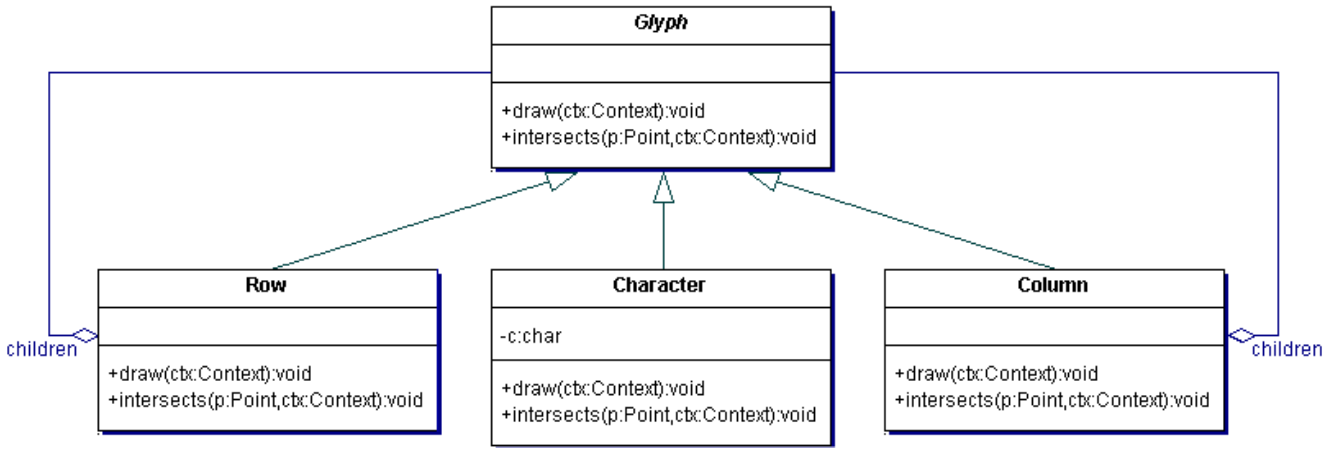


Flyweight

Flyweight is often combined with Composite to implement shared leaf nodes. [GoF, p206]

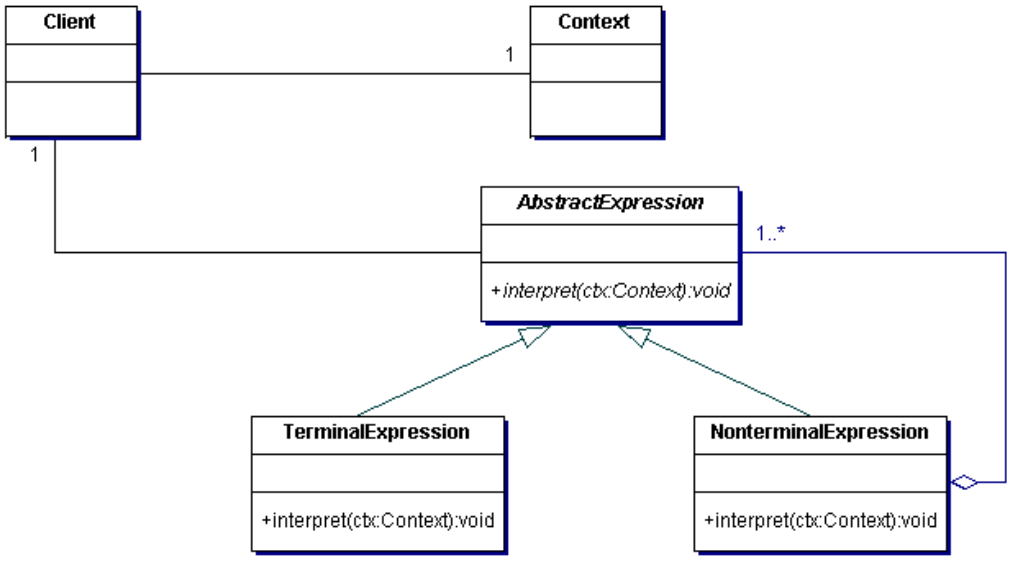
Flyweight shows how to make lots of little objects. Facade shows how to make a single object represent an entire subsystem. [GoF, p138]

This diagram is perhaps a better example of Composite than the Composite diagram.



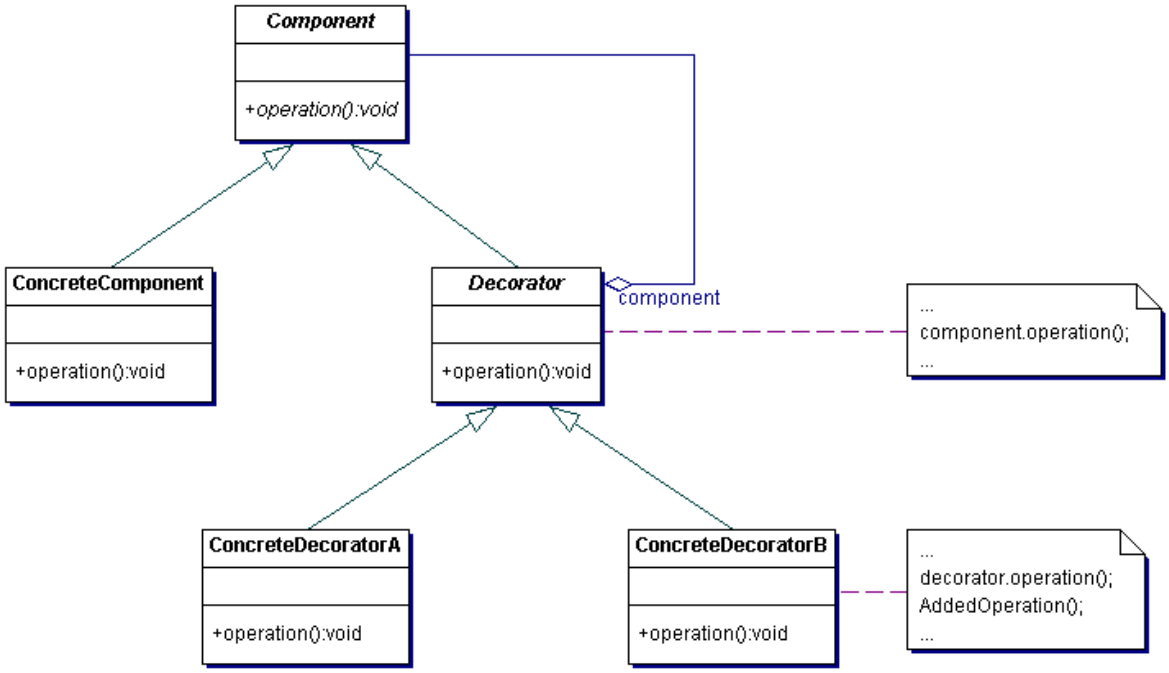
Interpreter

Interpreter is really an application of Composite.



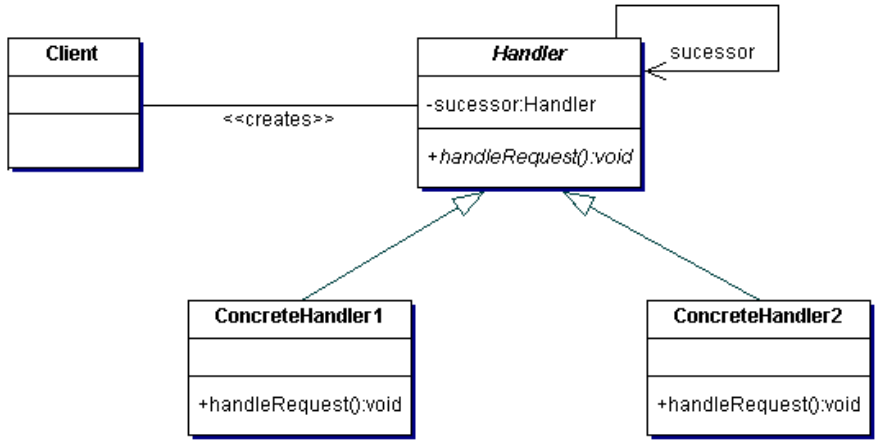
Decorator

Decorator is designed to let you add responsibilities to objects without subclassing. Composite's focus is not on embellishment but on representation. These intents are distinct but complementary. Consequently, Composite and Decorator are often used in concert. [GoF, p220]



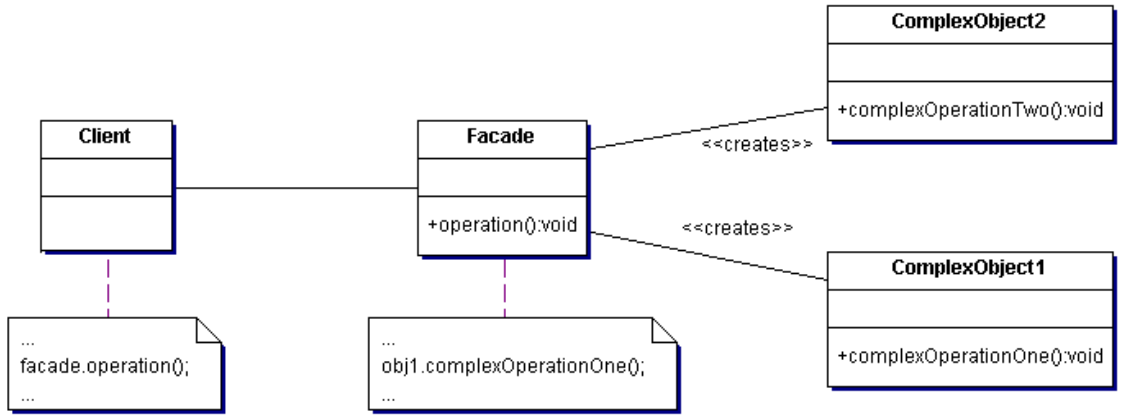
Chain of Responsibility

Chain of Responsibility, Command, Mediator, and Observer, address how you can decouple senders and receivers, but with different trade-offs. Chain of Responsibility passes a sender request along a chain of potential receivers.



Facade

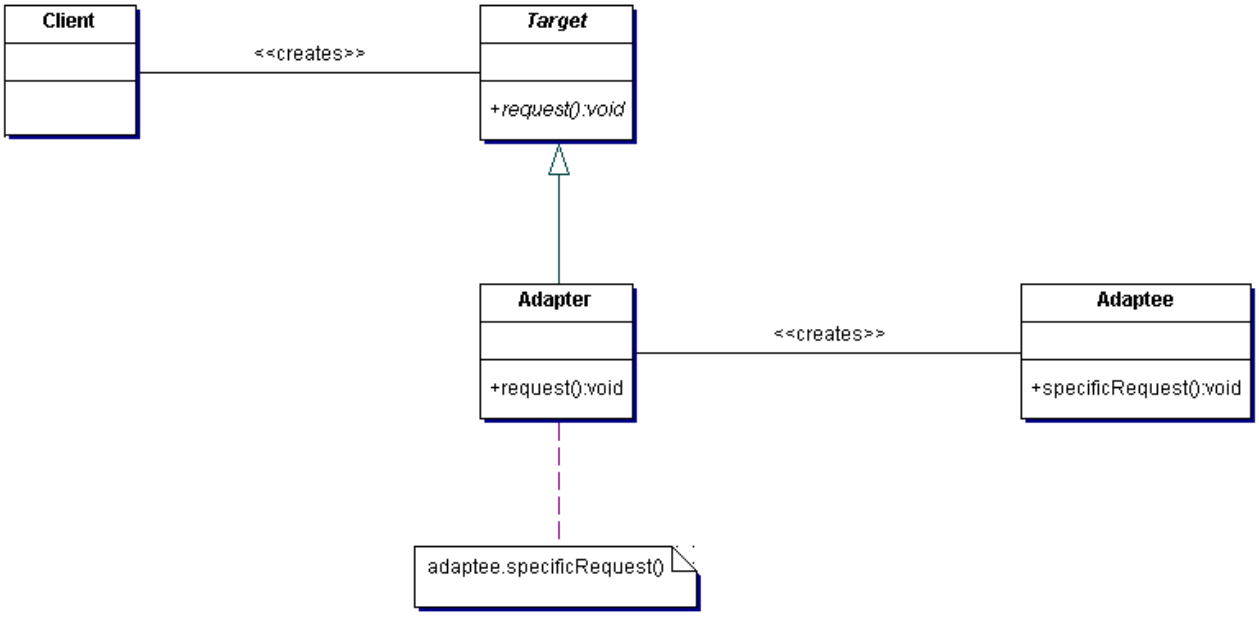
Facade defines a new interface, whereas Adapter uses an old interface. Remember that Adapter makes two existing interfaces work together as opposed to defining an entirely new one. [GoF, p219]



Adapter

Adapter makes things work after they're designed; Bridge makes them work before they are. [GoF, p219]

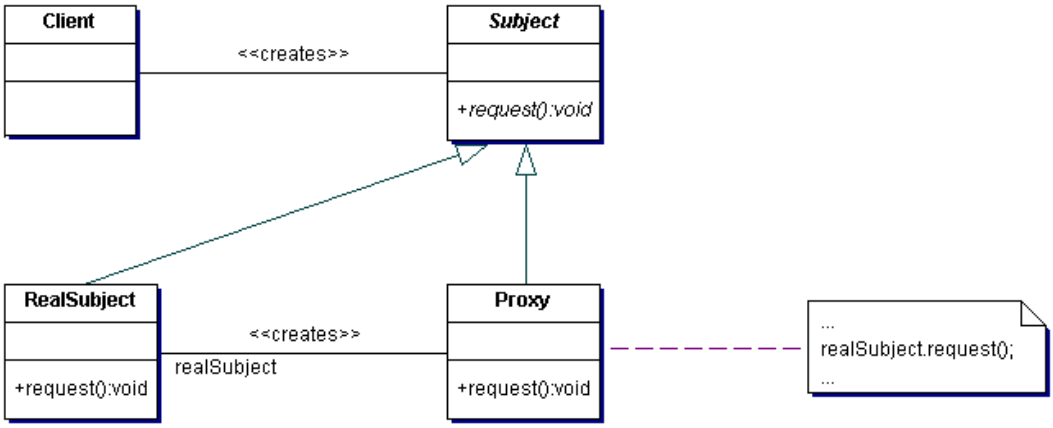
Bridge is designed up-front to let the abstraction and the implementation vary independently. Adapter is retrofitted to make unrelated classes work together. [GoF, 216]



Proxy

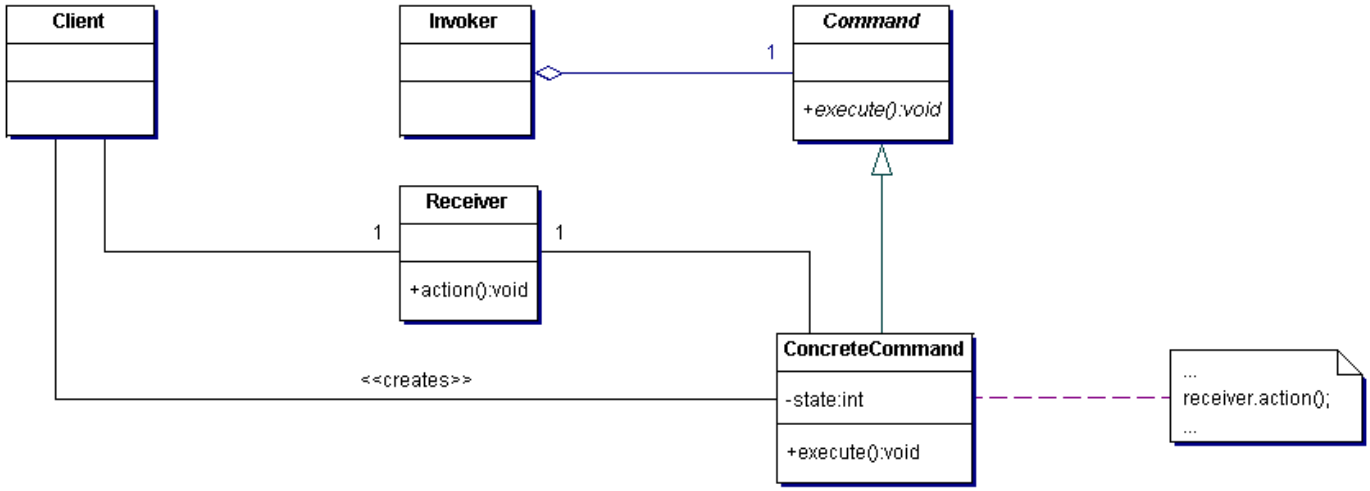
Decorator and Proxy have different purposes but similar structures. Both describe how to provide a level of indirection to another object, and the implementations keep a reference to the object to which they forward requests. [GoF, p220]

Adapter provides a different interface to its subject. Proxy provides the same interface. Decorator provides an enhanced interface. [GoF. p216]



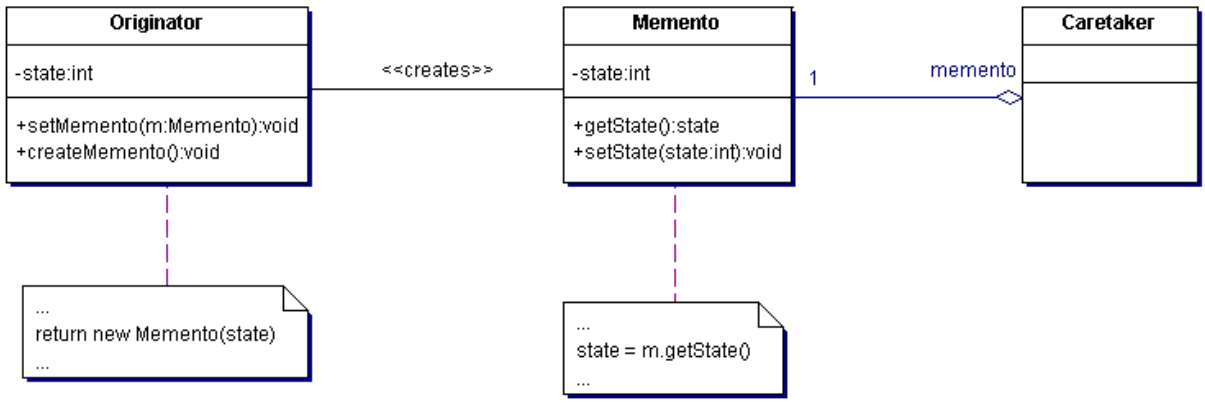
Command

Command and Memento act as magic tokens to be passed around and invoked at a later time. In Command, the token represents a request; in Memento, it represents the internal state of an object at a particular time. Polymorphism is important to Command, but not to Memento because its interface is so narrow that a memento can only be passed as a value. [GoF, p346]



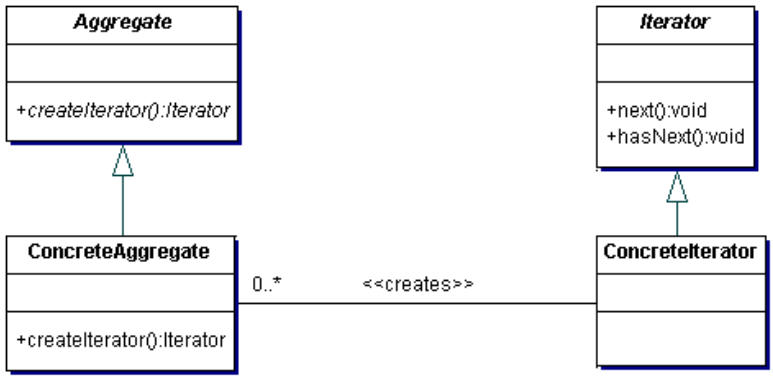
Memento

Command can use Memento to maintain the state required for an undo operation. [GoF, 242]



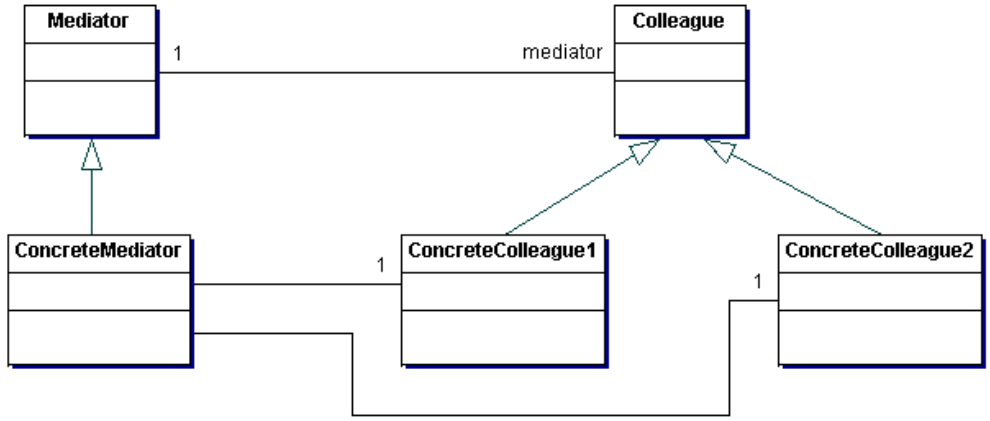
Iterator

Memento is often used in conjunction with Iterator. An Iterator can use a Memento to capture the state of an iteration. The Iterator stores the Memento internally. [GoF, p271]



Mediator

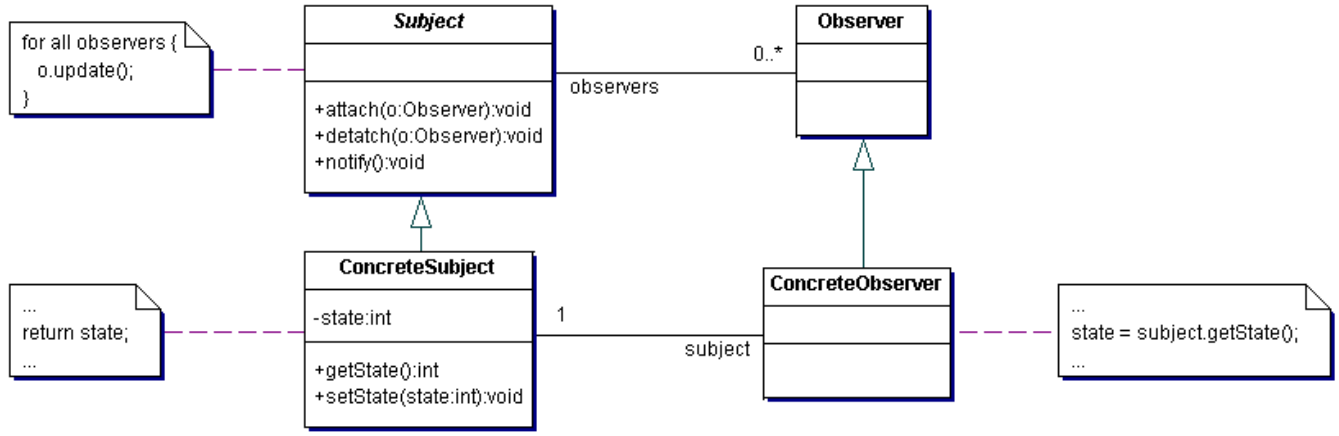
Mediator is similar to Facade in that it abstracts functionality of existing classes. Mediator abstracts/centralizes arbitrary communications between colleague objects. It routinely "adds value", and it is known/referenced by the colleague objects. In contrast, Facade defines a simpler interface to a subsystem, it doesn't add new functionality, and it is not known by the subsystem classes. [GoF, p193]



Observer

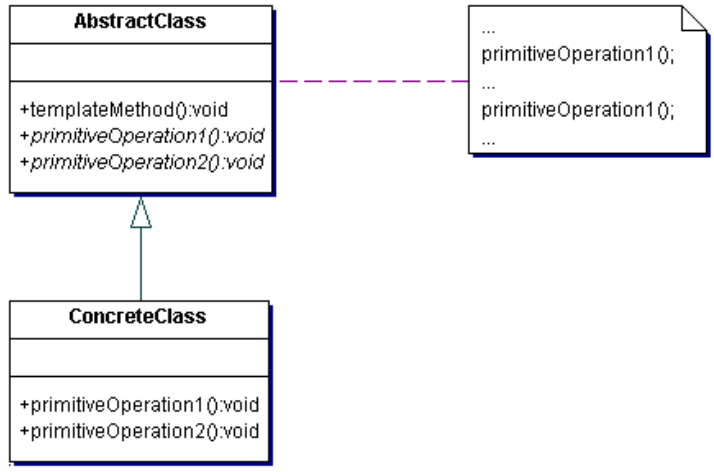
Mediator and Observer are competing patterns. The difference between them is that Observer distributes communication by introducing "observer" and "subject" objects, whereas a Mediator object encapsulates the communication between other objects. We've found it easier to make reusable Observers and Subjects than to make reusable Mediators. [GoF, p346]

On the other hand, Mediator can leverage Observer for dynamically registering colleagues and communicating with them. [GoF, p282]



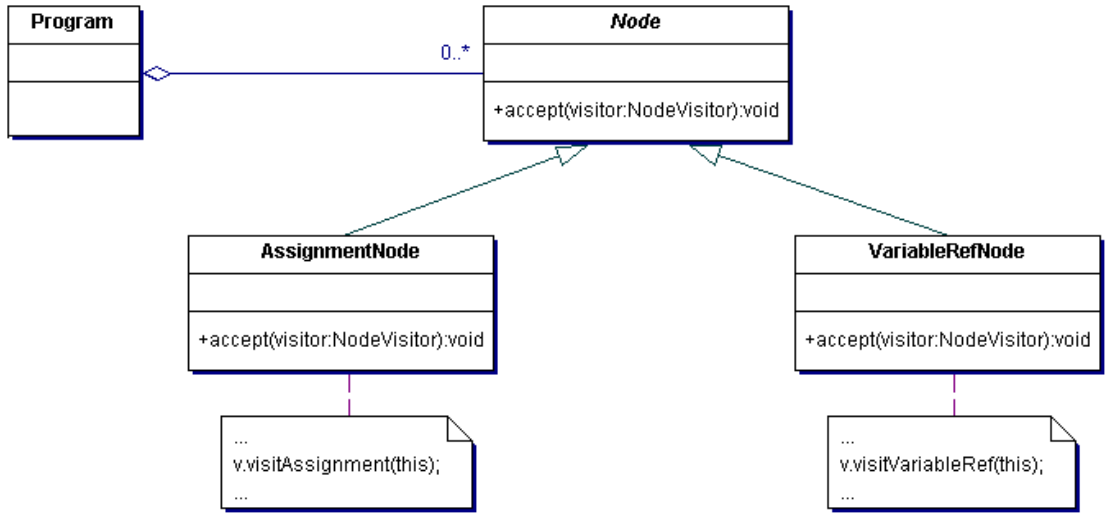
Template Method

Template Method uses inheritance to vary part of an algorithm. Strategy uses delegation to vary the entire algorithm. [GoF, p330]



Visitor

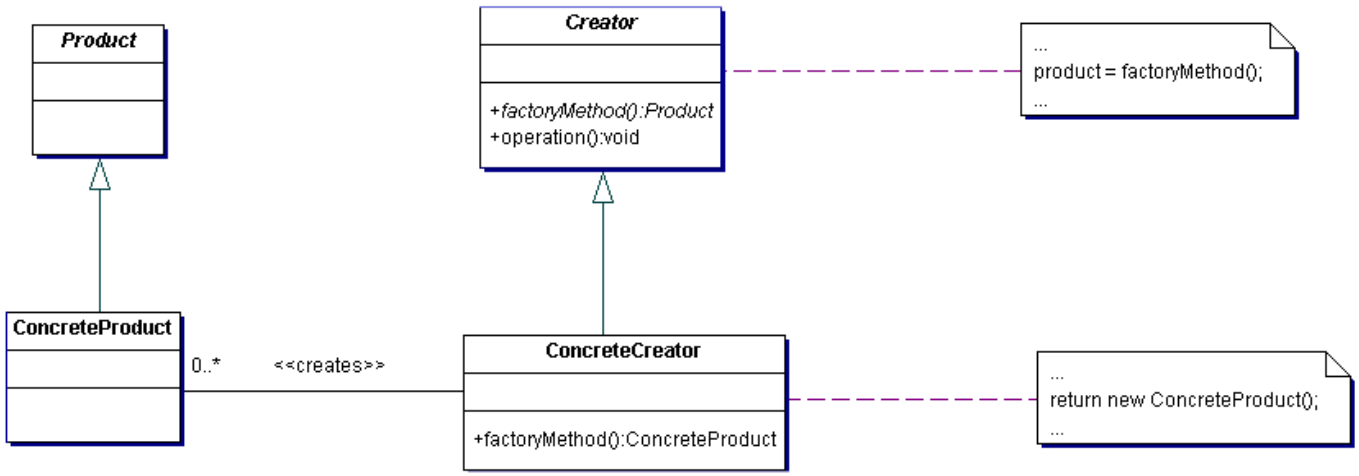
The Visitor pattern is the classic technique for recovering lost type information without resorting to dynamic casts. [Vlissides, "Type Laundering", *C++ Report*, Feb 97, p48]



Factory Method

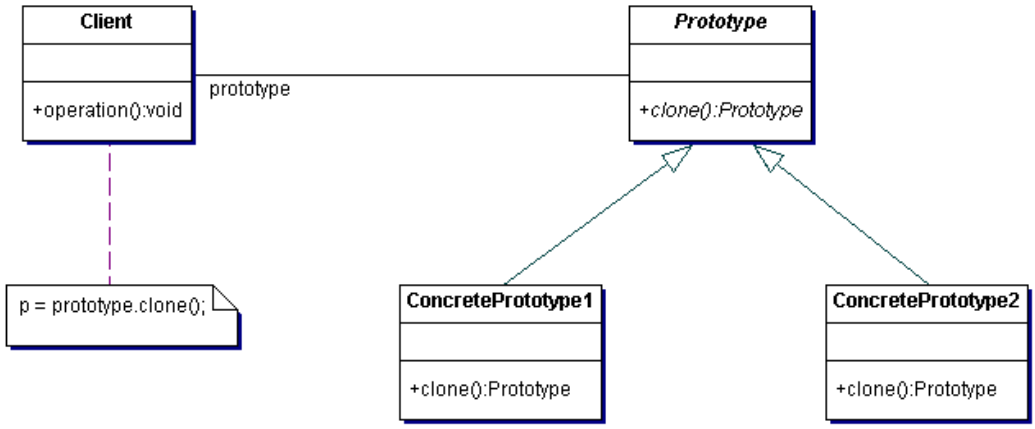
Factory Methods are usually called within Template Methods. [GoF, p116]

Often, designs start out using Factory Method (less complicated, more customizable, subclasses proliferate) and evolve toward Abstract Factory, Prototype, or Builder (more flexible, more complex) as the designer discovers where more flexibility is needed. [GoF, p136]



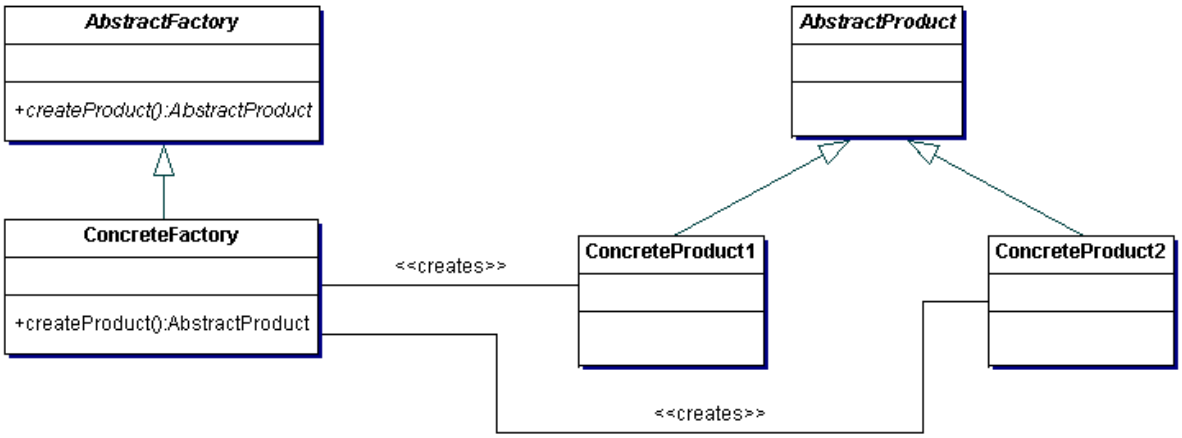
Prototype

Factory Method: creation through inheritance. Prototype: creation through delegation.



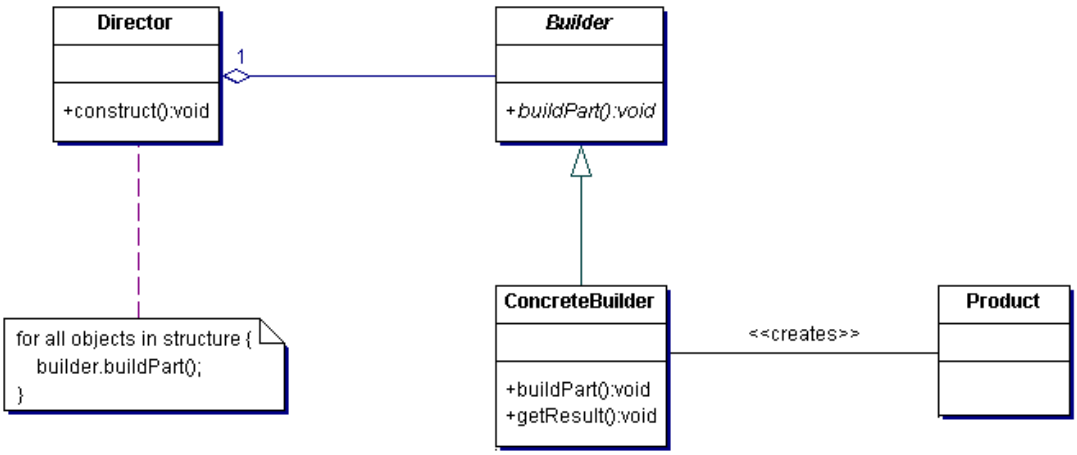
Abstract Factory

Abstract Factory classes are often implemented with Factory Methods, but they can be implemented using Prototype. [GoF, p95]



Builder

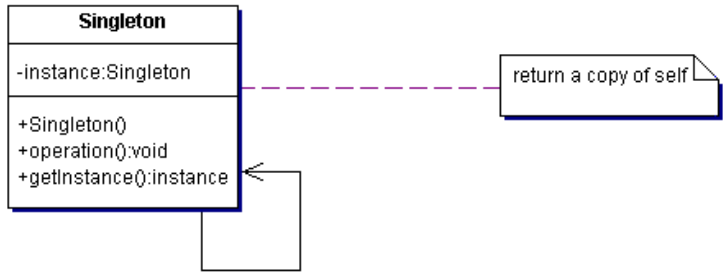
Builder focuses on constructing a complex object step by step. Abstract Factory emphasizes a family of product objects (either simple or complex). Builder returns the product as a final step, but as far as the Abstract Factory is concerned, the product gets returned immediately. [GoF, p105]



Singleton

Singleton should be considered only if all three of the following criteria are satisfied:

- Ownership of the single instance cannot be reasonably assigned
- Lazy initialization is desirable
- Global access is not otherwise provided for



---

* Original source is not an active link anymore. As of 3/30/07, these figures or their successors are posted at http://home.earthlink.net/~huston2/dp/all_uml.html.