Legend

C	Abstract Factory	S	Facade	S	Proxy	
S	Adapter	C	Factory Method	B	Observer	
S	Bridge	S	Flyweight	C	Singleton	
C	Builder	B	Interpreter	B	State	
B	Chain of Responsibility	B	Iterator	B	Strategy	
B	Command	B	Mediator	B	Template Method	
S	Composite	B	Memento	B	Visitor	
S	Decorator	C	Prototype			

Memento

Type: Behavioral

What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

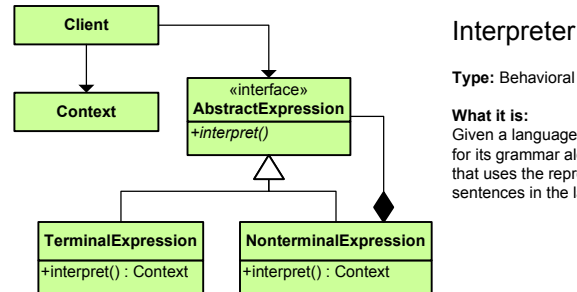## Chain of Responsibility

Type: Behavioral

What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Observer

Type: Behavioral

What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

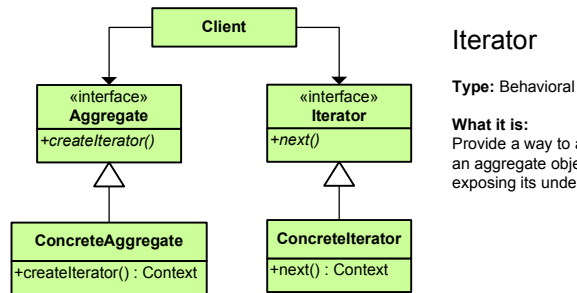## Command

Type: Behavioral

What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

State

Type: Behavioral

What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
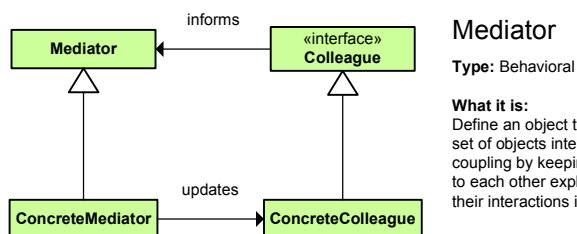

Interpreter

Type: Behavioral

What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Strategy

Type: Behavioral

What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.

Iterator

Type: Behavioral

What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Template Method

Type: Behavioral

What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
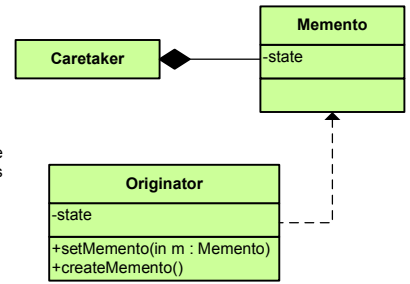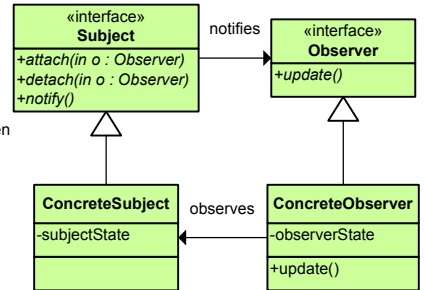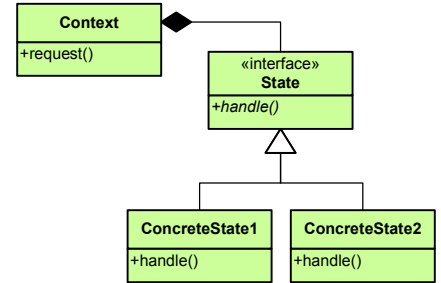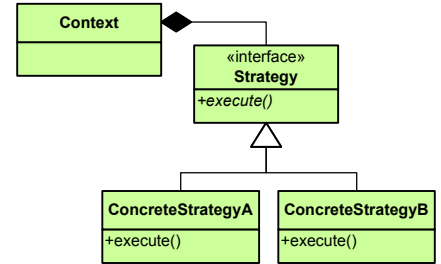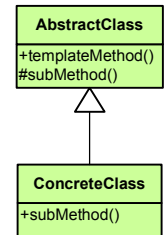

Mediator

Type: Behavioral

What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

Visitor

Type: Behavioral

What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc.
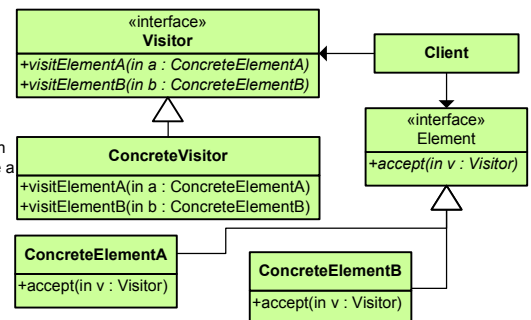
Adapter

Type: Structural

What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

«interface»
Adapter
+*operation()*
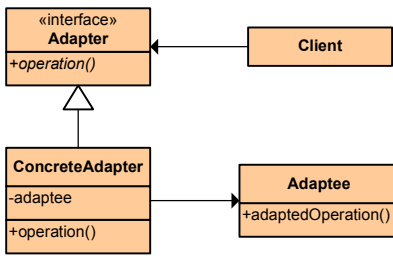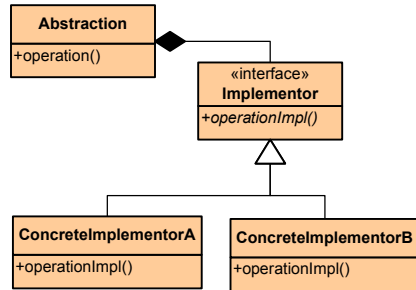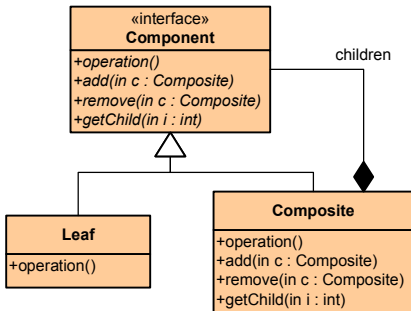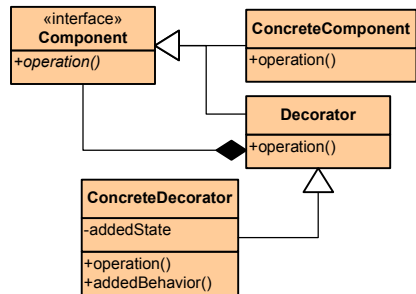
Client

ConcreteAdapter
-adaptee
+operation()

Adaptee
+adaptedOperation()

Bridge

Type: Structural

What it is:
Decouple an abstraction from its implementation so that the two can vary independently.

Abstraction
+operation()

«interface»
Implementor
+*operationImpl()*

ConcreteImplementorA
+operationImpl()

ConcreteImplementorB
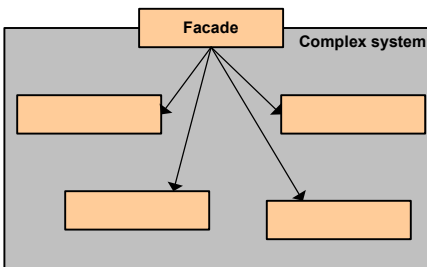+operationImpl()

Composite

Type: Structural

What it is:
Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.

«interface»
Component
+*operation()*
+*add(in c : Composite)*
+*remove(in c : Composite)*
+*getChild(in i : int)*

children

Leaf
+operation()

Composite
+operation()
+add(c : Composite)
+remove(in c : Composite)
+getChild(i : int)

Decorator

Type: Structural

What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.

«interface»
Component
+*operation()*

ConcreteComponent
+operation()

Decorator
+operation()

ConcreteDecorator
-addedState
+operation()
+addedBehavior()

Facade

Type: Structural

What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

Facade

Complex system
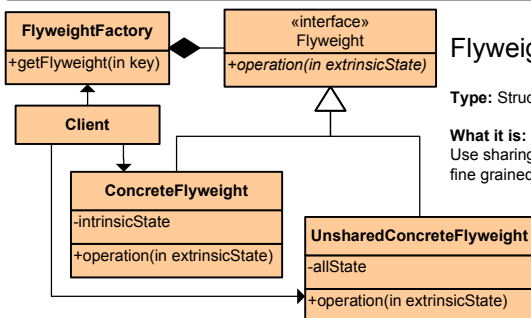
Flyweight

Type: Structural

What it is:
Use sharing to support large numbers of fine grained objects efficiently.

FlyweightFactory
+getFlyweight(in key)

«interface»
Flyweight
+*operation(in extrinsicState)*

Client

ConcreteFlyweight
-intrinsicState
+operation(in extrinsicState)

UnsharedConcreteFlyweight
-allState
+operation(in extrinsicState)

Proxy

Type: Structural

What it is:
Provide a surrogate or placeholder for another object to control access to it.

Client

«interface»
Subject
+*request()*

RealSubject
+request()

represents

Proxy
+request()
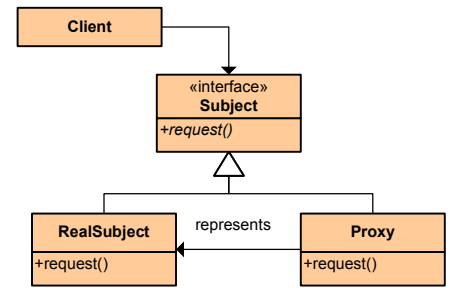
Abstract Factory

Type: Creational

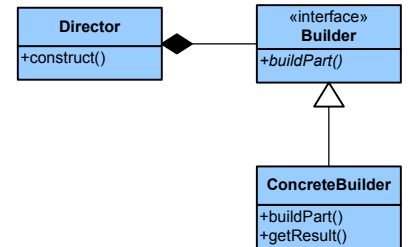**What it is:**
Provides an interface for creating families of related or dependent objects without specifying their concrete class.

Client

«interface»
AbstractFactory
+*createProductA()*
+*createProductB()*

«interface»
AbstractProduct

ConcreteFactory
+createProductA()
+createProductB()

ConcreteProduct
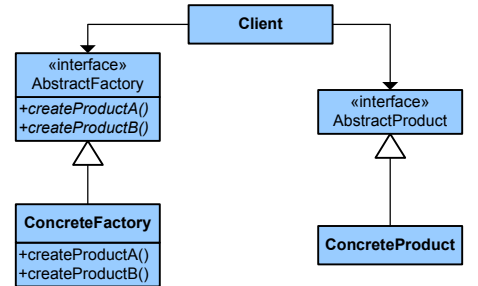
Builder

Type: Creational

What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.

Director
+construct()

«interface»
Builder
+*buildPart()*

ConcreteBuilder
+buildPart()
+getResult()

Factory Method

Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.

«interface»
Product

Creator
+factoryMethod()
+anOperation()

ConcreteProduct

ConcreteCreator
+factoryMethod()
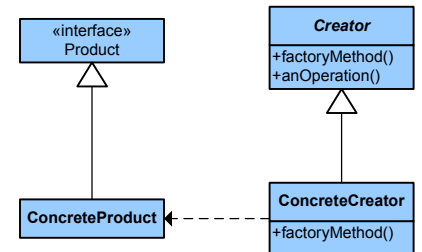
Prototype

Type: Creational

What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Client

«interface»
Prototype
+*clone()*

ConcretePrototype1
+clone()

ConcretePrototype2
+clone()
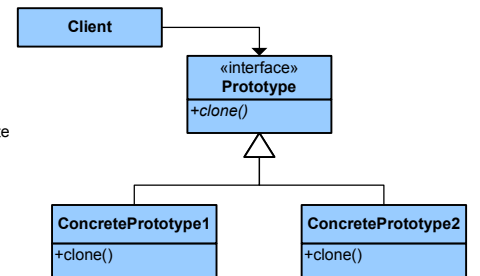
Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.

Singleton
-static uniqueInstance
-singletonData
+static instance()
+SingletonOperation()

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc..