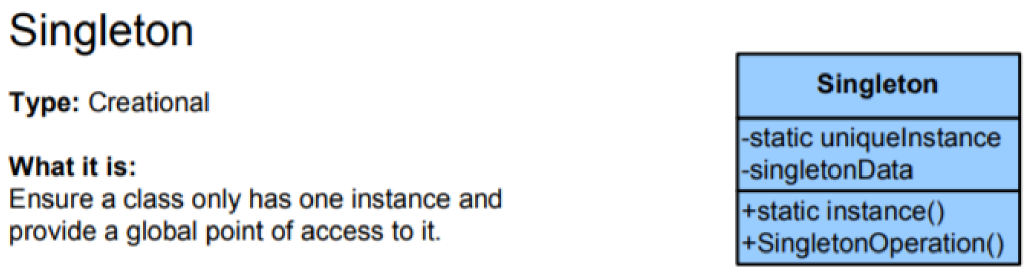
Design Patterns

Note this is not a complete list of the design patterns

Creational

Singleton

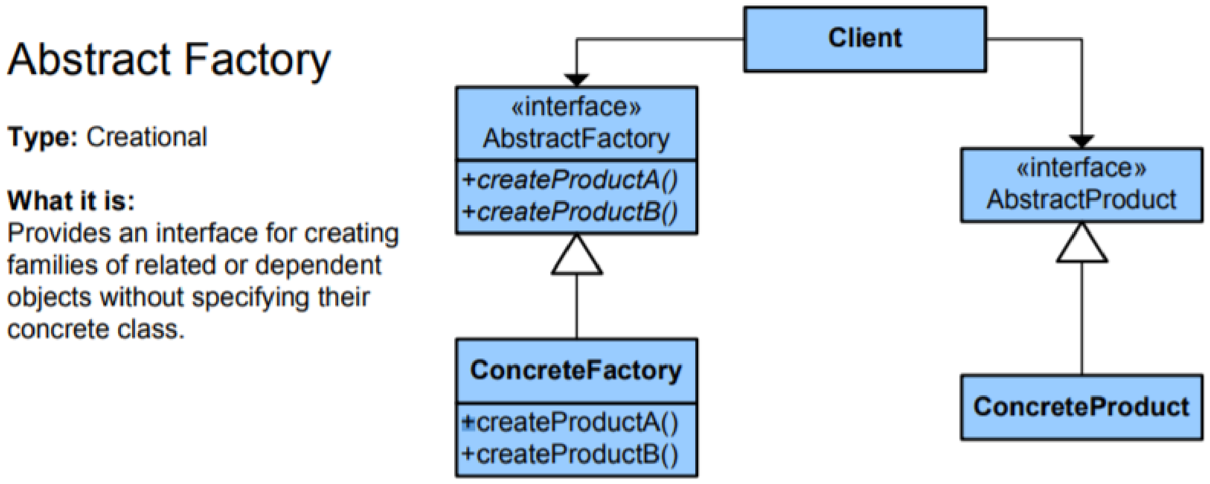


Singleton

Type: Creational

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

Abstract Factory

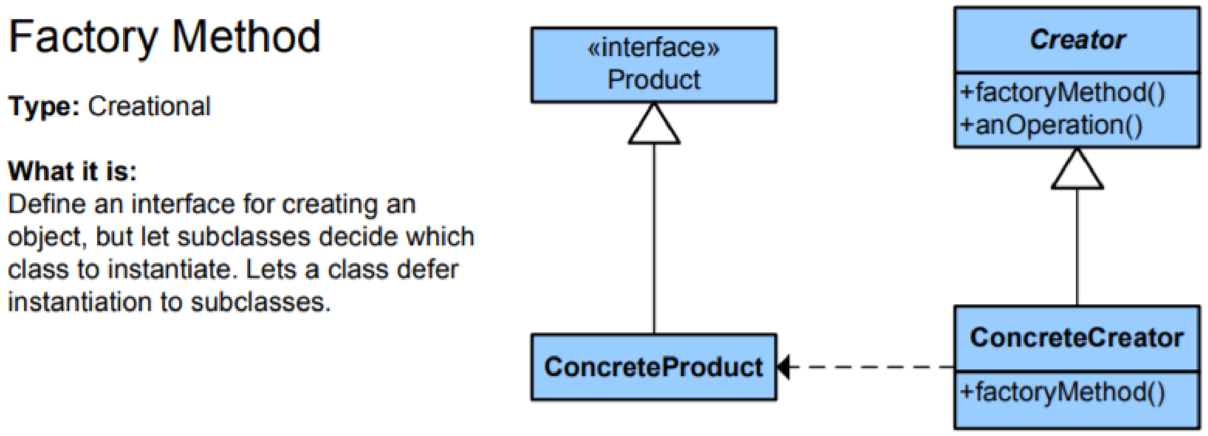


Abstract Factory

Type: Creational

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

[Factory Method](#)



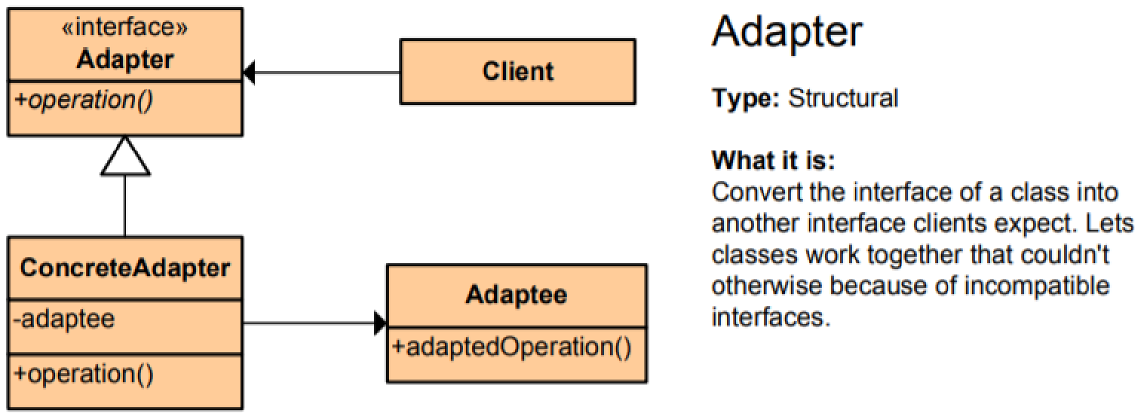
Factory Method

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.

Structural

[Adapter](#)

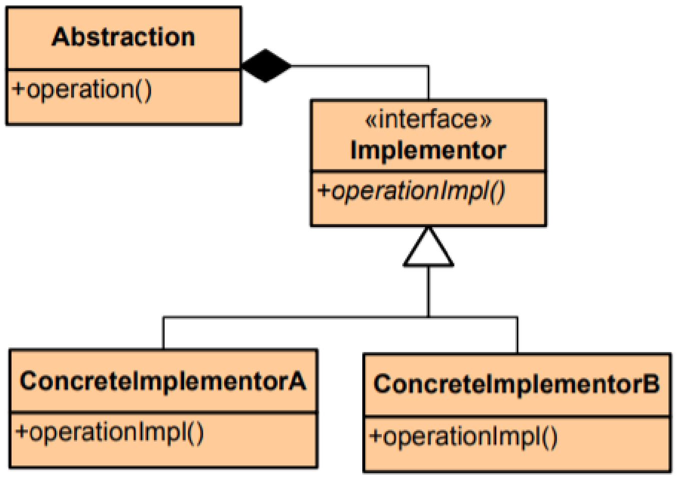


Adapter

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.

Bridge

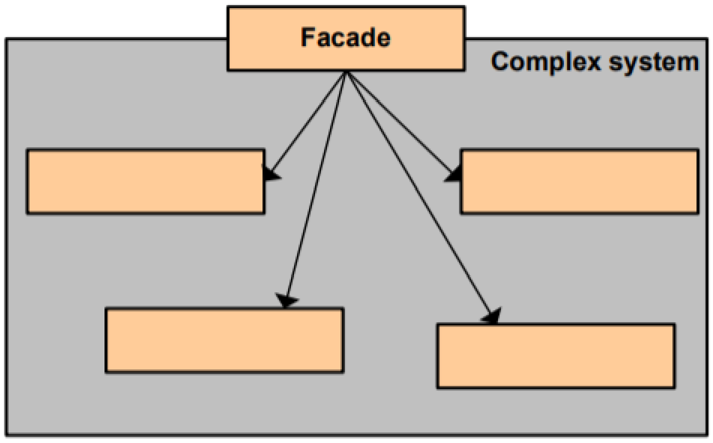


Bridge

Type: Structural

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

Facade



Facade

Type: Structural

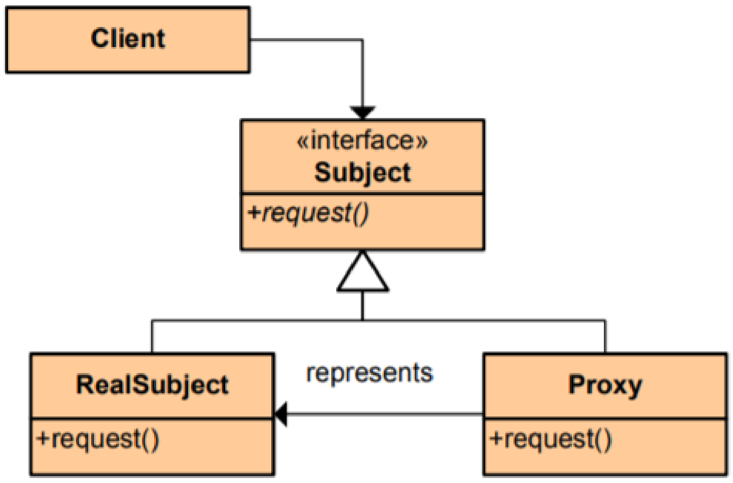
What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

Proxy

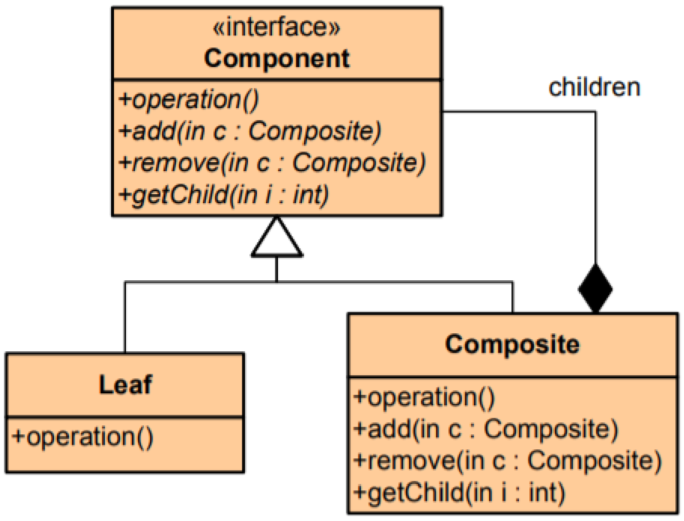
Proxy

Type: Structural

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



Composite



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.

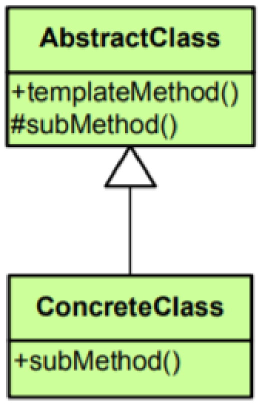
Behavioural

Template Method

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.

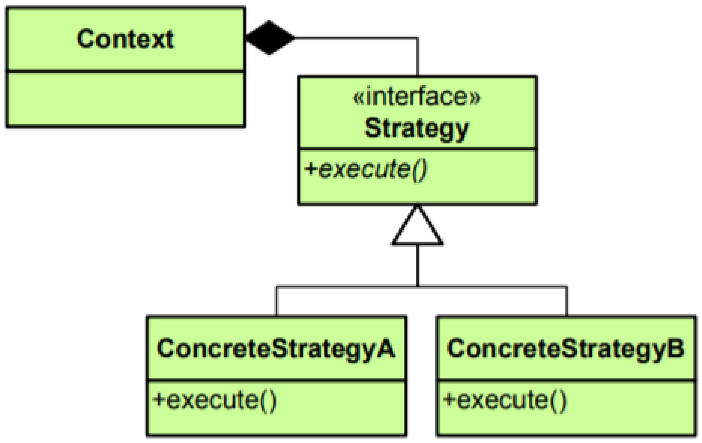


Strategy

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.

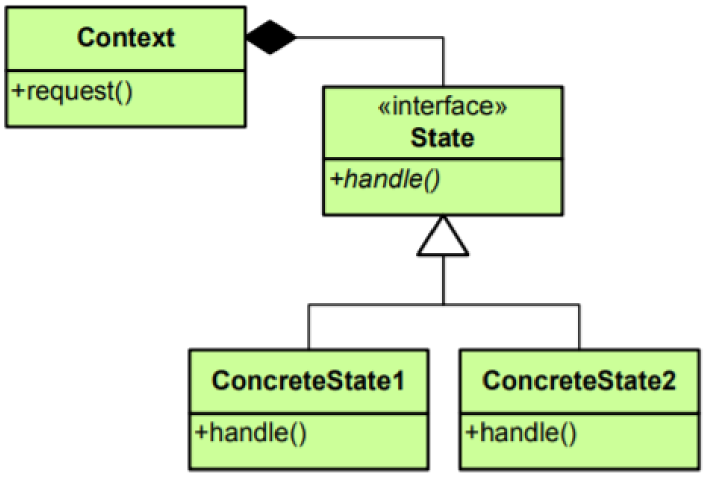


State

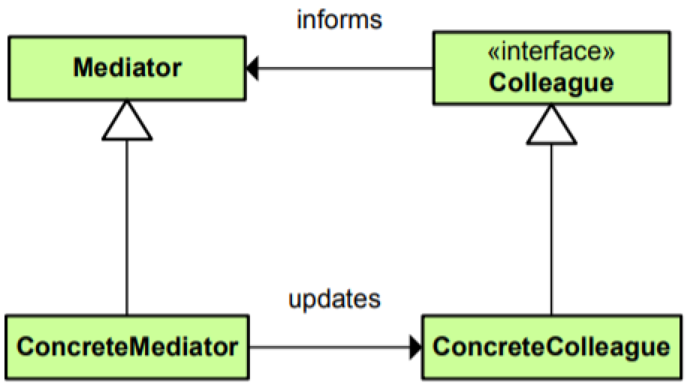
State

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.



Mediator



Mediator

Type: Behavioral

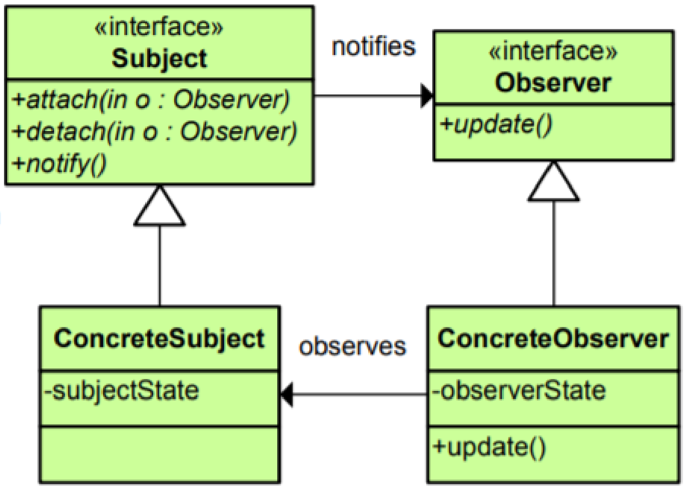
What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

Observer

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.

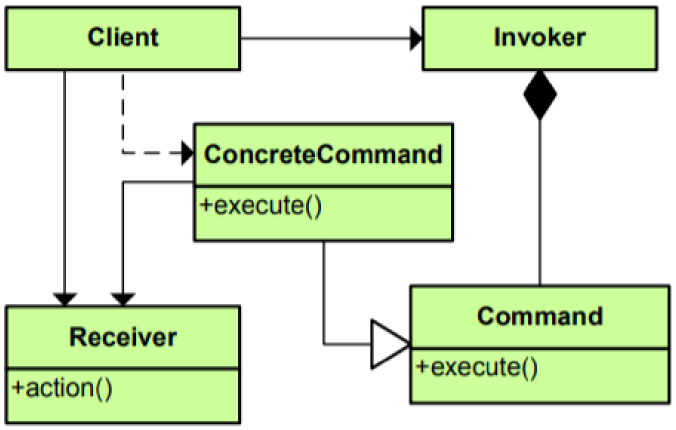


Command

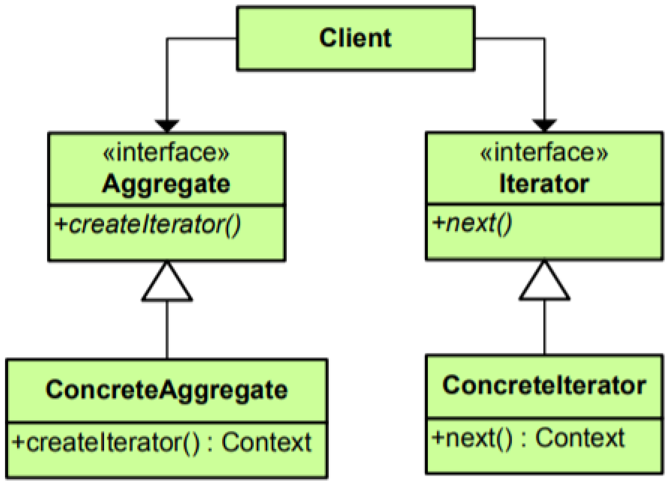
Command

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.



Iterator



Iterator

Type: Behavioral

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

Image Source