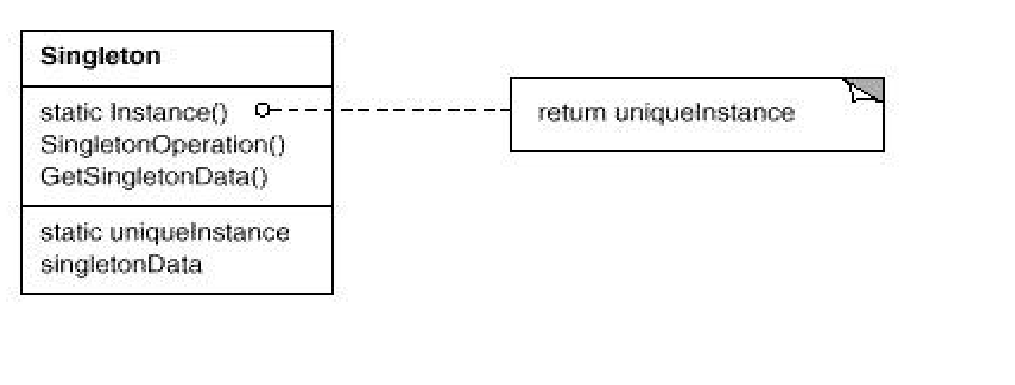
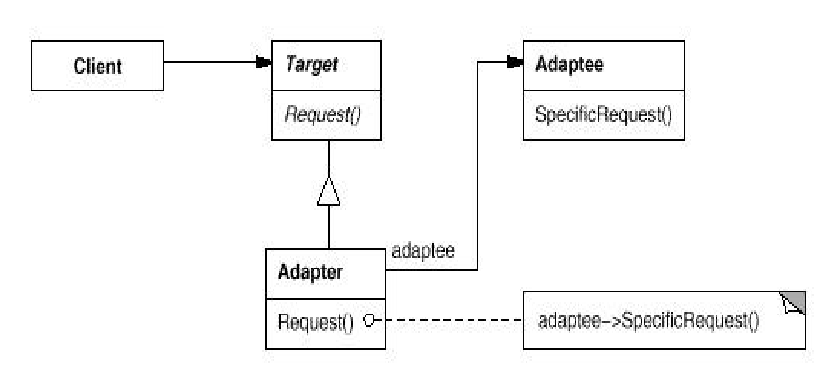
**Singleton Pattern**

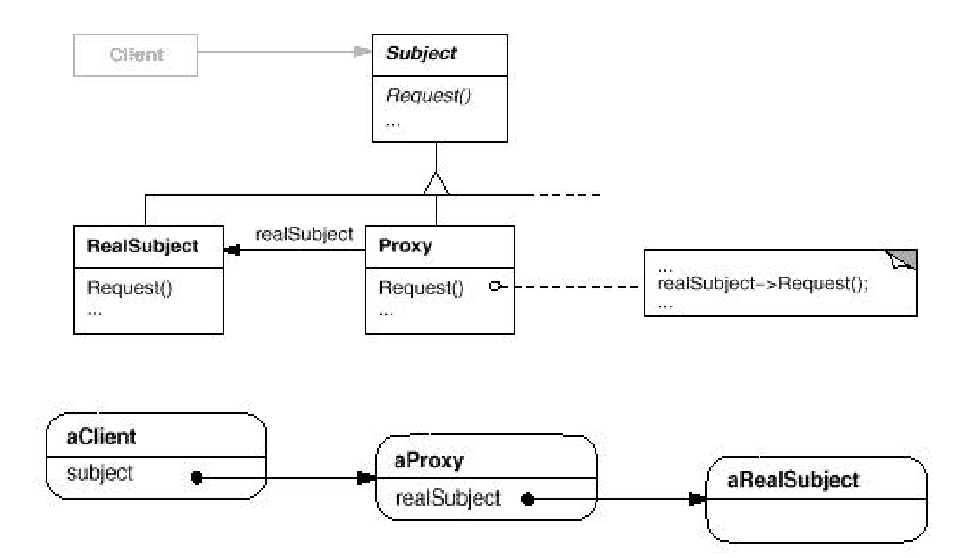
**Intent:** Ensure that a class have only one instance created and provide a global point to access to it.

**Adapter Pattern**

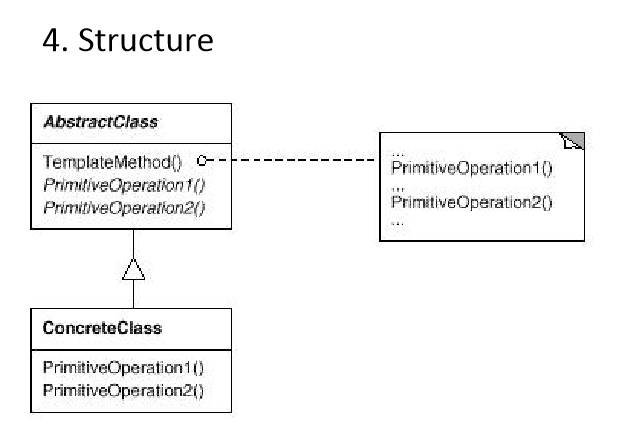
**Intent:** Convert the interface of a class in to another interface client expected and it let incompatible interfaces work togethers.

****

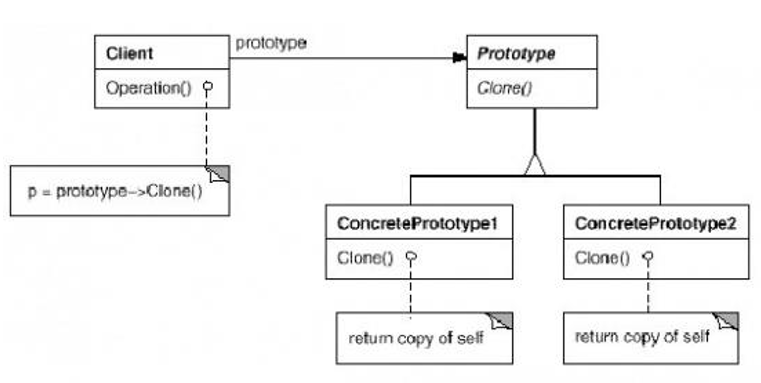
**Proxy Pattern**

**Intent:** Provide surrogate or place holder for another object to control access to it.

**Template Pattern**

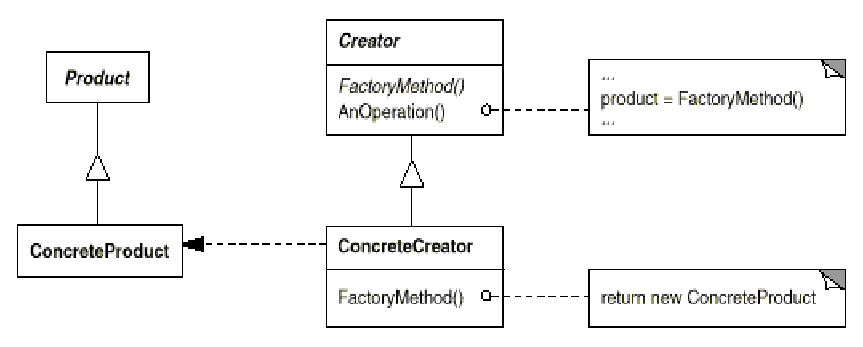
**Intent:** Define the skeleton of the algorithms in an operation and deferring some steps to subclass without changing the algorithms structure.

**Prototype Pattern**

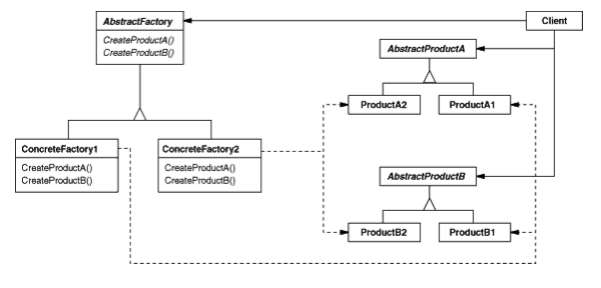
**Intent:** Specifies the kind of object from prototypical instance, and create new object by copying from this prototype.

**Factory Method Pattern**

**Intent:** Define an interface for creating object but let subclass decide which class to instantiate.

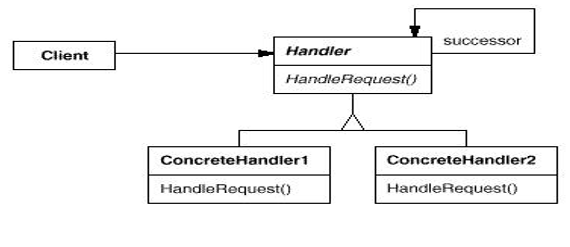


**Abstract Factory Pattern**

**Intent:** Provide an interface for creating families of related or dependent without specifies their concrete class.

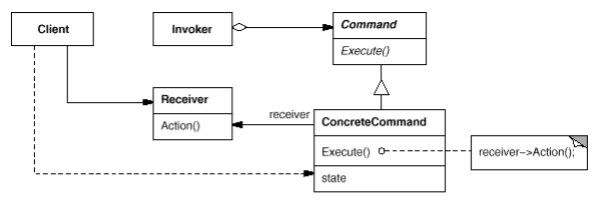
**Chain of Responsibility Pattern**

**Intent:** 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.



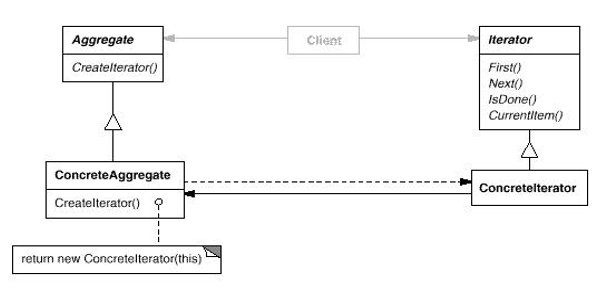
**Command Pattern**

**Intent:** Encapsulation a request as object thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.



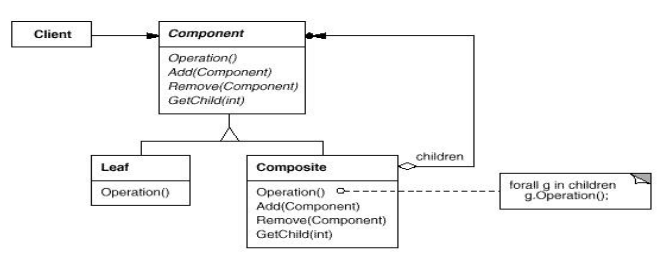
**Iterator Pattern**

**Intent:** Provide a way to access the elements of an aggregate object (collection) sequentially without exposing its underlying representation.



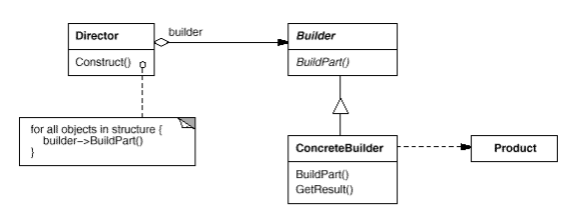
**Composite Pattern**

**Intent:** Compose objects into tree structures to represent part-whole or parent-child hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.



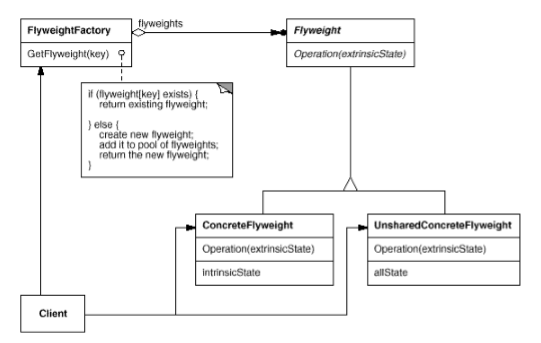
**Builder Pattern**

**Intent:** Separate the construction of a complex object from its representation so that the same construction process can create different representations.



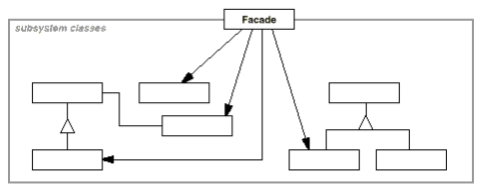
**Flyweight Pattern**

**Intent:** Use sharing to support large numbers of fine-grained objects efficiently.



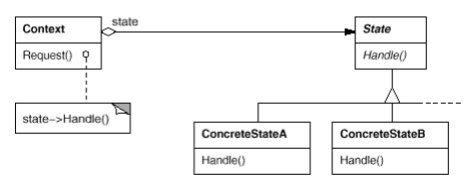
**Facade Pattern**

**Intent:** Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.



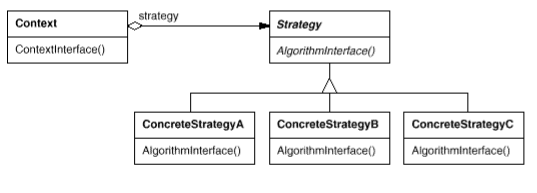
**State Pattern**

**Intent:** Allow an object to alter its behavior when it internal state change. The object will appear to change its class.



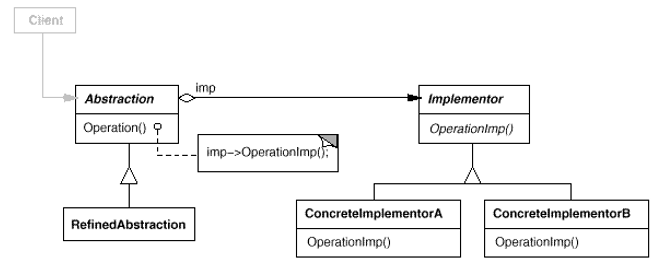
**Strategy Pattern**

**Intent:** Define a family of algorithms, encapsulate each one and make them interchangeable and it let algorithms vary independently from client that use it.



**Bridge Pattern**

**Intent:** Decouple an abstraction from its implementation so that the two can vary independently.



**Decorator Pattern**

**Intent:** Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

