

Creational Patterns

Pattern	Description	Image
Abstract Factory	Creates an instance of several families of classes	<pre> classDiagram class Client class AbstractFactory { +CreateProductA() +CreateProductB() } class AbstractProductA class AbstractProductB class ProductA1 class ProductA2 class ProductB1 class ProductB2 class ConcreteFactory1 { +CreateProductA() +CreateProductB() } class ConcreteFactory2 { +CreateProductA() +CreateProductB() } Client --> AbstractFactory Client --> AbstractProductA Client --> AbstractProductB AbstractFactory < -- ConcreteFactory1 AbstractFactory < -- ConcreteFactory2 AbstractProductA < -- ProductA1 AbstractProductA < -- ProductA2 AbstractProductB < -- ProductB1 AbstractProductB < -- ProductB2 ConcreteFactory1 ..> ProductA1 ConcreteFactory1 ..> ProductA2 ConcreteFactory2 ..> ProductB1 ConcreteFactory2 ..> ProductB2 </pre>
Builder	Separates object construction from its representation	<pre> classDiagram class Director { +Construct() } class Builder { +BuildPart() } class ConcreteBuilder { +BuildPart() +GetResult() } class Product Director o-- Builder : builder Builder < -- ConcreteBuilder ConcreteBuilder ..> Product note for Director "foreach item in structure\nbuilder.BuildPart()" </pre>
Factory Method	Creates an instance of several derived classes	<pre> classDiagram class Product class ConcreteProduct class Creator { +FactoryMethod() +AnOperation() } class ConcreteCreator { +FactoryMethod() } Product < -- ConcreteProduct Creator < -- ConcreteCreator ConcreteCreator ..> ConcreteProduct note for Creator "product = FactoryMethod()" note for ConcreteCreator "return new ConcreteProduct" </pre>
Prototype	A fully initialized instance to be copied or cloned	<pre> classDiagram class Client { +Operation() } class Prototype { +Clone() } class ConcretePrototype1 { +Clone() } class ConcretePrototype2 { +Clone() } Client --> Prototype : prototype Prototype < -- ConcretePrototype1 Prototype < -- ConcretePrototype2 note for Client "p=prototype.Clone()" note for ConcretePrototype1 "return copy of this" note for ConcretePrototype2 "return copy of this" </pre>

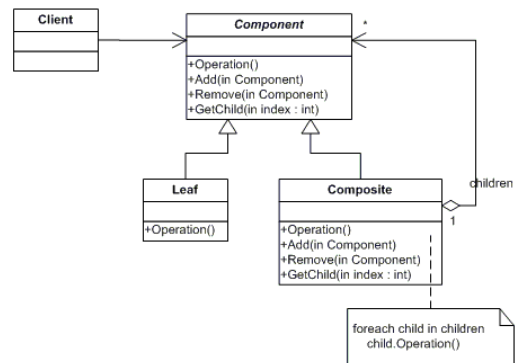
Singleton	A class of which only a single instance can exist	<pre> class Singleton { -instance : Singleton -Singleton() +Instance() : Singleton } </pre>
---------------------------	---	---

Structural Patterns

Pattern	Description	Image
Adapter	Match interfaces of different classes	<pre> classDiagram class Client class Target { +Request() } class Adapter { +Request() } class Adaptee { +SpecificRequest() } Client --> Target : target Adapter -- > Target Adapter --> Adaptee : adaptee note for Adapter "adaptee.SpecificRequest()" </pre>
Bridge	Separates an object's interface from its implementation	<pre> classDiagram class Client class Abstraction { +Operation() } class Implementor { +OperationImp() } class RefinedAbstraction class ConcreteImplementorA { +OperationImp() } class ConcreteImplementorB { +OperationImp() } Client --> Abstraction Abstraction < -- RefinedAbstraction Abstraction o-- Implementor : implementor Implementor < -- ConcreteImplementorA Implementor < -- ConcreteImplementorB note for Abstraction "implementor.OperationImp()" </pre>

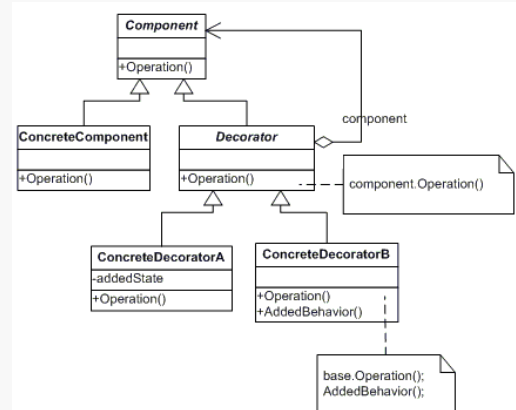
Composite

A tree structure of simple and composite objects



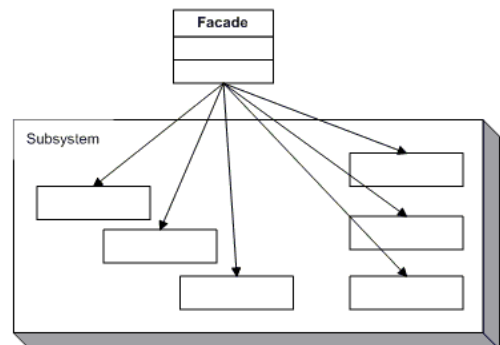
Decorator

Add responsibilities to objects dynamically



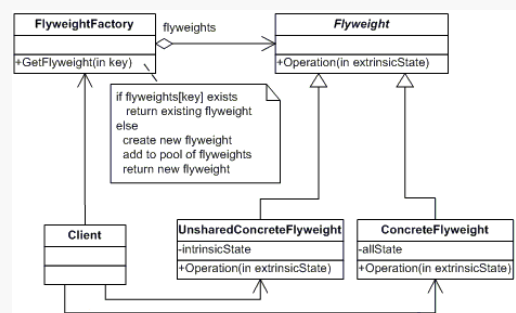
Facade

A single class that represents an entire subsystem



Flyweight

A fine-grained instance used for efficient sharing



Proxy	An object representing another object	
-----------------------	---------------------------------------	--

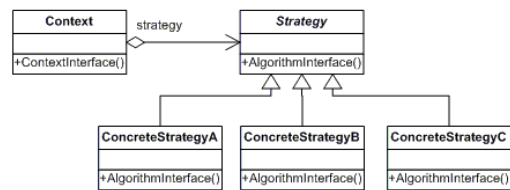
Behavioral Patterns

Pattern	Description	Image
Chain of Resp.	A way of passing a request between a chain of objects	
Command	Encapsulate a command request as an object	
Interpreter	A way to include language elements in a program	

Iterator	<p>Sequentially access the elements of a collection</p>	<pre> classDiagram class Aggregate { +CreateIterator() } class Client class Iterator { +First() +Next() +IsDone() +CurrentItem() } class ConcreteAggregate { +CreateIterator() } class ConcreteIterator Aggregate < -- ConcreteAggregate Iterator < -- ConcreteIterator Client --> Aggregate Client --> Iterator ConcreteAggregate ..> ConcreteIterator note for ConcreteAggregate "return new ConcreteIterator(this)" </pre>
Mediator	<p>Defines simplified communication between classes</p>	<pre> classDiagram class Mediator class Colleague class ConcreteMediator class ConcreteColleague1 class ConcreteColleague2 Mediator < -- ConcreteMediator Colleague < -- ConcreteColleague1 Colleague < -- ConcreteColleague2 ConcreteMediator --> Colleague : mediator </pre>
Memento	<p>Capture and restore an object's internal state</p>	<pre> classDiagram class Originator { -state +SetMemento(in Memento) +CreateMemento() } class Memento { -state +GetState() +SetState() } class Caretaker Originator ..> Memento Caretaker --> Memento : memento note for Originator "return new Memento(state)" note for Memento "state = m.GetState()" </pre>
Observer	<p>A way of notifying change to a number of classes</p>	<pre> classDiagram class Subject { +Attach(in Observer) +Detach(in Observer) +Notify() } class ConcreteSubject { -subjectState +GetState() } class Observer { +Update() } class ConcreteObserver { -observerState +Update() } Subject < -- ConcreteSubject Observer < -- ConcreteObserver ConcreteSubject --> ConcreteObserver : subject note for ConcreteSubject "return subjectState" note for ConcreteObserver "observerState = subject.GetState()" note for Subject "foreach o in observers o.Update()" </pre>
State	<p>Alter an object's behavior when its state changes</p>	<pre> classDiagram class Context { +Request() } class State { +Handle() } class ConcreteStateA { +Handle() } class ConcreteStateB { +Handle() } Context o--> State : state State < -- ConcreteStateA State < -- ConcreteStateB note for Context "state.Handle()" </pre>

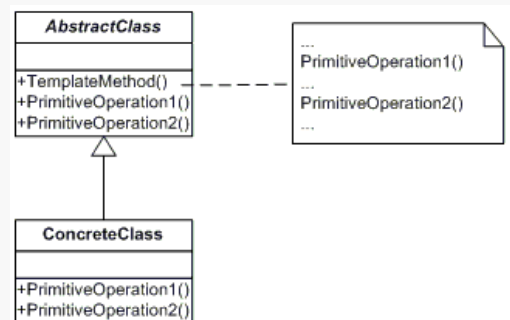
Strategy

Encapsulates an algorithm inside a class



Template Method

Defer the exact steps of an algorithm to a subclass



Visitor

Defines a new operation to a class without change

