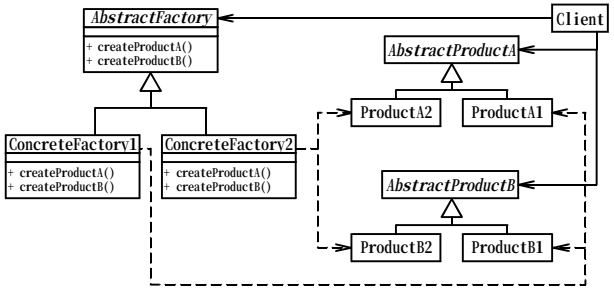
chapter-2

デザインパターン一覧

[生成に関するパターン]

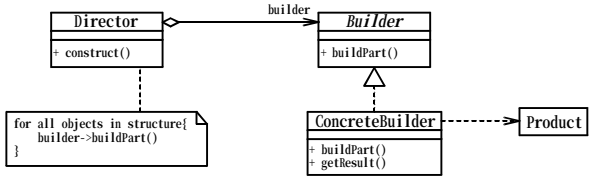
■ Abstract Factory パターン

互いに関連したり依存し合うオブジェクト群を，その具象クラスを明確にせずに生成するためのインタフェースを提供する．



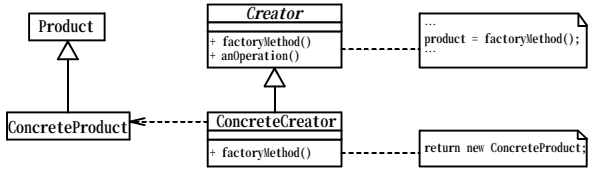
■ Builder パターン

複合オブジェクトについて，その作成過程を表現形式に依存しないものにすることにより，同じ作成過程で異なる表現形式のオブジェクトを生成できるようにする．



■ Factory Method パターン

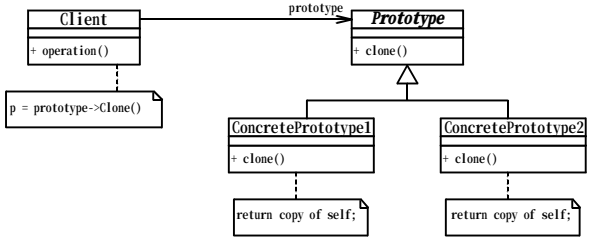
オブジェクトを生成する時のインタフェースだけを規定して，実際にどのクラスをインスタンス化するかはサブクラスが決めるようにする．



出典：「デザインパターン」，Eric Gamma 他，ソフトバンクパブリッシング

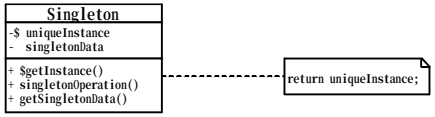
■ Prototype パターン

生成すべきオブジェクトの種類を原形となるインスタンスを使って明確にし，それをコピーすることで新たなオブジェクトの生成を行う．



■ Singleton パターン

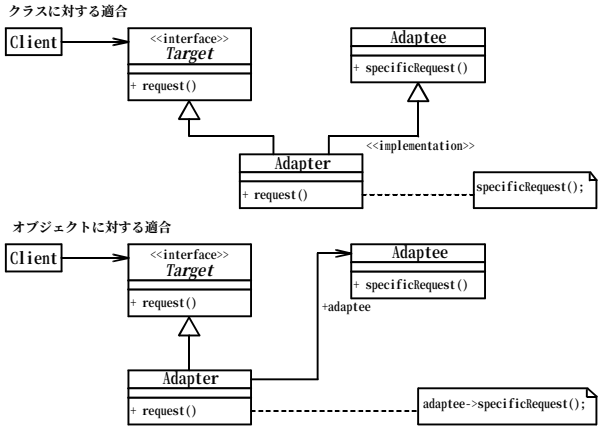
あるクラスに対してインスタンスが1つしか存在しないことを保証し，それにアクセスするためのグローバルな方法を提供する．



[構造に関するパターン]

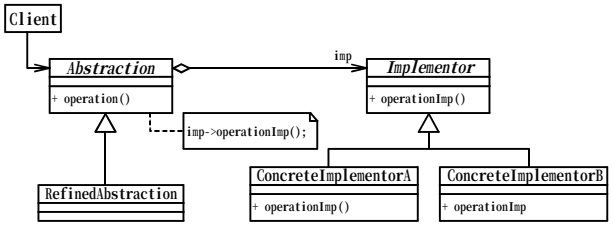
■ Adapter パターン

あるクラスのインタフェースを，クライアントが求める他のインタフェースへ変換する．
Adapter パターンは，インタフェースに互換性のないクラス同士を組み合わせることができるようにする．



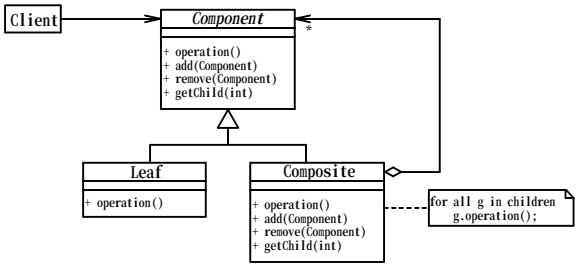
■ Bridge パターン

抽出されたクラスと実装を分離して，それらを独立に変更できるようにする．



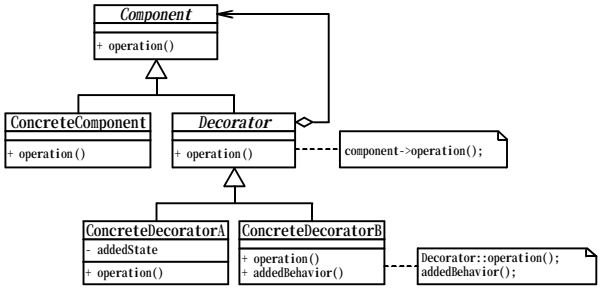
■ Composite パターン

部分－全体階層を表現するために，オブジェクトを木構造に組み立てる．
Composite パターンにより，クライアントは個々のオブジェクトとオブジェクトを合成したものを一様に扱うことができるようになる．



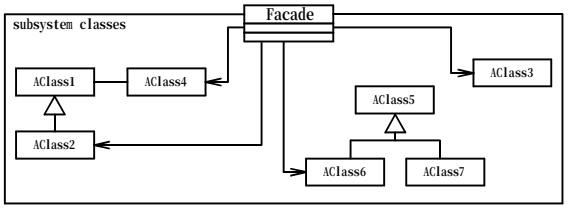
■ Decorator パターン

オブジェクトに責任を動的に追加する．サブクラス化よりも柔軟な機能拡張方法を提供する．



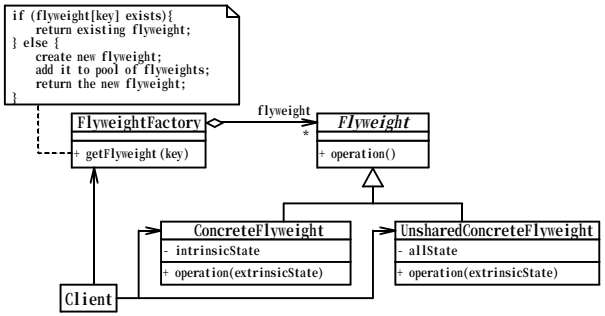
■ Facade パターン

サブシステム内に存在する複数のインタフェースに1つの統一インタフェースを与える．
サブシステムの利用を容易にするための高レベルインタフェースを定義する．



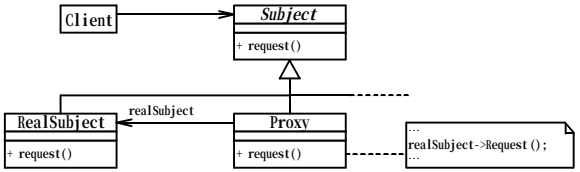
■ Flyweight パターン

多数の細かいオブジェクトを効率良くサポートするために共有を利用する．



■ Proxy パターン

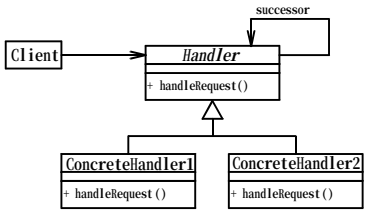
あるオブジェクトへのアクセスを制御するために，そのオブジェクトの代理，または入れものを提供する．



[振舞に関するパターン]

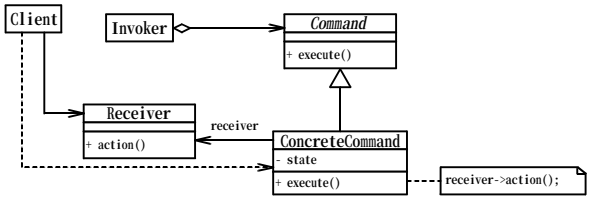
■ Chain Of Responsibility パターン

1つ以上のオブジェクトに要求を処理する機会を与えることにより，要求を送信するオブジェクトと受信するオブジェクトの結合を避ける．受信する複数のオブジェクトをチェーン状につなぎ，あるオブジェクトがその要求を処理するまで，そのチェーンに沿って要求を渡して行く．



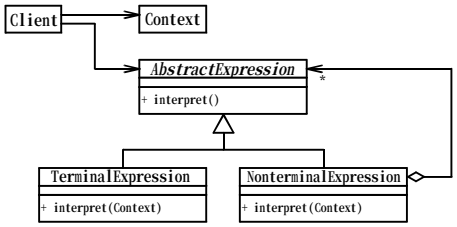
■ Command パターン

要求をオブジェクトとしてカプセル化することによって，異なる要求や，要求からなるキューやログにより，クライアントをパラメータ化する．また，取消可能なオペレーションをサポートする．



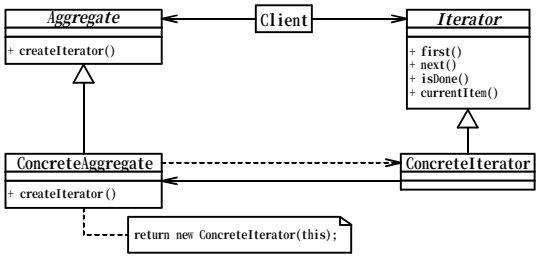
■ Interpreter パターン

言語に対して，文法表現と，それを使用して文を解釈するインタプリタを一緒に定義する．



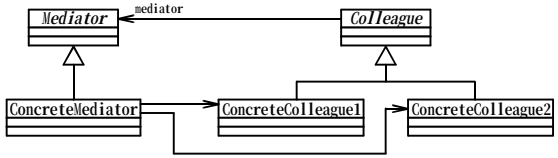
■ Iterator パターン

集約オブジェクトが基にある内部表現を公開せずに，その要素に順にアクセスする方法を提供する．



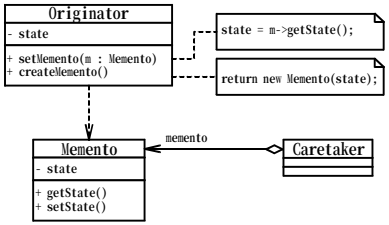
■ Mediator パターン

オブジェクト群の相互作用をカプセル化するオブジェクトを定義する．オブジェクト同士がお互いを明示的に参照し合うことがないようにして，結合度を低めることを促進する．それにより，オブジェクトの相互作用を独立に変えることができるようにする．



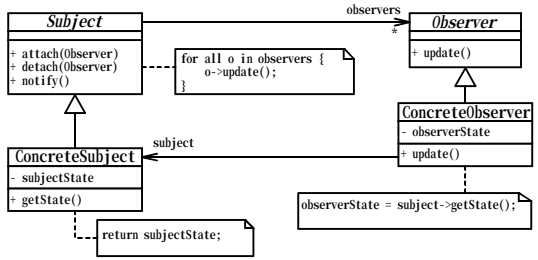
■ Memento パターン

カプセル化を破壊せずに，オブジェクトの内部状態を捉えて外面化しておき，オブジェクトを後にこの状態に戻すことができるようにする．



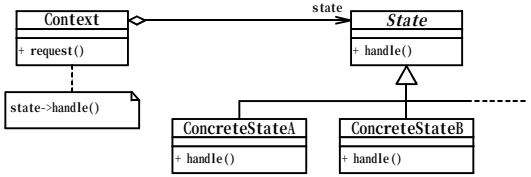
■ Observer パターン

あるオブジェクトが状態を変えた時に，それに依存するすべてのオブジェクトに自動的にそのことが知らされ，また，それらが更新されるように，オブジェクト間に1対多の依存関係を定義する．



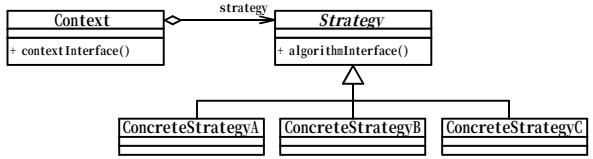
■ State パターン

オブジェクトの内部状態が変化した時に，オブジェクトが振舞を変えるようにする．クラス内では，振舞の変化を記述せず，状態を表すオブジェクトを導入することでこれを実現する．



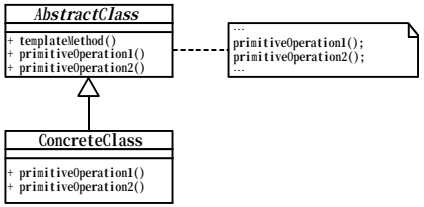
■ Strategy パターン

アルゴリズムの集合を定義し，各アルゴリズムをカプセル化して，それらを交換可能にする．アルゴリズムを，それを利用するクライアントからは独立に変更することができるようにする．



■ Template Method パターン

1つのオペレーションにアルゴリズムのスケルトンを定義しておき，その中の幾つかのステップについては，サブクラスでの定義にまかせることにする．アルゴリズムの構造を変えずに，アルゴリズム中のあるステップをサブクラスで再定義する．



■ Visitor パターン

あるオブジェクト構造上の要素で実行されるオペレーションを表現する．オペレーションを加えるオブジェクトのクラスに変更を加えずに，新しいオペレーションを定義することができるようにする．