オブジェクト指向開発おける アーキテクチャ設計のパターン:

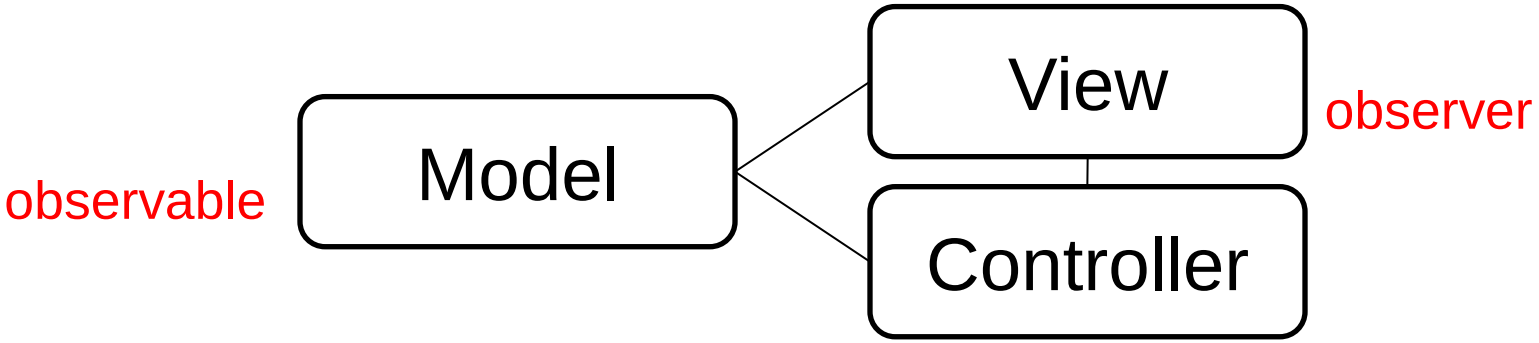
MVCモデル

- Model: データ操作
- View: 出力
- Controller: 入力

M,V,Cを独立して実装．特に，GUIのシステムに向いている．

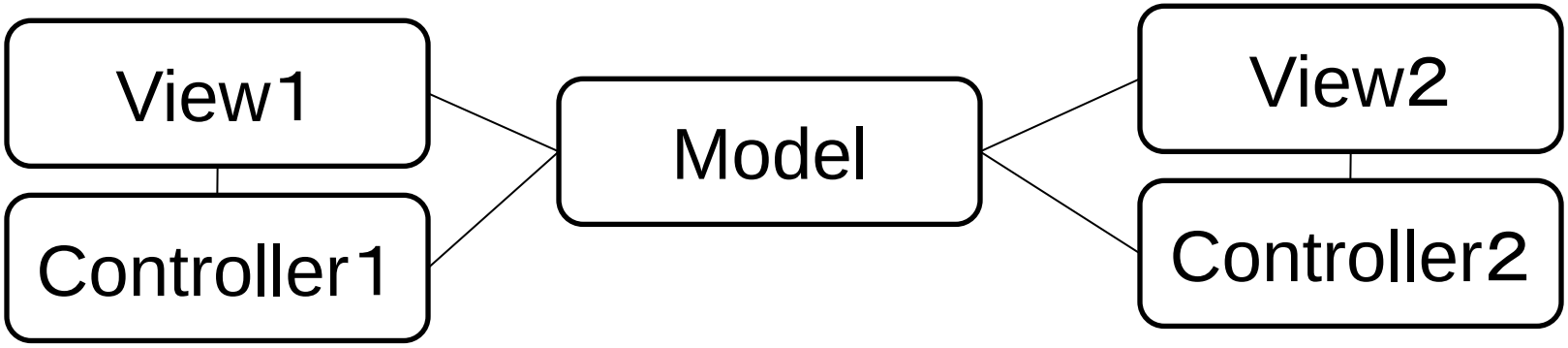
MVCモデル(1)

- Model, View, Controllerの3つに分けて，システムを設計する方法
  - できるだけ，独立に設計する
    - Model: データ操作
    - View: 出力．モデルの状態に応じて出力を変化させる．
    - Controller: 入力



MVCモデル(2)

- 例：スクリーンエディターの作成



- Model（データ操作のクラス）は同一で，VCを交換することによって，全く見た目の異なる（例えば，GUIとCUI)エディタを実現可能.

MVC

- 利点
  - インタフェースとモデル部分の分離
  - インタフェースの交換が容易

- 欠点
  - 複雑度の増大
  - 更新が過度に起こる可能性
  - ビューとコントローラの依存性が大きくなりやすい
  - VCとモデルの直接の結合
    - 次のPACでは，それを解決する方法を提案

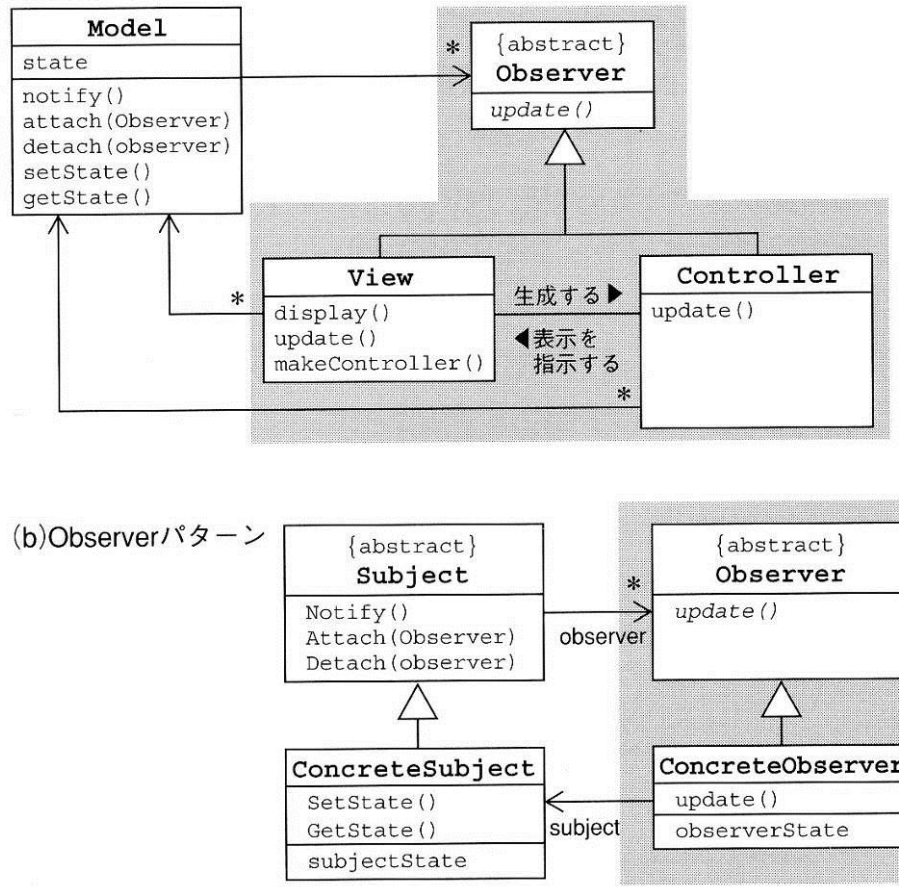
MVCモデル と observerパターン

- **MVCモデル:**
  Model, View, Controllerを分離して設計する設計モデル

  C, VがboundaryMがcontrolに相当する.
  - SmallTalkで用いられた
  - Javaの設計思想にも取り入れられている.

- **Observerパターン:**
  デザインパターンの一種
  observer がsubjectの状態変化を監視して, 変化があれば, 何らかのアクションを起こす.

図 C-1 MVC モデルと Observer パターン



MVCモデル と observerパターン

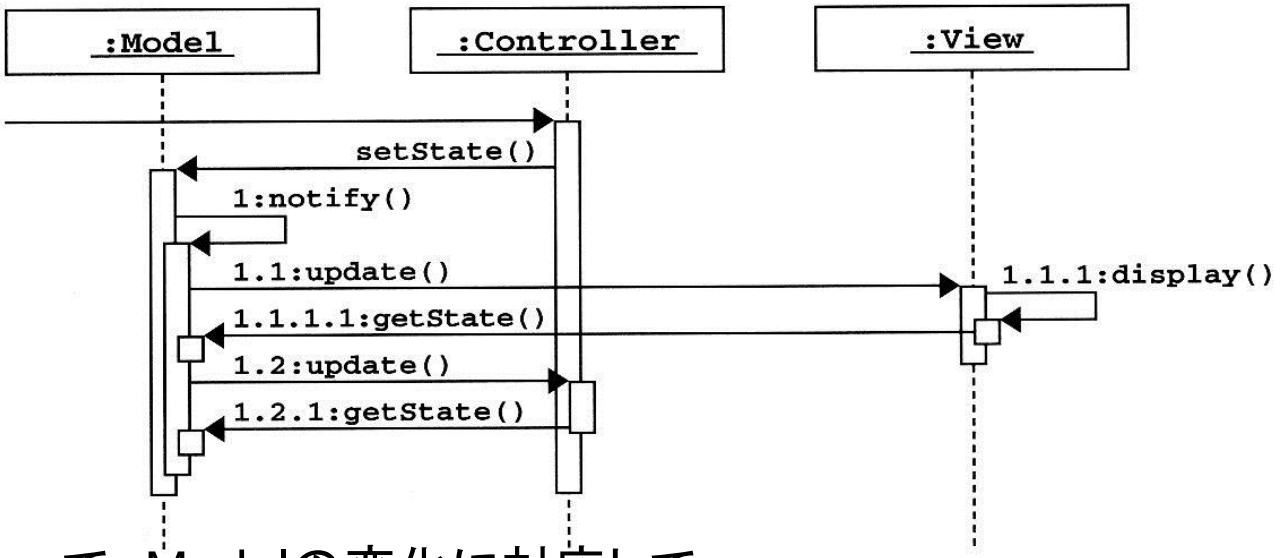
Model: データ構造の中心 （基本データ構造 と その操作）

View: 表示のクラス （GUIの場合は GUI部品のクラス）

Controller: 入力のクラス （GUIの場合はイベントハンドリングのクラス）

図 C-2

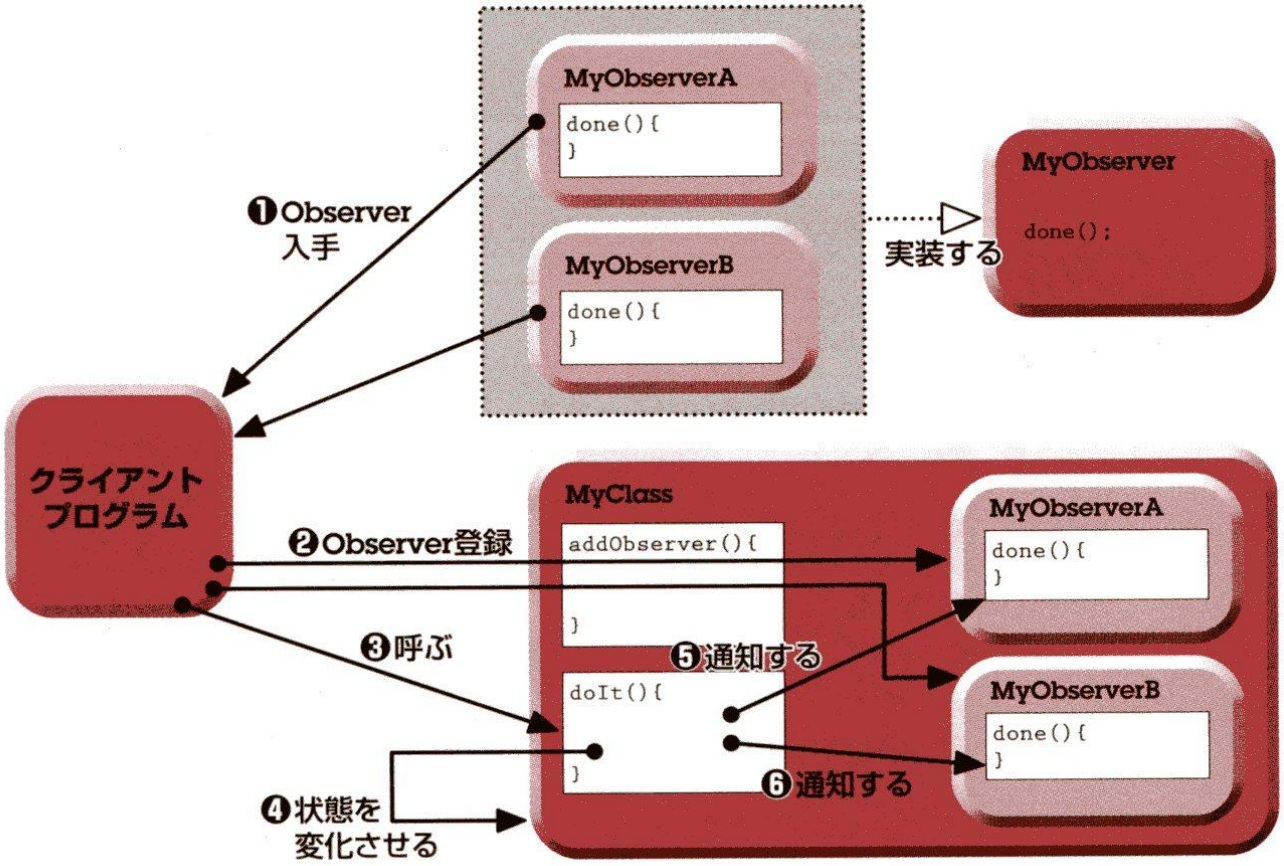
MVC モデルにおけるデータ変化通知のシーケンス図



Observer パターンによって Modelの変化に対応して View を更新(update)する

Observer パターン ＜状態の変化を通知＞

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通知されるようなパターン．



Observerパターン (2)

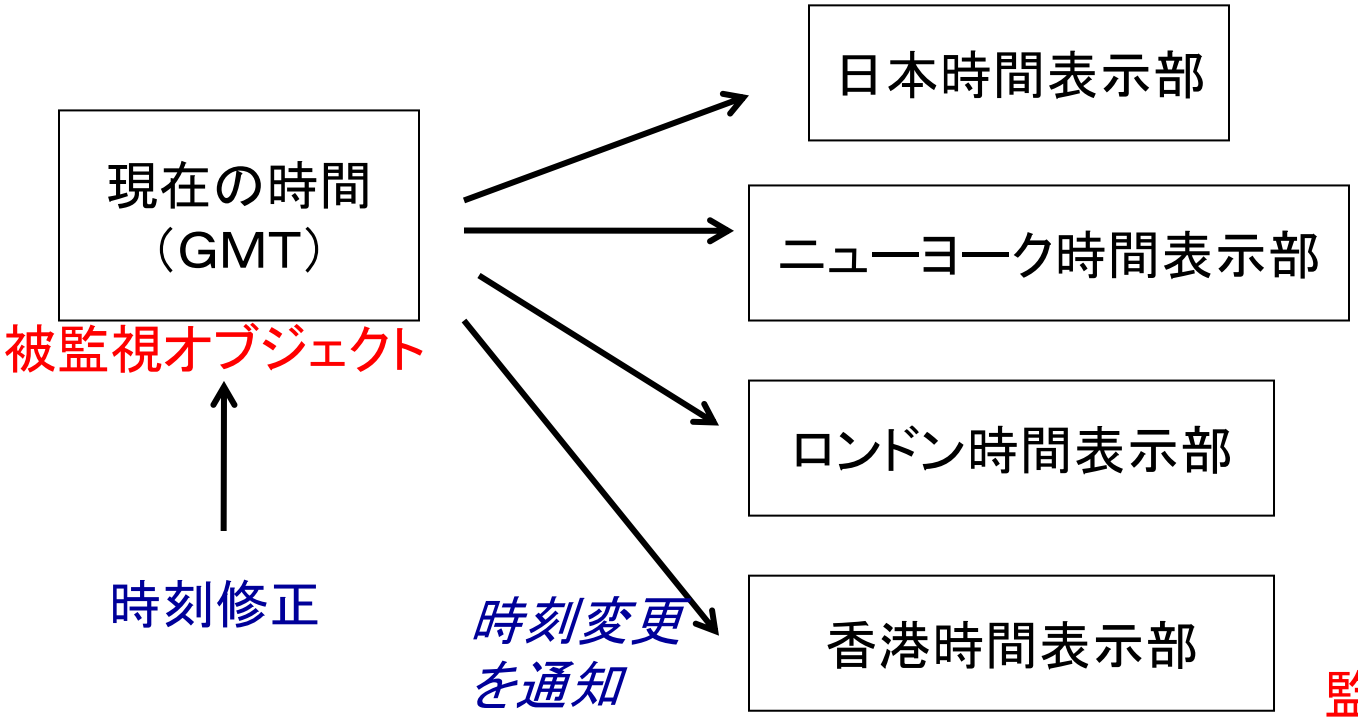
```
// 監視する(状態変化を通知
// される)オブジェクトのクラス
interface Observer {
   void update(int u);
} // doneは更新時に呼ばれ
//   るメソッド
class ObserverA
      implements Observer{
  public void update(int u){
     System.out.println(
        "A: updated : "+u); }
}
class ObserverB
      implements Observer{
  public void update(int u){
     System.out.println(
        "B: updated : "+u); }
}
```

```
// 状態変化を通知するクラス(被監視)
class MyClass{
  int value=0;
  Vector<Observer> observers=
                new Vector<Observer>();
  // observerの登録メソッド
  void addObserver(Observer ob){
     observers.add(ob); }
  // observer への更新通知メソッド
  void doIt() {
     value++;
     for(int i=0;i<observers.size();i++){
          observers.get(i).update(value); }
   }
}
```

Observerを実装したクラスを最初に登録し，
更新時に登録クラスのメソッドdone()を呼ぶ

Observerパターン の 例

- あるオブジェクトの状態変化に伴って，他のオブジェクト(observer)も変化する
  - 例） 世界時計プログラム



Observerパターン (3)

- Javaの標準ライブラリ　Java.utilに
  Java.util.Observerインターフェース　と
  Java.util.Observableクラスがある.
  - 考え方は同じ. 更新を通知する方が, Obserableを継承する.
  - ただし, Observableがクラスであるので,
    更新を通知するクラスは他のクラスを継承できない.（欠点）(継承は1つのクラスのみなので)
- Observerパターンは, アーキテクチャパターンのMVCモデルと関係が深い