Observer

- Coupler des classes
  - Mais sans dépendances !
- En IR1/IG1, on codait directement (sans réfléchir ?)
- Puis, on a appris à utiliser la délégation
- L'observer va encore plus loin :
  - Je ne délègue plus une opération
  - Je "préviens" "quelqu'un" de quelque chose
  - Je ne sais pas QUI je préviens
  - Je ne sais donc pas CE qu'il fera

Observer: pourquoi

```
// les objets dépendants sont divers et leur mise à jour est codé « en dur », de manière static
class DivObserver {
   private int  m_div;
   public DivObserver( int div ) { m_div = div; }
   public void update( int val ) {  …}
}
class ModObserver {
  private int  m_mod;
   public ModObserver( int mod ) { m_mod = mod; }
   public void maj( int val, String msg ) {…}
}
class Subject {
   int  m_value;
   public Subject() {... }
   void set_value( int value ) {
      m_value = value;
   }
}
```

- Connaissance des observers par les clients
- Connaissance des actions que réalisent les observers
- Dépendances statiques
- redondances de code !

```
int main( void ) {
  DivObserver  div_obj = new DivObserver(0);
  ModObserver  mod_obj = new ModObserver(0);
  ...
  Subject  subj = new Subject();
  subj.set_value( 14 );
  div_obj.update( 14 );
  mod_obj.maj( 14, « ... » );
}
```

Mieux ?

```
// les objets dépendants sont divers et leur mise à jour est codé « en dur », de manière static
class DivObserver {
   private int  m_div;
   public DivObserver( int div ) { m_div = div; }
   public void update( int val ) {  …}
}
class ModObserver {
  private int  m_mod;
   public ModObserver( int mod ) { m_mod = mo
   public void maj( int val, String msg ) {…}
}
class Subject {
   int  m_value;
   DivObserver  m_div_obj = new DivObserver(0);
   ModObserver  m_mod_obj = new ModObserver(0);
   public Subject() {... }
   void set_value( int value ) {
      m_value = value;
      notify();
   }
   void notify() {
      m_div_obj.update( m_value );
      m_mod_obj.maj( m_value, « ... » );
  }
}
```

```
int main( void ) {
  DivObserver  div_obj = new DivObserver(0);
  ModObserver  mod_obj = new ModObserver(0);
  ...
  Subject  subj = new Subject();
  subj.set_value( 14 );
}
```

- Connaissance précise des observers
- Connaissance des actions que réalisent les observers
- Dépendances statiques

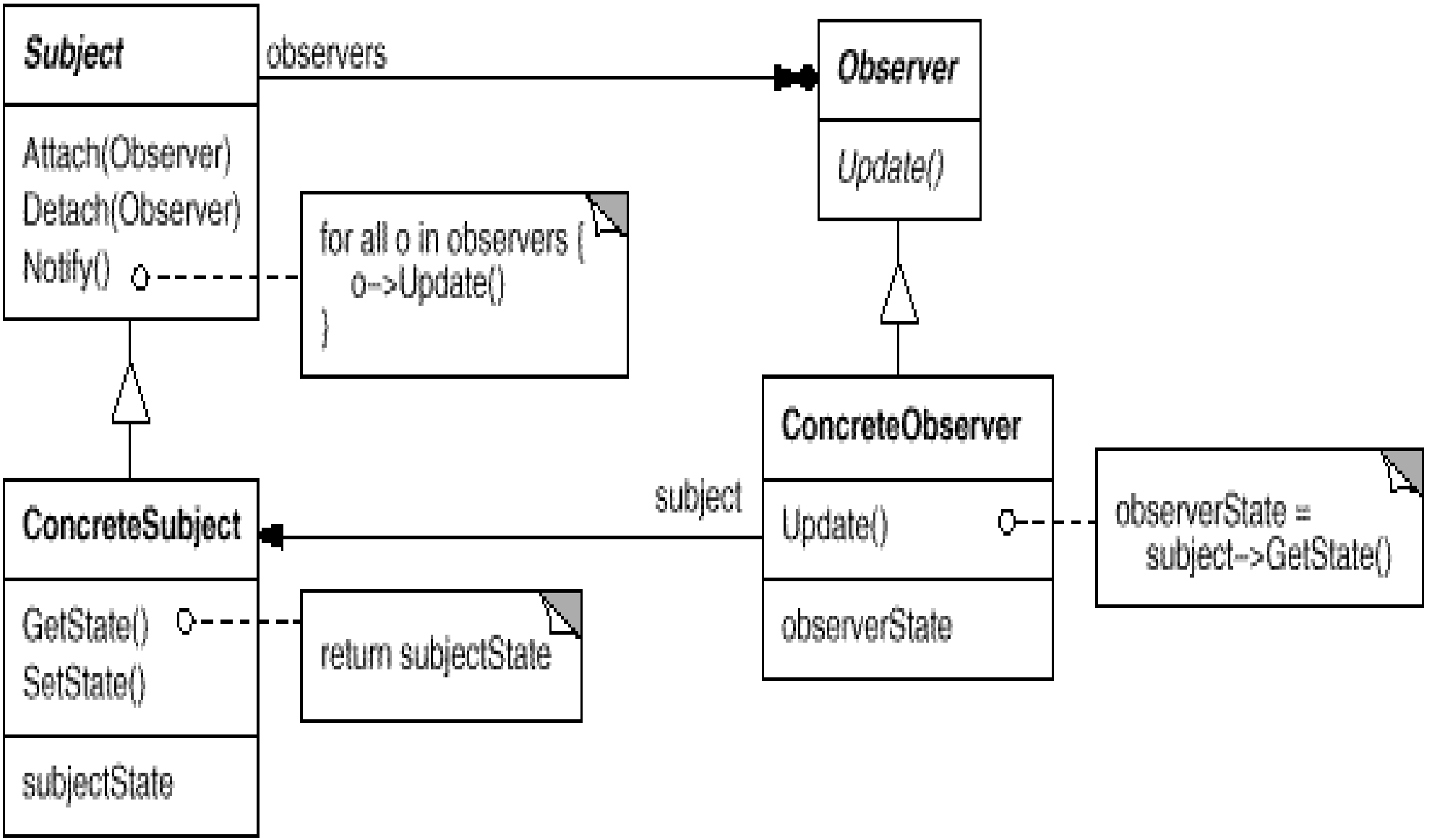
Observer : exemple

```
interface Observer {
  public void update( int value );
}
class Subject {
   int  m_value;
   List<Observer>  m_views;
   public void attach( Observer obs ) { m_views.add( obs ); }
   public void detach( Observer obs ) { m_views.remove( obs ); }
   void set_val( int value ) { m_value = value;  notify(); }
   void notify() {for (Observer o : m_views) o.update( m_value );  }
}
class DivObserver implements Observer {
   int  m_div;
  public DivObserver( Subject model, int div ) {
      model.attach( this );
      m_div = div;
   }
   void update( int v ) { maj(v) ;  }
}
class ModObserver implements Observer {
   int  m_mod;
   public ModObserver( Subject model, int mod ) {
      model.attach( this );
      m_mod = mod;
   }
   void update( int v ) {…}
}
```

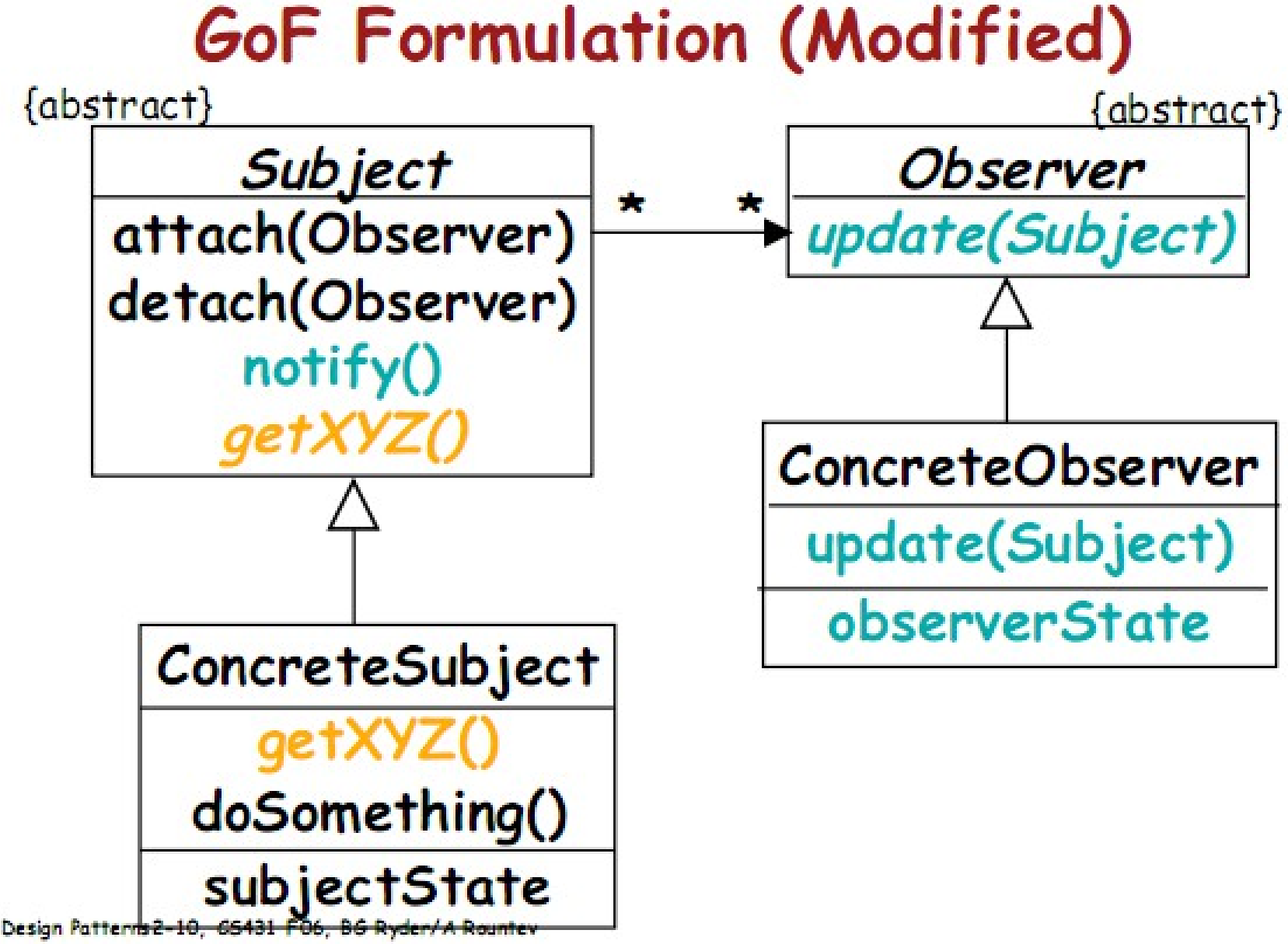
- couplage faible
- Connaissance uniquement des abstractions des observers
- aucune connaissance des actions que réalisent les observers
- pas de dépendances statiques

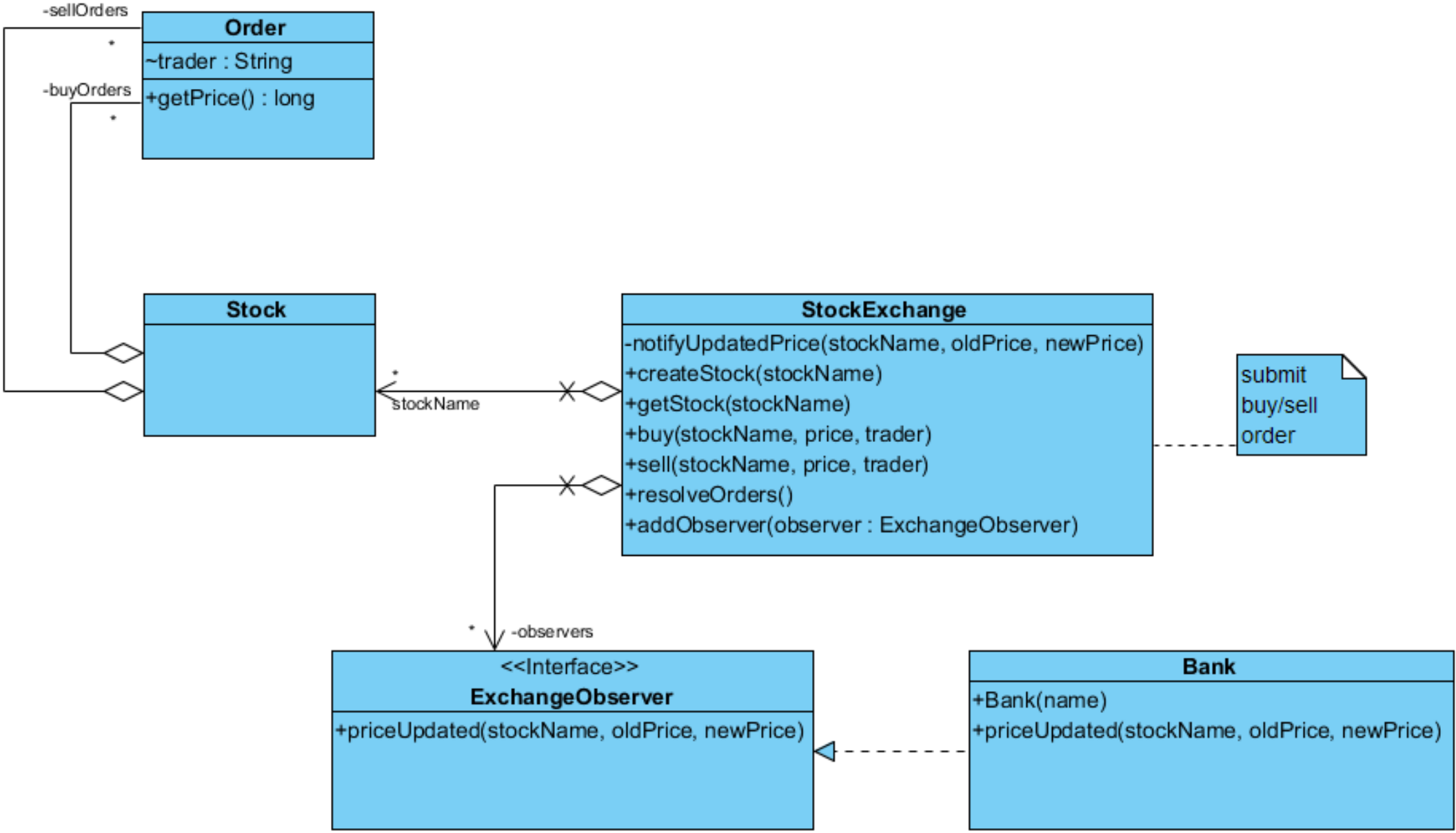
```
int main( void ) {
   Subject      subj = new Subject();
   DivObserver  divObs1( subj, 4 );
   DivObserver  divObs2( subj, 3 );
   ModObserver  modObs3( subj, 3 );
   ...
   subj.set_val( 14 );
}
```

Observer (GoF)



Variante


GoF Formulation (Modified)
{abstract}
Subject
attach(Observer)
detach(Observer)
notify()
getXYZ()
*
*
{abstract}
Observer
update(Subject)
ConcreteObserver
update(Subject)
observerState
ConcreteSubject
getXYZ()
doSomething()
subjectState


-sellOrders
*
-buyOrders
*
Order
~trader : String
+getPrice() : long
Stock
*
stockName
StockExchange
-notifyUpdatedPrice(stockName, oldPrice, newPrice)
+createStock(stockName)
+getStock(stockName)
+buy(stockName, price, trader)
+sell(stockName, price, trader)
+resolveOrders()
+addObserver(observer : ExchangeObserver)
submit buy/sell order
*
-observers
<<Interface>>
ExchangeObserver
+priceUpdated(stockName, oldPrice, newPrice)
Bank
+Bank(name)
+priceUpdated(stockName, oldPrice, newPrice)

End !

Synthèse

Observer

- Super moyen pour découpler !
- Open Close !
  - Je peux interagir avec un objet déjà existant
  - En l'écoutant
  - Alors qu'il ne me connaît pas ! Ni mon interface !

Observer : Facile ?

- Attention au mode de notification
  - Push ou pull
  - Impact important sur les performances
- Attention à la sémantique de notification :
  - Interface unique ou pas
  - Granularité des méthodes de notification
  - Impact important sur les performances