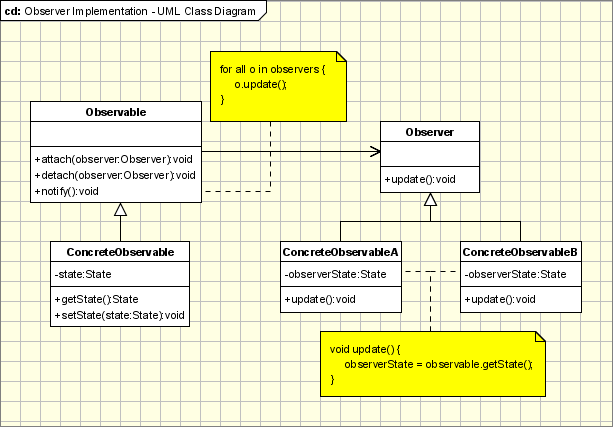
# Observer

**Definition**: The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Problem**: In one place or many places in the application we need to be aware of a system event or an application state change. We’d like to have a standard way of subscribing to listening for system events and a standard way of notifying the interested parties. The notification should be automated after an interested party subscribed to the system event or application state change. There also should be a way to unsubscribe.

**Solution**: After subscribing the listening objects will be notified by a way of the method call.



**Participants**:

* **Observable** — interface or abstract class defining the operations for attaching and de-attaching observers to the client. In the GOF book, this class/interface is known as Subject.
* **ConcreteObservable** — concrete Observable class. It maintains the state of the object and when a change in the state occurs it notifies the attached Observers.
* **Observer** — interface or abstract class defining the operations to be used to notify this object.
* **ConcreteObserverA**, **ConcreteObserverB** — concrete Observer implementations.
* Subject(Observable): has many observers, vector < **class** **Observer** \* > views; When notify is called, update of observer is called as well.
* Observer: DivObserver, ModObserver: use subject as a parameter of the constructor to call subject->attach(this).