I am watching you!

We need to be informed when certain things happen.

* Object’s fields changes
* Object does something
* Some external event occurs

We want to listen to events and get notified when they occur.

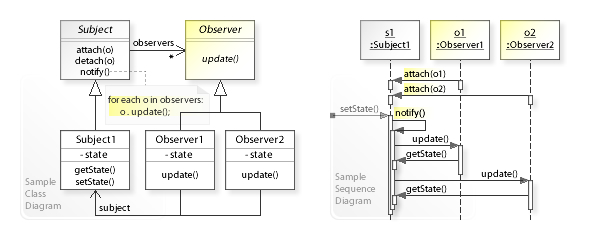
An **observer** is an object that wishes to be informed about events happening in the system.

The entity generating the events is an **observable**.

GOF Definition:

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Class Diagram:



Example:

Stock market with thousands of stocks needs to send updates to objects representing individual stocks.

The Subject (Publisher) sends many stocks to the observers.

The Observers (Subscribers) takes the ones they want and use them.

Loose coupling is the benefit.

The Subject (Publisher) doesn’t need to know anything about the observers (subscribers).

Negatives: The Subject (Publisher) may send updates that don’t matter to observers (subscribers).

Let’s now summarize the participants of the Observer pattern.

* **Subject** (Subject interface): Provides an interface to attach and detach **Observer** objects.
* **ConcreteSubject** (Product class): Implements the Subject interface. A **ConcreteSubject** sends notification to Observer objects when its state change.
* **Observer** (**Observer** interface): Provides an interface for objects that should be notified of changes in a Subject.
* **ConcreteObserver** (Bidder class): Implements Observer to receive notifications from the Subject and keep its state consistent with the state of the Subject.

**Observer vs Pub-Sub**

*In ‘Publisher-Subscriber’ pattern, senders of messages, called****publishers****, do not program the messages to be sent directly to specific receivers, called****subscribers.***

* In the **Observer** pattern, the O***bservers are aware of the Subject, also the Subject maintains a record of the Observers***. Whereas, in **Publisher/Subscriber**, publishers and subscribers ***don’t need to know each other***. They simply communicate with the help of message queues or broker.
* In **Publisher/Subscriber** pattern, components are loosely coupled as opposed to **Observer**pattern.
* **Observer** pattern is mostly implemented in a ***synchronous*** way, i.e. the Subject calls the appropriate method of all its observers when some event occurs. The **Publisher/Subscriber** pattern is mostly implemented in an ***asynchronous*** way (using message queue).
* **Observer**pattern needs to be implemented in a single application address space. On the other hand, the **Publisher/Subscriber** pattern is more of a cross-application pattern.