Observer Design Pattern:  
**Other name (if any)**

The observer pattern is also known as Dependents or Publish-Subscribe.

**What it does**

Observer design is a design pattern which is used to notify some objects of a class based on the the state change of another class. Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object.

**Where to use**

1. Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.  
2. Notification services which sends notification based on state change.  
3. Weather update provider.  
4. Notify me when a particular data is available  
5. Notify for stock update.  
6. Observers can be used to propagate changes across different components or nodes in a distributed system.  
7. Observers ensure that all dependent objects are updated consistently when the subject changes.  
8. You want to support broadcast communication.  
9. You need to support event handling or notification systems.  
10. The Observer pattern provides a foundation for implementing publish-subscribe messaging systems.  
11. Observers can be used to monitor changes in system state or sensor data and trigger appropriate actions.  
12. Observers can be used to log or audit (ting system keeps a close eye on what happens within a system or organization, recording important actions or events for later review.) changes in system behavior or data.  
13. Observers can be used to monitor changes in market conditions or trading data and trigger automated trading strategies or alerts.

**Steps**

1. A observable interface: A class which holds the data. There will be two properties. One is the data and another one is the list of observers interface object. There will be some function. Add, remove, notify, setData, getData.  
2. A observer interface: A class which get notification and perform certain task based on that notification. There will be one property which is observable object. There will be one method/function which is update.  
3. Observable concrete classes: It will implement observable interface. Add/subscribe/register method will add new observers to the observer object list. Remove/unsubscribe/unregister method will delete an observer from the observer object list. Setdata will set a certain property and call the notify method based on certain condition. Notify method will loop through the observer object list and call the update method of the observer.  
4. Observer concrete classses: It will implement the mothod of observer interface. It will perform certain task inside the update method using its own properties. For example send email/ send sms/ display output etc.  
5. Client code: It will create some observers based using observer interface and observer concrete class. Then create an observable object using observable interface and observable concrete class. Add the observers to the observable object. Then finally set the data of the observable object.

**Advantages**

1. It provides the support for broadcast-type communication.  
2. It promotes loose coupling between the subject (the object being observed) and its observers, allowing them to interact without knowing each other's details.  
3. Observers can be reused across different subjects, promoting code reuse and reducing duplication.  
4. It supports multiple observers, allowing for a scalable architecture where new observers can be easily added as needed.  
5. The pattern enables event-driven programming, where actions or updates in one part of the system can trigger responses in other parts.  
6. Observers receive updates from the subject immediately, ensuring that they stay synchronized with changes in real-time.  
7. It separates the logic for triggering events (subject) from the logic for handling those events (observers), promoting a cleaner separation of concerns.  
8. It facilitates complex interactions between objects by allowing multiple observers to react to changes in the subject in different ways.

**Disadvantages**

1. It provides the support for broadcast-type communication.  
2. Implementing the pattern can add complexity to the codebase, especially when dealing with multiple observers and subjects.  
3. The dynamic nature of the Observer pattern may make it harder for developers to understand and reason about the behavior of the system.  
4. In programs with multiple threads, you need to make sure everything stays synchronized to avoid problems.  
5. Implementing the Observer pattern typically requires additional classes and interfaces, increasing the overall code size.  
6. Understanding the Observer pattern and its intricacies may require developers to invest time in learning and mastering the concept.  
7. In long-running applications, continuously adding new observers without proper management can exhaust memory resources.

**Code**

Tutorialspoint

**Difference with similar pattern**

May be mediator design pattern [[later]]

**Diagram**

Tutorialspoint