**Observable Observer Design Pattern:** The Observer Design Pattern is a [behavioral design pattern](https://www.geeksforgeeks.org/behavioral-design-patterns) that defines a one-to-many dependency between objects so that when one object (the subject) changes state, all its dependents (observers) are notified and updated automatically.

*It primarily deals with the interaction and communication between objects, specifically focusing on how objects behave in response to changes in the state of other objects.*

* The pattern is concerned with defining a mechanism for a group of objects to interact based on changes in the state of one object (the subject). The observers’ behavior is triggered by changes in the subject’s state.
* It encapsulates the behavior of the dependent objects (observers) and allows for a clean separation between the subject and its observers. This separation promotes a more modular and maintainable design.
* The pattern promotes loose coupling between the subject and its observers. The subject doesn’t need to know the concrete classes of its observers, and observers can be added or removed without affecting the subject.
* The primary mechanism in the Observer Pattern is the notification of observers when a change occurs. This notification mechanism facilitates the dynamic and coordinated behavior of multiple objects in response to changes in the subject.

Example: Let us Imagine a scenario where the **weather station** is observed by various **smart devices**. The weather station maintains a list of registered devices. When there’s a change in weather conditions, the weather station notifies all devices about the update.

* Each device, acts as a concrete observer, interprets and displays the information in its own way.
* The Observer Design Pattern facilitates a flexible and scalable system where adding new devices or weather stations doesn’t disrupt the overall communication, providing real-time and location-specific weather updates to users

**Challenges or difficulties while implementing this system without Observer Design Pattern**

* Components interested in weather updates would need direct references to the weather monitoring system, leading to tight coupling.
* Adding or removing components that react to weather changes requires modifying the core weather monitoring system code, making it hard to maintain.

**How Observer Pattern helps to solve above challenges?**

The Observer Pattern helps decouple the weather monitoring system from the components interested in weather updates. Each component can register as an observer, and when the weather changes, the observers are notified. This way, adding or removing components doesn’t affect the weather monitoring system.

**When to use the Observer Design Pattern?**

* **One-to-Many Dependence:**
  + Use the Observer pattern when there is a one-to-many relationship between objects, and changes in one object should notify multiple dependent objects.
  + This is particularly useful when changes in one object need to propagate to several other objects without making them tightly coupled.
* **Decoupling:**
  + Use the Observer pattern to achieve loose coupling between objects.
  + This allows the subject (publisher) and observers (subscribers) to interact without being aware of each other’s specific details. It promotes a flexible and maintainable system.
* **Change Propagation:**
  + When changes in the state of one object should automatically trigger updates in other objects, the Observer pattern is beneficial.
  + This helps ensure that all dependent objects are informed and can respond accordingly to changes in the subject.
* **Dynamic Composition:**
  + If you need to support dynamic composition of objects with runtime registration and deregistration of observers, the Observer pattern is suitable.
  + New observers can be added or existing ones removed without modifying the subject.
* **Event Handling:**
  + The Observer pattern is often used in event handling systems.
  + When events occur in a system, observers (listeners) can react to those events without requiring the source of the events to have explicit knowledge of the observers.

**When not to use the Observer Design Pattern?**

* **Performance Overhead:**
  + In scenarios where performance is critical and there is a concern about the overhead of managing and notifying multiple observers, the Observer pattern might not be the best choice.
  + It adds some runtime overhead due to maintaining the list of observers and notifying them.
* **Complexity for Simple Scenarios:**
  + For simple scenarios where there are only a few objects that need to be notified of changes, using the Observer pattern might introduce unnecessary complexity.
  + In such cases, a direct approach might be more straightforward.
* **Unintended Broadcasts:**
  + If there’s a risk of unintentionally broadcasting changes to a large number of observers and you need more control over which observers should be notified, consider alternative patterns that provide more fine-grained control.
* **Overuse in Small Systems:**
  + In small applications where the benefits of decoupling and dynamic composition are not crucial, using the Observer pattern might be overkill.
  + Simpler mechanisms or direct communication between objects might be more appropriate.



