When an object is modified, its dependent objects are automatically notified. The Observer pattern is a behavioral pattern.

**Intent**: Define a one-to-many dependency between objects. When the state of an object changes, all objects that depend on it are notified and automatically updated.

**The main solution**: the problem of notifying other objects of changes in the state of an object, and taking into account ease of use and low coupling to ensure a high degree of collaboration.

**When to use**: When the state of an object (target object) changes, all dependent objects (observer objects) will be notified and broadcast notification.

**How to fix**: Using object-oriented techniques, this dependency can be weakened.

**Keycode**: There is an ArrayList in the abstract class to store the observers.

**Application examples**: 1. During the auction, the auctioneer observes the highest bid and then notifies other bidders to bid. spring event event

**Advantages**: 1. The observer and the observed are coupled. 2. Establish a trigger mechanism.

**Disadvantages**: 1. If an observed object has many direct and indirect observers, it will take a lot of time to notify all observers. 2. If there is a circular dependency between the observer and the observation target, the observation target will trigger a cyclic call between them, which may cause the system to crash. 3. The observer mode has no corresponding mechanism to let the observer know how the observed target object has changed, but only know that the observed target has changed.

**scenes to be used**:

The change of one object will cause one or more other objects to change, and it is not known how many objects will change, which can reduce the coupling between objects.

**Note**: Avoid circular references.

**accomplish**

The Observer pattern uses three classes Subject, Observer and Client.