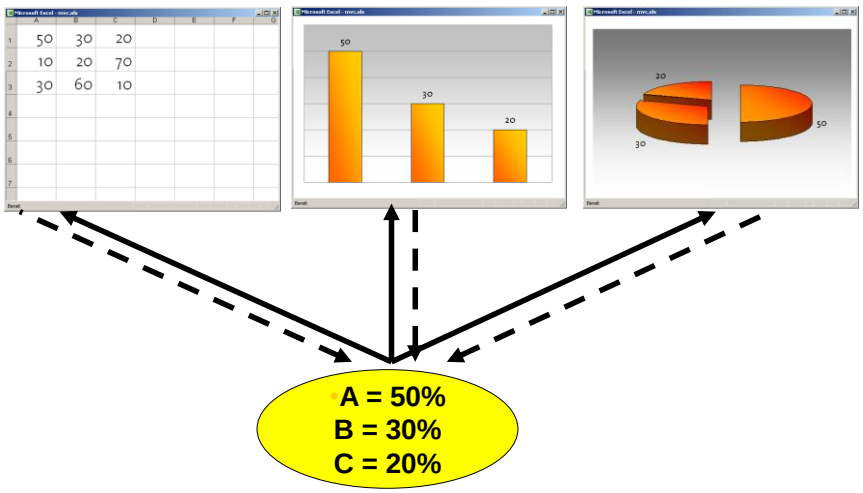
Observer

Goal

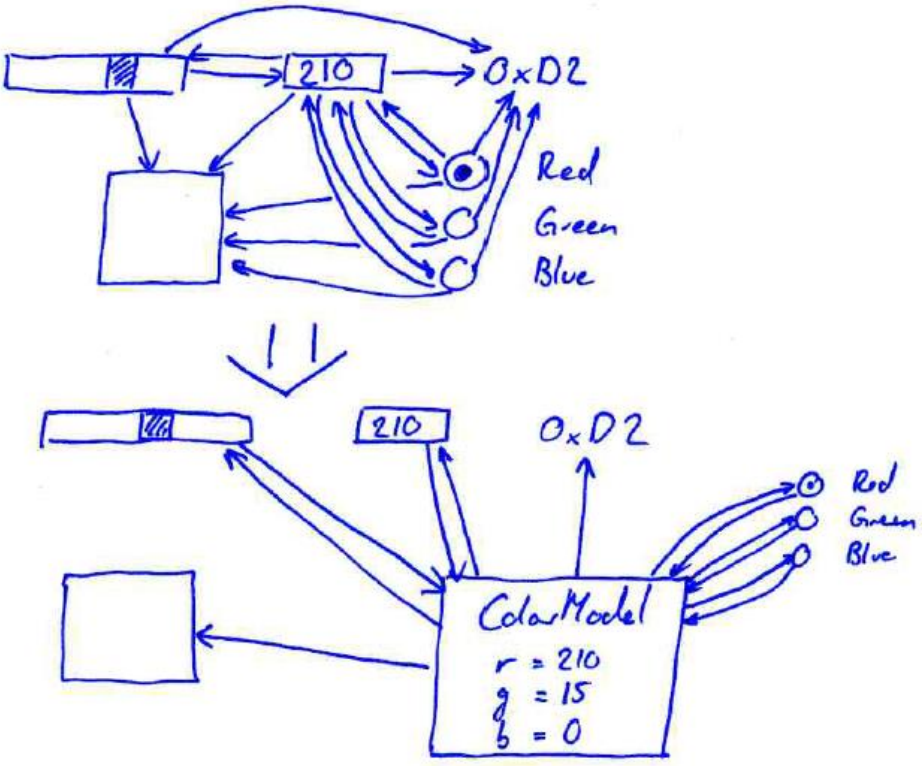
Implements a one-to-many relation between objects so that when one object changes its state, all dependent objects are informed about this state change automatically.

Motivation

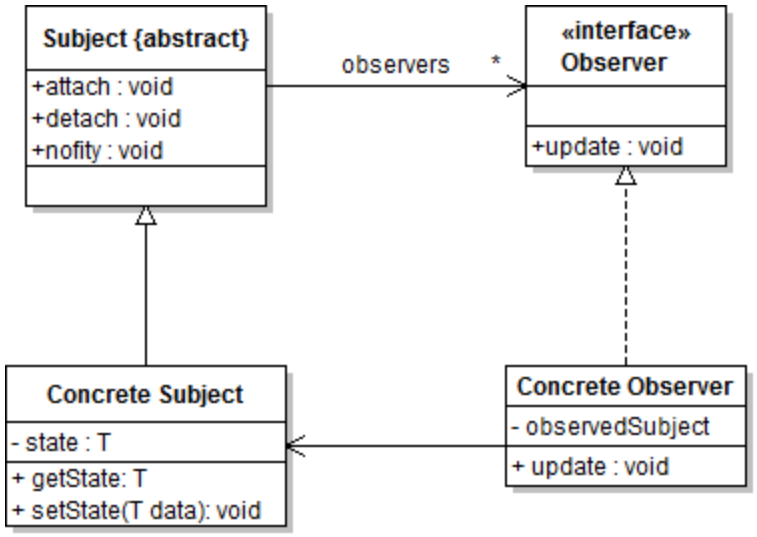
- Keeping cooperating object consistence without coupling them too much.
- Example: Separation Model – Presentation → Modell and presentation can be used independently from each other.



- Notification *without* knowing the *concrete* observer type.
- Reduction of mutual dependencies (cf. color picker assignment)



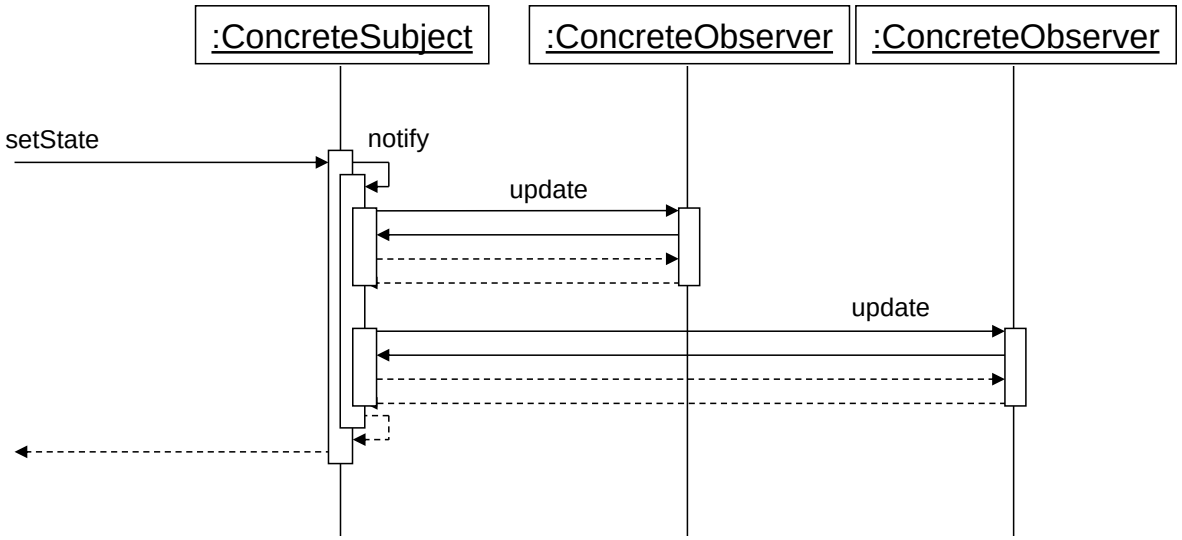
Structure



Subject:
- Knows the Observer
- Provides methods to add and remove observers (attach/detach)
- Can be declared as an abstract class

Observer:
- Defines the interface for actions which can be fired.

Sequence diagram



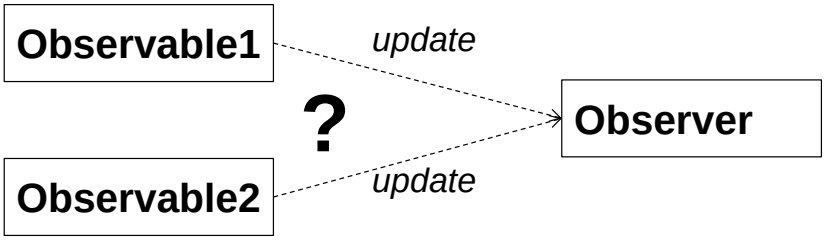
Observer: Remarks

Who has changed?

If an observer observes several subjects:

Definition
```
update(Observable s);
```

Invocation
```
update(this);
```



Application: AWT/Swing Listener which observe several buttons.

What has changed?

Example 1: In a document which is several pages long a character is inserted.
Example 2: Several thousand users have a subscription to a stock market ticker.

Push- / Pull-Modell

Push:
1. Sending the whole document text upon each keystroke to all observers (push) would be very inefficient.
2. The stock exchange price is a short information and can be published together with the notification about the state change.

Pull:
1. The observer only get the information about the text position where the change happened together with an information about the type of the change (insertion, deletion or replacement). Each observer then decides whether they present this text position and whether their view is affected by this change. If not, then the observer has nothing to do. If yes, then the observer fetches (pull) the necessary information from the subject.
2. We would have a higher network load if the observers are only informed that the stock exchange price has changed without informing the observers about the actual change, as each observer would then have to query the new stock exchange price at the subject.

Who initiates the notification that the state has changed?

Often, the notification of the registered observers is implemented in a dedicated method:

```
notifyObservers (Observable s, Object arg) {
  for(Observer o: observers) {
    o.update(s, arg);
  }
}
```

Who should invoke method `notifyObservers`?

a.	*The function which initiates the state change*	→	This typically leads to many notifications
b.	*Explicitly*	→	Someone has to think to explicitly trigger the notification

Possible solutions

- Delayed updated, e.g. asynchronous updates at idle time
- setChanged / clearChanged (Example: class java.util.Observable)
  - notifyObservers only does something if setChanged has been called
  - notifyObservers clears the changed flag (by invoking clearChanged)
- JavaFX: ChangeListener vs InvalidationListener

How does a notification look like?

- *update()* - *without parameters*

Implies the pull model. Extremely simple. The changed state has to be queried by the observer from the observable.

- *update(Observable s, Color c)* - *with the source and/or*
  - *with information about the new state*

Simple, but difficult to extend if the type of notification changes as the parameters are listed individually in the argument list.

- *update(Observable s, Object args)* - *with the source and/or*
  - *with information about the change (or the new state)*

Simple. With the argument args the state change can be described or the new state can be published. This solution is extensible (but if the actual type passed to the observer is changed, then all observers sill have to be adjusted; but they could simultaneously support the old and the new version).

.

- *update(Event e)/update(Message msg)* - *with a message parameter*

A single argument (type Event or Message) encapsulates the information needed by the observer. This could be the source and/or information about the new state or the new state itself. This is extensible.

The information object is typically called event or message. If the update method is invoked (e.g. inside method notifyListeners), then this is also called

- sending messages, or that
- an event is distributed (e.g. if a button receives the event “mouse pressed”, then all registered listeners (=observers) are informed by the button about this event.