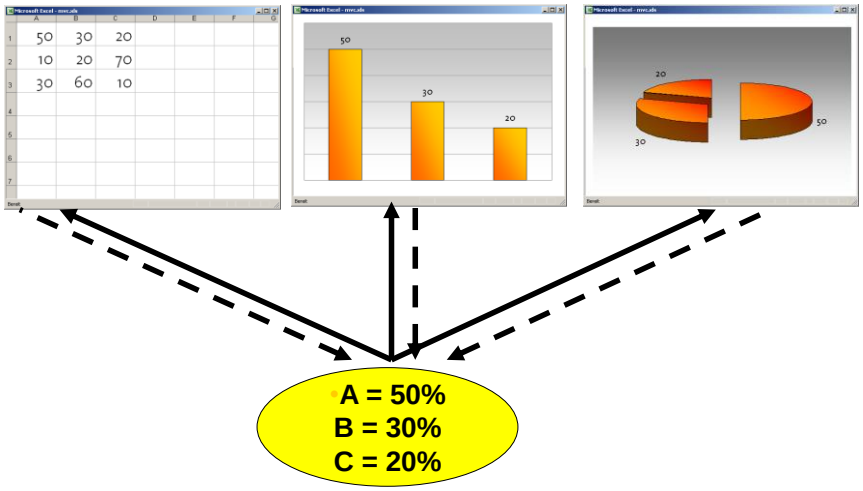
Observer

Ziel

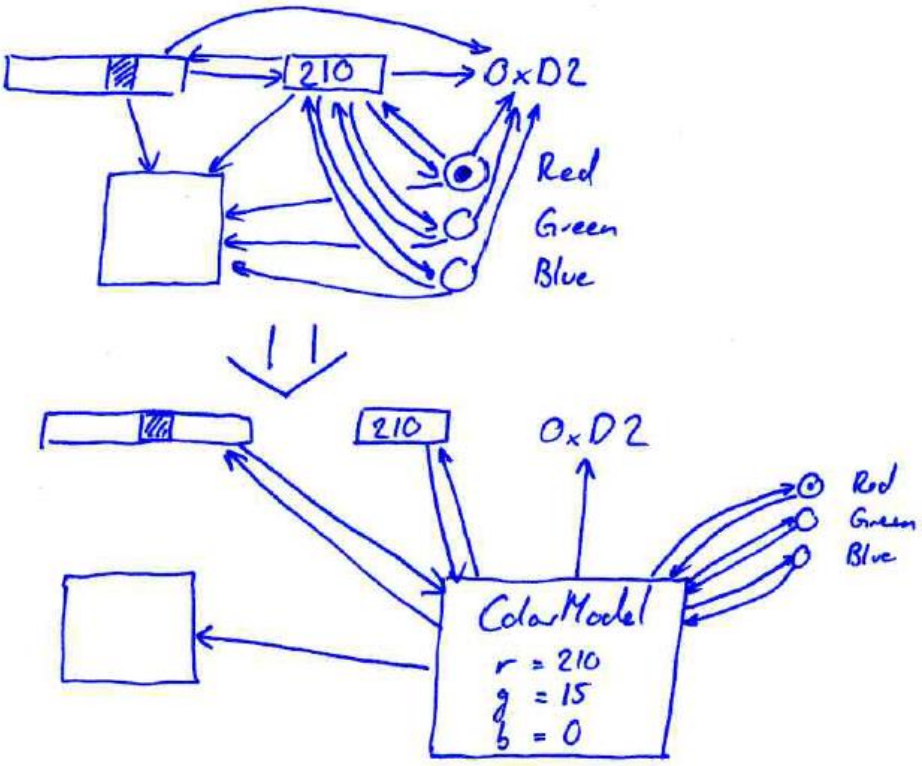
1:*-Beziehung zwischen Klassen, die es ermöglicht, die abhängigen Objekte von Zustandsänderungen zu informieren.

Motivation

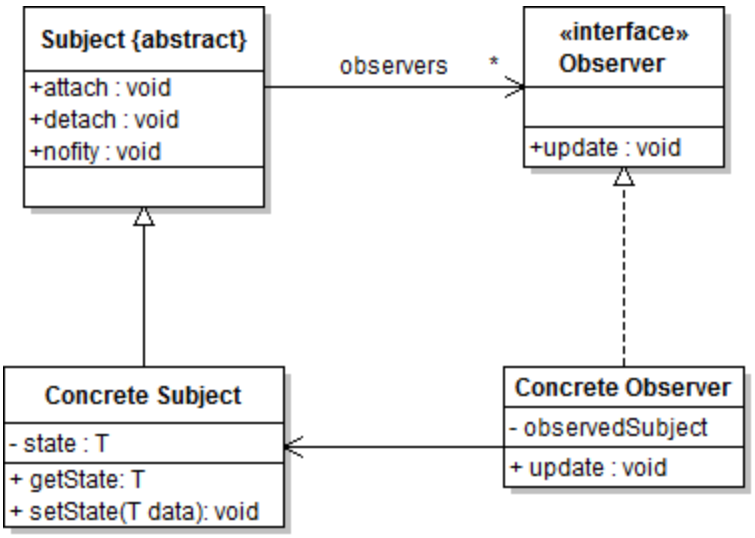
- Konsistenzsicherung zwischen kooperierenden Objekten ohne diese zu nahe aneinander zu binden.
- Beispiel: Trennung Modell – Darstellung → Modell und Darstellung können unabhängig voneinander verwendet werden.



- Notifikation *ohne* Art der *Beobachter* zu kennen.
- Reduktion der gegenseitigen Abhängigkeiten (siehe Übung Colorpicker).



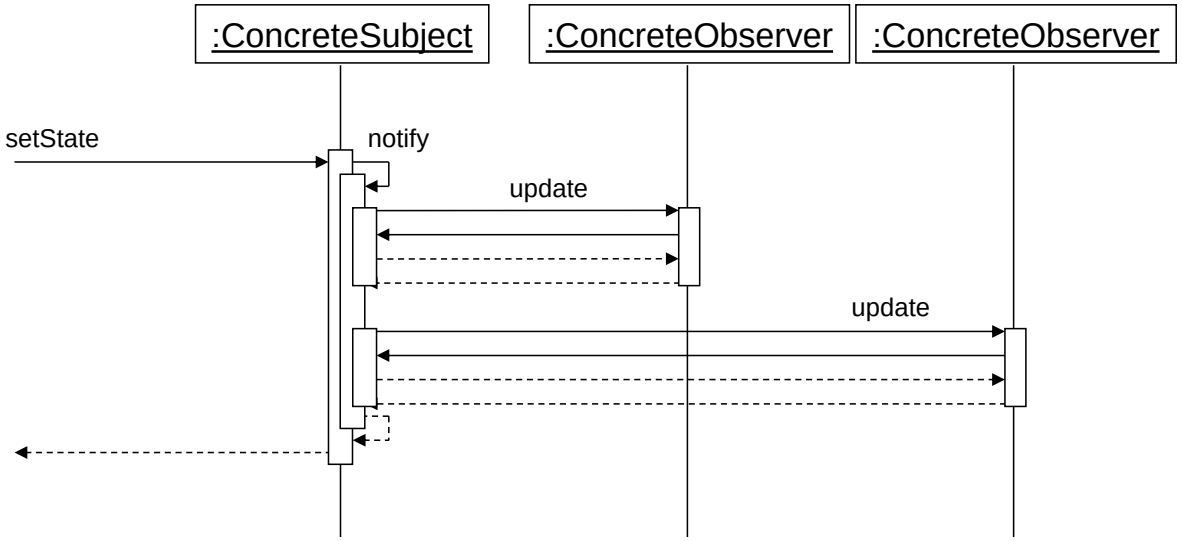
Struktur



Subject:
- kennt Observer
- bietet Methoden zum an- und abmelden (attach/detach)
- kann als abstrakte Basisklasse bereitgestellt werden

Observer:
- definiert Schnittstelle für Aktionen die ausgelöst werden können.

Sequenzdiagramm



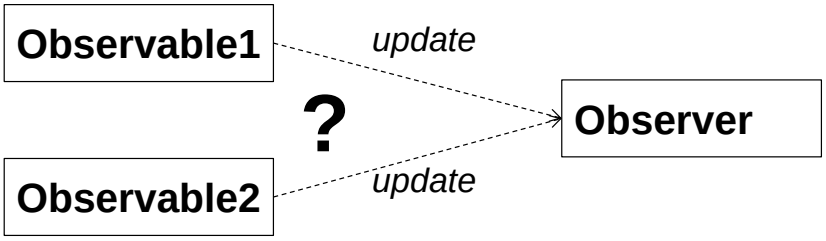
Observer: Bemerkungen

Wer hat sich geändert?

Falls ein Observer mehrere Subjekte beobachten soll:

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

Aufruf
```
update(this);
```



Anwendung: AWT/Swing Listener welcher mehrere Schaltflächen beobachtet.

Was hat sich geändert?

Beispiel 1: In einem mehrere Seiten langen Dokument wird ein einziges Zeichen neu eingefügt.
Beispiel 2: mehrere Tausend Benutzer haben einen Börsenticker abonniert.

Push- / Pull-Modell

Push:
1. Den gesamten Dokumenttext bei jedem Tastendruck an alle Observer zu übermitteln *(push)*, wäre wohl sehr ineffizient.
2. Die Informationen über die Kursänderungen sind sehr kurz und können gleich im update() verschickt werden.

Pull:
1. Es wird den Observern nur die Textstelle mitgeteilt welche sich verändert hat. Die einzelnen Observer entscheiden dann selber, ob sie die fragliche Textstelle überhaupt darstellen. Falls nicht, gibt es für den Observer auch nichts zu tun. Falls doch, dann holt *(pull)* sich der Observer vom Subjekt die nötigen Informationen.
2. Mehrfache Netzwerkbelastung falls nur mitgeteilt wird, dass sich etwas geändert hat. Denn danach muss jeder der Observer nochmals beim Subjekt nachfragen, was sich tatsächlich geändert hat.

Wer löst eine Benachrichtigung über eine Zustandsänderung aus?

Oft wird die Benachrichtigung der Observer in einer speziellen Methode ausgeführt:

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

Wer soll nun die Methode notifyObserver aufrufen?

a.	*Funktion welche Zustandsänderung bewirkt*	→	im Allgemeinen viele Updates
b.	*Explizit*	→	dann darf der Aufruf nicht vergessen werden

Mögliche Lösungen

- Verzögertes update, z.B. zu idle-time, d.h. asynchrones update.
- setChanged / clearChanged (Beispiel: Klasse java.util.Observable)
  - notifyObservers wird nur aktiv, falls zuvor setChanged aufgerufen wurde
  - notifyObservers ruft clearChanged auf.
- JavaFX: ChangeListener vs InvalidationListener

Wie soll eine Benachrichtigung aussehen?

- *update()* *- ohne Parameter*

Nur Pull-Modell. Extrem einfach. Der geänderte Zustand muss vom Observer beim Observable abgefragt werden.

- *update(Observable s, Color c)* *- mit Absender (s) und/oder*
  - Information über neuen Zustand

Einfach. Problematisch zu erweitern falls Argumente einzeln als Parameter auftreten

- *update(Observable s, Object args)* *- mit Absender (s) und/oder*
  - Information über Änderung (args)

Einfach. Über das Objekt args kann die Änderung beschrieben werden oder der neue Zustand kann übermittelt werden. Sehr gut erweiterbar.

.

- *update(Event e)/update(Message msg)* *- mit einem Meldungsparameter*

Ein einziges Objekt kapselt alle notwendigen Informationen. Im Code etwas umständlicher zu programmieren. Sehr gut erweiterbar.

Das Informationsobjekt wird häufig auch Ereignis (Event) oder Meldung (Message) genannt. Wenn nun die update-Methode (z.B. innerhalb von notifyListeners) aufgerufen wird, spricht man auch oft

- von Meldungen die verschickt werden, oder
- davon, dass ein eingetroffenes Ereignis mitgeteilt wird (z.B. wenn das Ereignis „Maus gedrückt“ auf einem Knopf eintrifft, so informiert der Knopf alle Listeners (= Observers!))