4. Patterns 2: GoF Observer

# Redegør for, hvad et software design pattern er.

Design Pattern kan beskriver et problem der opstår igen og igen og derefter beskriver kernen i en løsning til det problem. Det der gør det til et mønster er så at man kan bruge løsningen om og om igen, uden at gøre samme.

Når vi anvender designregler skal det nemlig altid tilpasses. Et design pattern er derfor aldrig en færdig løsning på et problem, men en skabelon til løsning af et problem.

# Redegør for opbygningen af GoF Observer

Problemet vi ønsker at løse er det generelle ”data-update” problem. Provider der indeholder data der ændres og en consumer der bruger data og gerne vil vide hvornår det opdateres.

Det vi ønsker er en mekanisme der:

* Tillader consumers til at blive tilføjet til providers uden at ændre provideren (OCP)
* Gør at provider kan give besked om ændring i data til consumeren.
* Gør at mange consumere kan blive informeret om opdateringer på samme data.

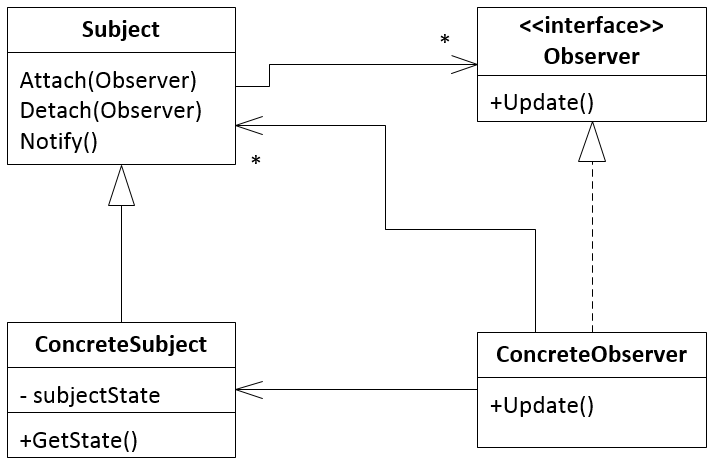
Det er GOF observer

**Opbygningen:**

Dette pattern er opbygget af subject (consumer) og Observer (provider), subjects er klasser som ændre sig, og det ønsker vi at følge med i og observes er dem der ønsker at blive opdateret når subjects ændre sig.

**Observer** er et inferface der skal implimenters af alle klasser der ønsker at informeres om updateringer

**Subject** er en abstrakt basis klasse for alt data (alt det opdateres)



Dette er den faktisk klasse der modtager opdateringer

Den faktiske klasse det bliver ovevåget.

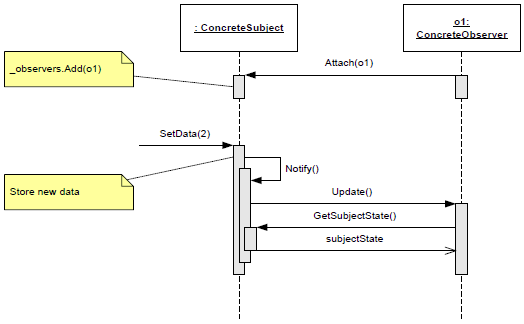
Her ses pull versionen push er næsten det samme her kender Observer bare ikke Subject.

# Sammenlign de forskellige varianter af GoF Observer – hvilken vil du anvende hvornår?

Der er to variationer: Push og Pull

## Pull

Pull bruges når der er ønskværdigt at observeren bestemmer om den vil opdateres, det er en fordel at bruge i et simpelt system. Det er også nemt at implementere subject og Observer som genbrugelige enheder

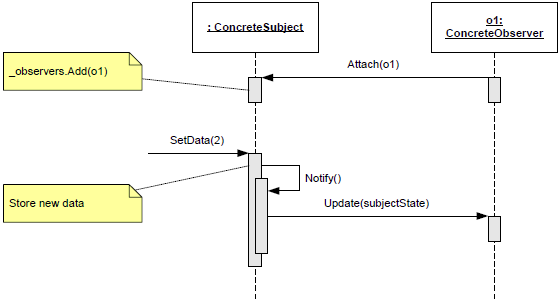


I pull varianten notify'er Subjectet(publisheren) om ændringer, hvorefter observer'erne requester staten

på subjectet, hvis observeren har behov for dette.(notify kalder alle opdates på observers)

## Push

Push er bedre at bruge hvis vi har meget data der kan ændre sig eller hvis alle observers skal have alle opdateringer. I tilfældet med meget data kan det være at kun lidt af dataen er blevet opdateret og det der denne vi ønsker at sende ud til observers.



I Push varianten skubber subjektet informationen til observeren. Dette har den fordel at der kun kræves et kald hver gang der kommer en opdatering.

# Redegør for, hvordan anvendelsen af GoF Observer fremmer godt software design

Fordelen ved at bruge observer pattern ses når vi har et system med meget kommunikation da det kan overskueligøre kommunikationen med sin genanvendelige struktur.

Observer gør det også nemt at have lagdelt software da data som vi ønsker at udstille blot kan laves observable og derved ikke behøves at kende til laget ovenover.

# Redegør for fordele og ulemper ved anvendelsen af GoF Observer

Fordele:

* Opfylder OPC da der nemt kan tilføjes nye observere
* Lav kobling da subject ikke skal kende observer

Ulemper

* Kan ende i et meget komplekst system hvis pattern bruges for meget. Her er det vigtigt at overveje sine behov og bygge systemet efter disse.

# Redegør for, hvilke(t) SOLID-princip(per) du mener anvendelsen af GoF Observer understøtter

* OPC er overholdt subjecter er lukkede for ændringer men kan udvides med observer.
* LSP overholdes også da concretesubject er en substitut for subject.
* DIP overholdes også da subject aldrig bør instantieres.