2. Observer Pattern

1. **Redegør for hvad software design pattern er**
2. **Redegør for opbygning af GoF Observer**
    a. Opbygget af Subjects og Observers.
    b. Subject giver et interface og ”Attach’er” og ”Detach’er” til Observers
        i. Subject kender til alle sine Observers interface.
    c. ConcreteSubject arbejder på sig selv og kan notify ConcreteObservers igennem Observer interface
        i. Sender et *Notify* ved ændringer
    d. Observer er et interface som mindes om at ændringer sker
3. **Sammenlign Pull og Push**
    a. Pull
        i. Benyttes når Observeren selv bestemmer om den skal opdateres – laver kald til concrete Subject.
        ii. Bruges i simple systemer
        iii. Nem implementering af Subject og Observer som genbrugelige enheder
        iv. Se disposition for eksempel på Pull
    b. Push
        i. Udnyttes bedre til systemer med meget data.
        ii. Subject sender *Notify*
        iii. Observers modtager alle ændringer øjeblikkeligt
            1. påtvungen update!
        iv. Se disposition for eksempel på Push
4. **Redegør for hvordan Observer pattern fremmer godt software design**
    a. Overskueliggør meget kommunikation
        i. Genanvendelig struktur
    b. Lagdelt data
        i. Data gøres *Observable* (behøver ikke kende til laget ovenover)
5. **Redegør for hvilke SOLID Principper det opfylder**
    a. OCP (Open-Close principle)
        i. Subject er lukket for ændringer
        ii. Kan udvides med Observers
    b. LSP (Lisskov Substitution Principle)
        i. ConcreteSubject er substitut for Subject
    c. DIP (Dependency Inversion Principle)
        i. Subject bør aldrig instantieres

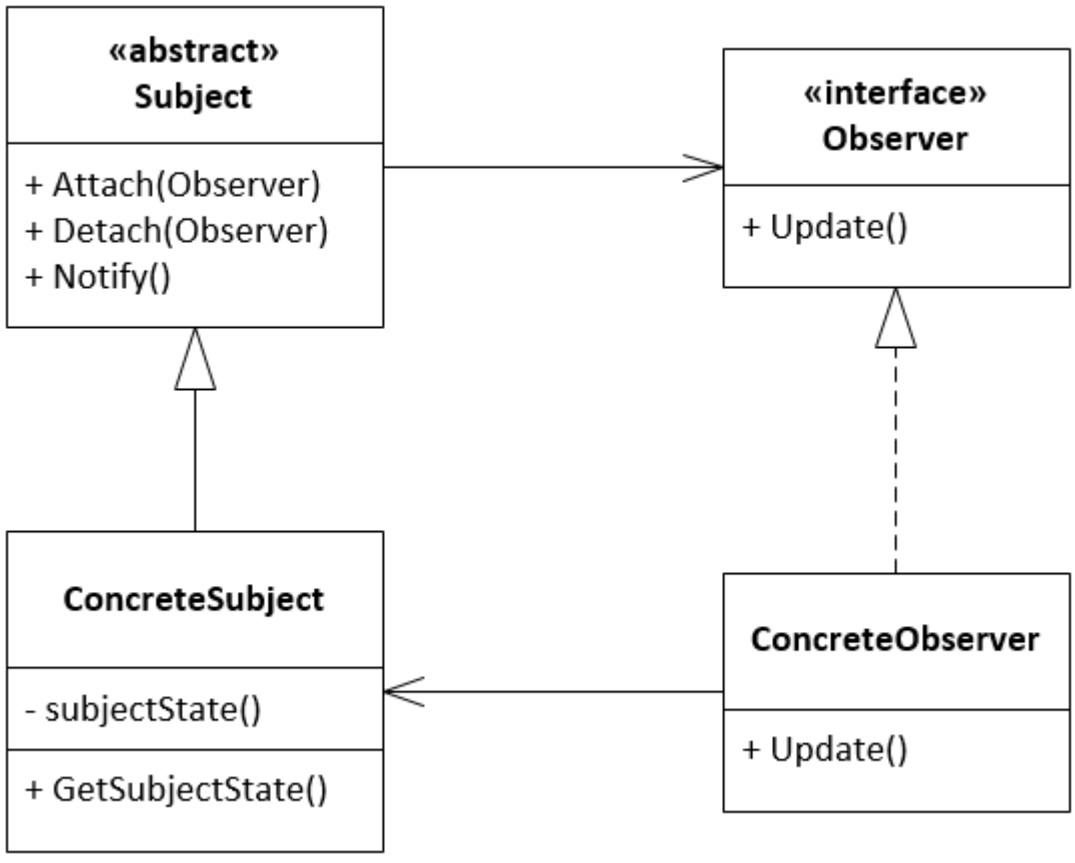
2. Observer pattern

Redegør for, hvad et software design pattern er

Et software design pattern er en **general genbrugelig** løsning til problemer, der ofte opstår i en given kontekst i software design. Det er **ikke** et færdigt design der kan laves direkte til kilde kode. Det er en beskrivelse eller skabelon for hvordan et problem kan løses i mange forskellige situationer. Det er formaliserede bedste praksisser som en programmør kan bruge til at løse problemer med.

Redegør for opbygningen af GoF Observer

Dette pattern er opbygget af **subjects** og **observers**.
"**Subjectet"** kender til alle sine observers og giver et grænseflade for at kunne "Attach" og "Detach" observers. "**ConcreteSubject"** modificerer sig selv og sender et "**Notify**" hver gang en ændring sker.
"**Observer"** har defineret et interface for objekter som skal mindes om, at en ændring er sket.
"**ConcreteObserver**" lagre samme stadie, som "**ConcreteSubject**" hvis der er tale om en **push** konfiguration.
Dette mønster benytter sig derfor af et en til mange forhold, hvor når det ene objekt skifter stadie, således kan objektes dependencies.

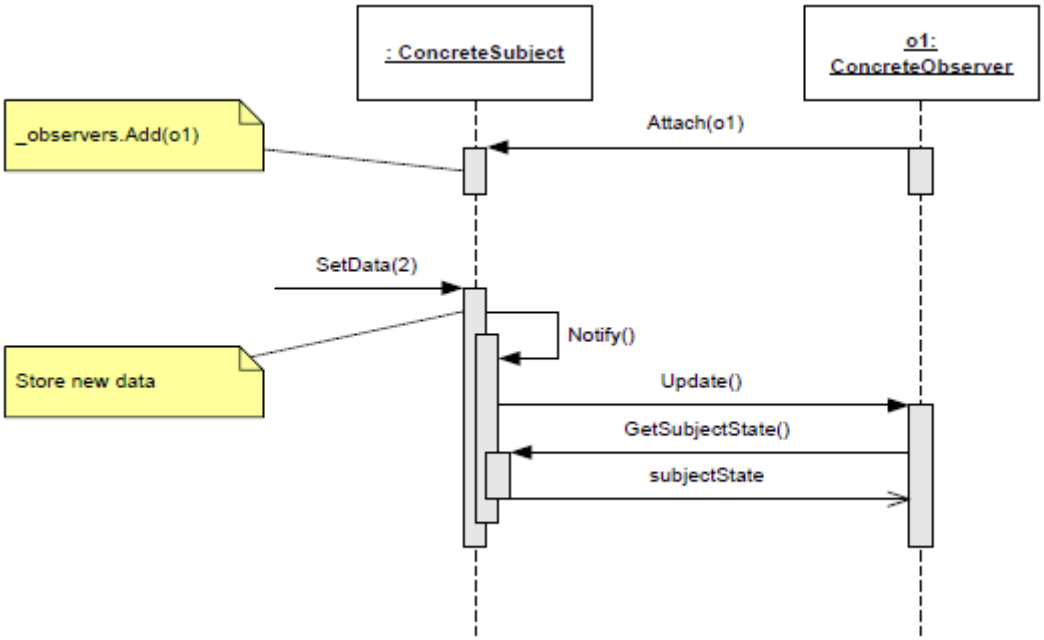


Her ses **pull** versionen af observer pattern. Push er næsten det samme. Her findes der bare ikke en update metode, eftersom, at alle observers bliver automatisk opdateret, når der sker en ændring i subject.

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

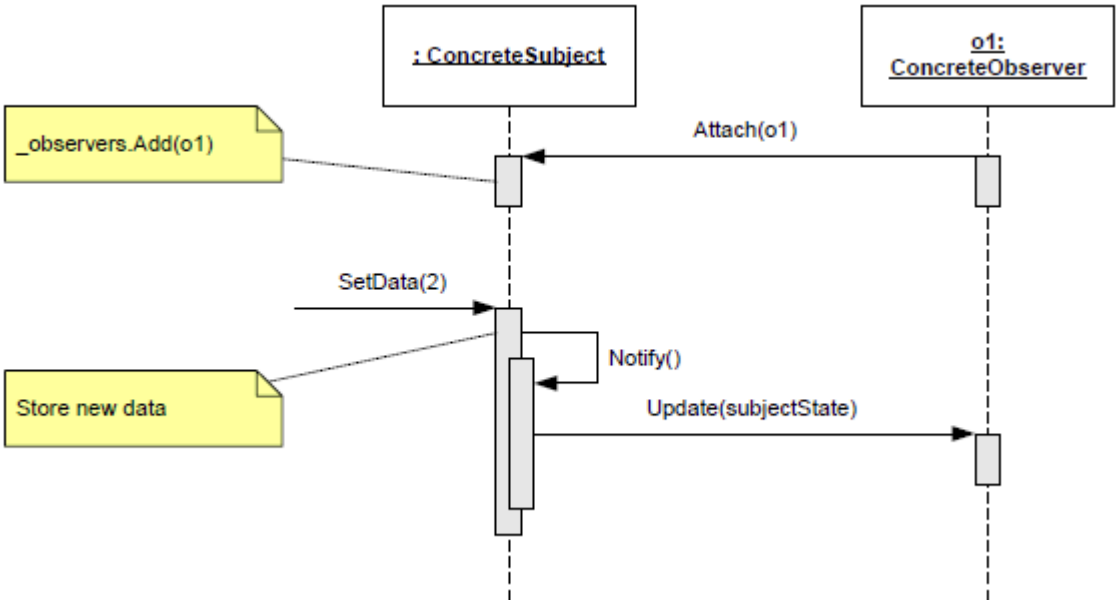
Der er to variationer push og pull.

Pull bruges når det 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.



Her ses sekvens diagrammet med et subject og en observer hvor pull er blevet brugt

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 dataene er blevet opdateret og det er denne vi ønsker at sende ud til observers.



Push varianten her ses det at observer ikke skal spørge om data men at det bliver skubbet ud til ham.

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 **OCP** da der nemt kan tilføjes nye concreteObserver
- **Lav kobling** da subject ikke skal kende concreteObserver

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

- OCP 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.

Relative pattern

Mediator (305): Byencapsulating complex update semantics, the ChangeManager acts

As mediator between subjects and observers.

Singleton (144):The ChangeManager may use the Singleton pattern to make it

uniqueand globally accessible.