6. Patterns 4: State patterns

# 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.

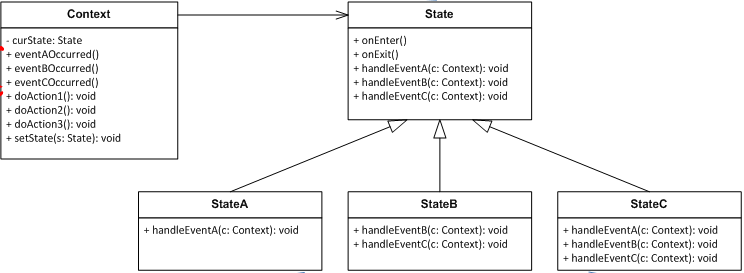
Når vi anvender designregler skal det nemlig altid tilpasses. Et design pattern er derfor aldrig en færdig løsning på et problem, men en skabelon til løsning af et problem.

# Redegør for strukturen i GoF State Pattern

State pattern er opbygget via klasser hvor der er en klasse der agere kontekst det er denne klasse vi kalder når der sker events. Dertil er der en anden klasse som agere ”Superstate” det betyder at den kan blive kaldt med alle de mulige events der er i systemet. Fra denne SuperState nedarver de individuelle states og implementer de events de har behov for. Det er ikke konteksten der sætter state det gør de individuelle states.

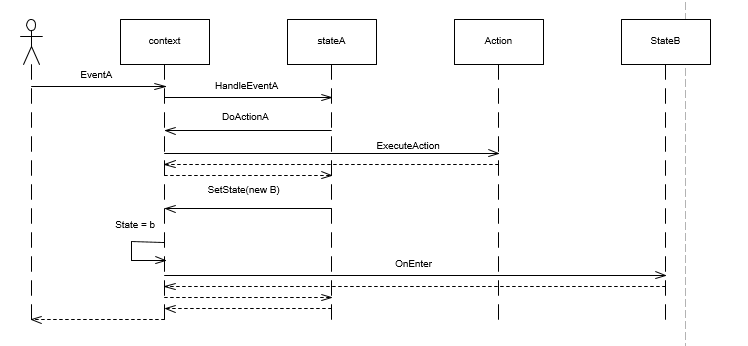
(En statemachine der ikke indeholders sine states logik)

Context er ejer af state machine. Indeholder eventhandler og actions.



c.doAction1()  
c.SetState(new StateB())

* Når Context modtager en event bliver den direkte sendt videre til dens state objekt
* Hvis en aktion skal blive udført, skal state objektet kaldes tilbage i Contexten for at udføre aktionen
* Hvis en state ændring skal udføres, vil stateobjektet kalde tilbage til contexten for at sætte contextens nye state objekt – *MEN det er ikke context der sætter den nye state, det gør currentState.*



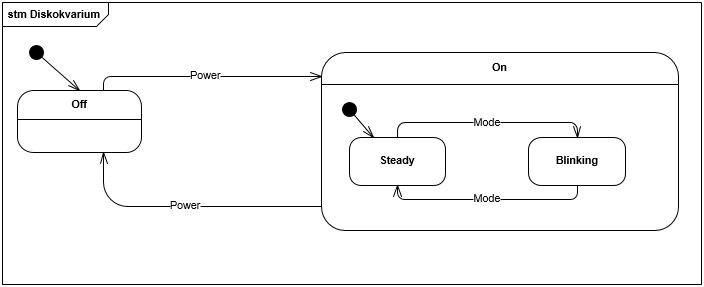
* Person kalder handle eventA på context
* Kalder onEnter så snart state er skiftet
* On exit kan ikke lade sig gøre pga. nested states

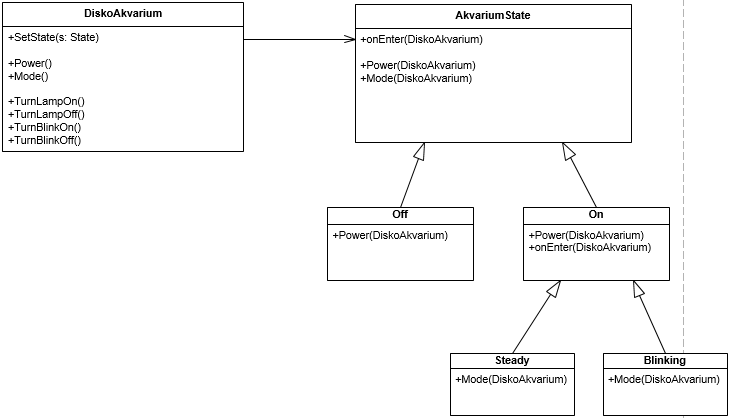
# Sammenlign switch/case-implementering med GoF State

Ved mindre state maskiner kan GoF state ikke svare sig her er det mere overskueligt med en switch case.

Men har vi for eksempel nestede states giver GoF nogle fordele.







# Redegør for fordele og ulemper ved anvendelsen af GoF State.

Fordele:

* Der opnås større overskuelighed ved store statemachines. Det er nemt at udvide
* Der er adskillelse mellem actions og state machine logikken.
* Det er fleksibelt og effektivt

Ulemper:

* Det kan være svært at denne sig et overblik over statemachinen ud fra kode.
* Der skal meget kode til!

# Redegør for, hvordan et UML (SysML) state machine diagram mapper til GoF State.

Der kan mappes fra et statediagram til GoF state så ledes:

* Alle states 🡪 Superstate
* Individuelle 🡪 SubState på superstate
* Starting point 🡪 Første state der bliver sat
* Events 🡪 De events der kan kaldes på konteksten
* Guards 🡪 Logik der skal implementeres i states
* Activity 🡪 Kald der laves tilbage på konteksten
* Tilstandskiftet 🡪 Hvilken State SetState skal kaldes med