

# Objekt-orienteret programmering uden klasser: Self.

Sammenligning klasse-baseret og  
klasseløs programstrukturering.

Basale forhold

Singulære objekter

Dynamisk nedarvning

Variable i forhold til metoder.

Metodeopslag.

Indkapsling og synlighed.

Navnerum.

**PS3 - OOP uden klasser**

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 1

## Noter

Self er beskrevet i en række artikler og dokumenter:

SELF: The Power of Simplicity

Organizing Programs without Classes.

Parents are Shared Parts of Objects: Inheritance and Encapsulation in SELF.

Alle disse findes i tidsskriftet Lisp and Symbolic Computation, 4, 3, 1991.

Endvidere kan jeg anbefale tutorial materialet

Prototype-based application construction using SELF 4.0 af Wolczko og Smith.  
som følger med distributionen af Self fra Sun's FTP site.

## Klasser og objekter i et klasse-baseret sprog.

Ethvert *objekt* er en instans af en *klasse*.

En klasse beskriver egenskaberne af et objekt.

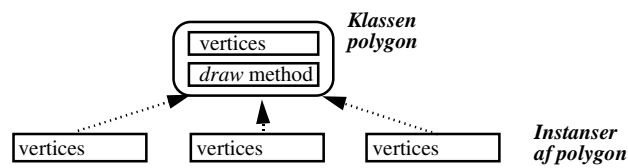
Alle egenskaber, som objekter af samme klasse er fælles om, kan repræsenteres i klassen.

To vigtige relationer:

Relationen mellem et objekt og den klasse hvorfra objektet er en instans.

Relationen mellem en klasse og dens superklasse. ←.....

*Instantiering* er mekanismen hvorved objekter skabes ud fra klassen. ←



### PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 2

## Noter

På ovenstående slide opsummerer vi forholdet mellem klasser og objekter i 'normale', klasse-baserede sprog.

Der er stor begrebsmæssig forskel mellem en klasse og et objekt, også selv om vi har set eksempler på OOP, hvor klasser er objekter.

## Objekter i et klasseløst sprog: Self.

Ethvert *objekt* beskriver sit eget format.

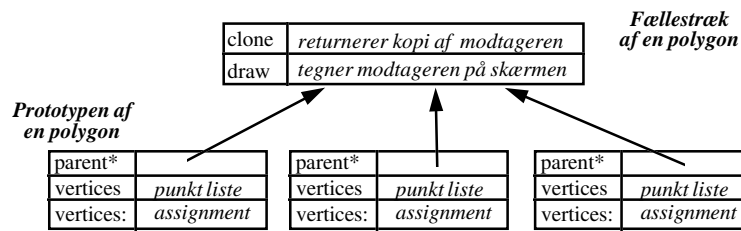
Nedarvning mellem objekter: En mængde af objekter kan udfaktorisere fælles egenskaber til et nyt objekt, som de alle *arver* fra.

Én vigtig relation:

Relationen mellem et objekt som *arver* fra et andet objekt. ←

Objekter skabes ved at *klone* (shallow copy) et *prototype objekt*.

Objekter kan *delegere* beskeder til andre objekter.



### PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 3

## Noter

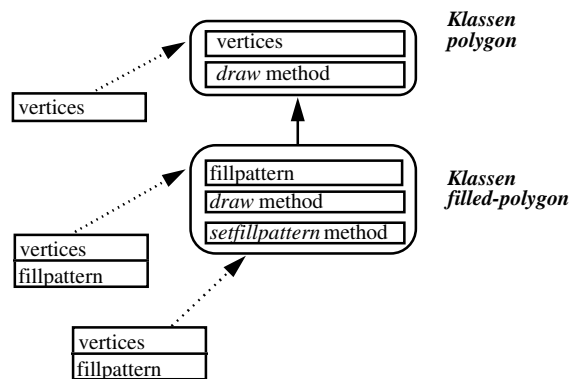
Denne slide viser samme situation som på forrige slide. Bemærk fællestræk objektet (på engelsk kaldet et trait objekt) som alle polygoner arver fra.

På denne slide er alt dog objekter. Der er altså ingen klasser involveret.

## Differentialprogrammering i et klasse-baseret sprog.

Ved nedarvning udvides en klasse med nye egenskaber.

Instanser af den specialiserede klasse er tilsvarende udvidet i forhold til instanser af superklassen.



PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

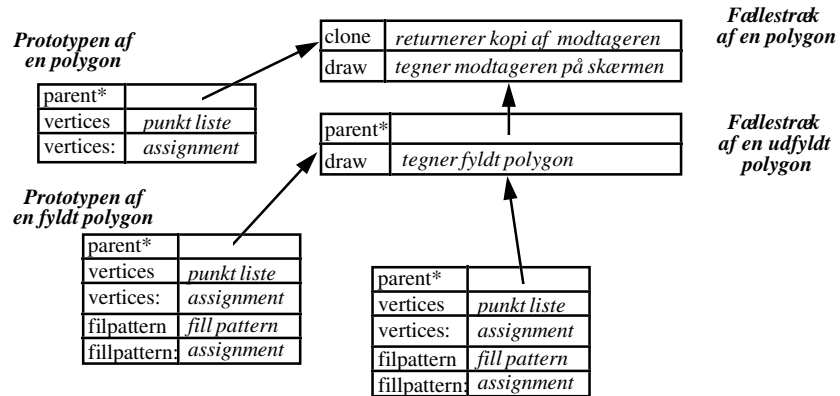
11/14/96 s. 4

## Noter

Vi introducerer nu nedarvning. Ovenfor viser vi en klasse `filled-polygon`, som arver fra `polygon`. Endvidere vises tre objekter af disse klasser. På næste slide ser vi på den samme situation i Self.

## Differentialprogrammering i et klasseløst sprog: Self.

Udvidelse af en klasse kan umiddelbart overføres til et nyt *fællestræk objekt*, som arver fra det oprindelige fællestræk objekt.



PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 5

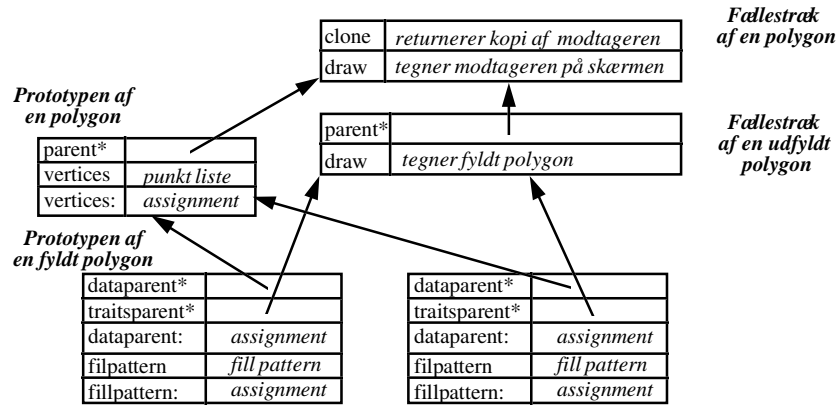
## Noter

Ligesom på forrige slides vises er tre polygoner (to udfyldte og én ordinær). Det vigtige er imidlertid de to fællestrækobjekter, som repræsenterer de fælles egenskaber ved hhv. fyldte og ordinære polygoner.

På næste slide vil vi råde bod på gentagelserne i prototypen af en fyldt polygon (vertices og vertices:) i forhold til prototypen af en polygon.

## Repræsentationsudvidelse i Self.

Udvidelsen kan også udstrækkes til repræsentationen.  
 Dette kræver understøttelse af multipel nedarvning mellem objekter.



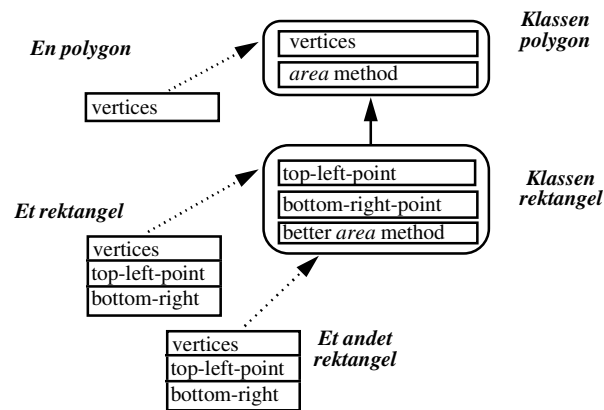
### PS3 - OOP uden klasser

## Noter

For at undgå at gentage egenskaber i prototypen af en fyldt polygon i forhold til prototypen af en polygon lader vi en fyldt polygon arve fra en polygon. Da vi også har brug for at arve fra fællestræk objektet introducerer vi her et udtalt behov for multipel objekt nedarvning.

## Repræsentationsproblemer i klasse-baserede sprog.

Specialisering medfører undertiden en *uønsket repræsentationsudvidelser*.  
Eksempelvis *overrepræsenteres* rektangler herunder med både en liste af hjørner og to diagonalt modsatte punkter.



PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 7

## Noter

Denne slide diskuterer klassehierarkier, hvor specialiserede klasser ikke har behov for at udvide repræsentationen i forhold til superklassen. Derimod har man typisk behov for at lægge bånd på superklassens repræsentation i forhold til subclasses. I eksemplet skal vi sikre os at et rektangel er en firkant med parvise parallelle sider, og med rette vinkler. Med andre strammer vi klasseinvarianten i subclasses i forhold til superklassen.

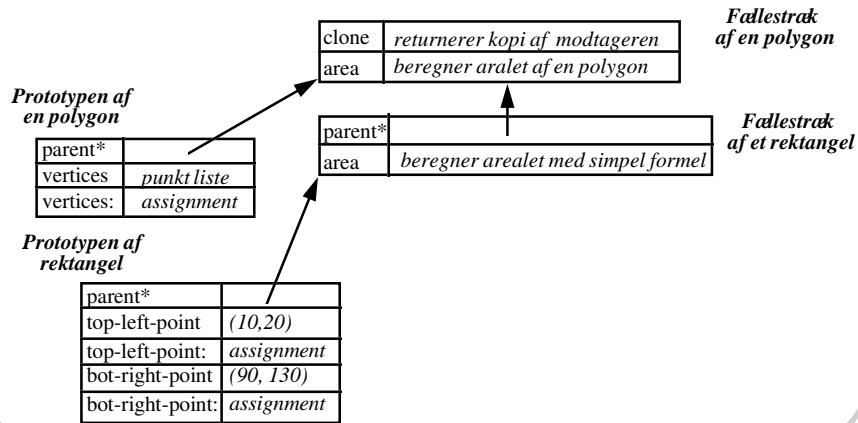
Hvis man studerer det naturlige klassehierarki af trekanter vil man observere præcist det samme fænomen.

I nedrivning i klassebaserede sprog kan man kun undgå overrepræsentere objekter ved at lade alle superklasse være abstrakte, uden repræsentation. Dette er imidlertid ikke tilfredsstillende i trekanthierarkiet, idet det giver fin mening at instantiere en generel trekant.

Så der er et reelt problem her i klasse-baserede sprog.

## Løsning af repræsentationsproblemet i klasseløse sprog: Self.

Ved at undlade at arve fra prototypen af en polygon kan vi let arrangere, at et rektangel ikke overrepræsenteres.



### PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

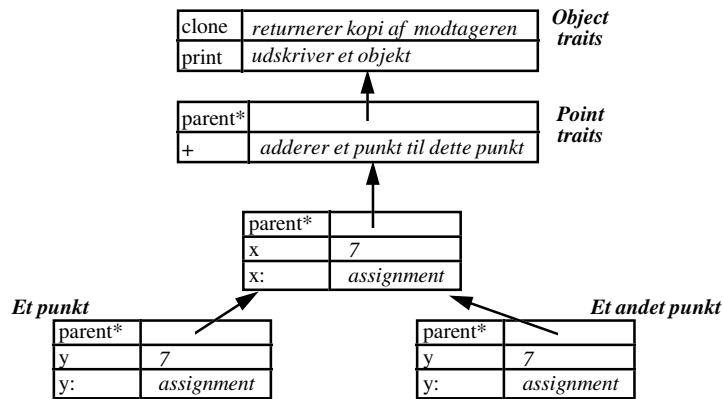
11/14/96 s. 8

## Noter

Problemet fra forrige slide løses elegant i Self ved at rektangel prototypen undlader at arve fra polygon prototypen. Bemærk dog, at fællestrækobjektet for rektangel arver fra polygon fællestrækobjektet.

## Fleksibel 'sharing' giver nye muligheder i Self.

Ved at manipulere 'parent' kan det eksempelvis lade sig gøre at lave en række punkt-objekter med delt x-koordinat.



### PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 9

## Noter

Denne slide viser, hvordan vi ved at jonglere med parent referencerne kan organisere objekterne på nye spændende måder.

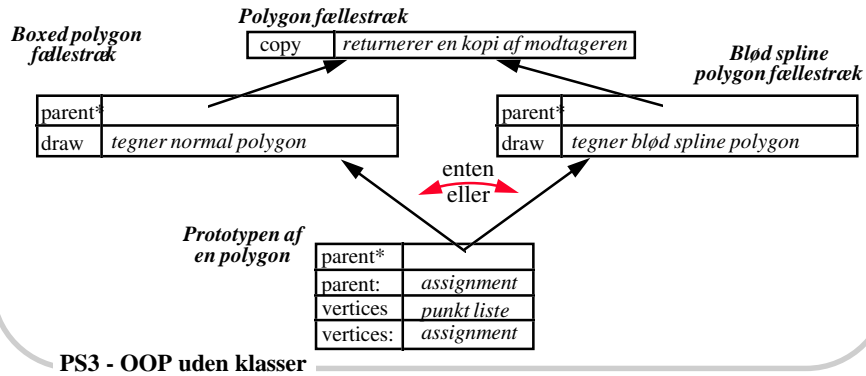
I Self arbejder vi reelt med *objekt-dele* i stedet for en *objekt-helhed*. Ved at mutere referencerne mellem objekt-dele opstår der helt nye muligheder i forhold til det sædvanlige objekt-orienterede paradigme baseret på klasser.

## Dynamisk nedarvning.

Ved dynamisk nedarvning menes muligheden for at ændre på 'parents' under kørslen af et program.

Med dynamisk nedarvning muliggøres midlertidige forandringer i objekters opførsel.

Dynamisk nedarvning udgør en finkortnet mekanisme til ændring af 'klassen af et objekt'.



PS3 - OOP uden klasser

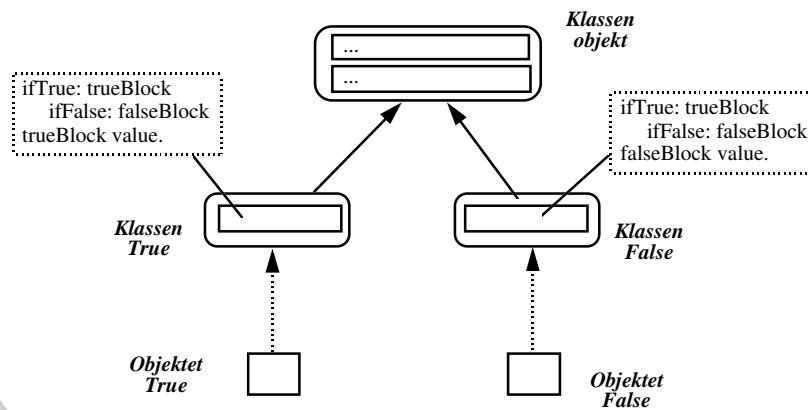
## Noter

Denne slide fortsætter i samme spor som foregående. Vi viser hvordan man ved at ændre på en parent reference fra ét fællestræk til et andet fællestræk kan lave en (måske midlertidig) forandring af et objekts opførsel.

## Singulære objekter i klasse-baserede sprog.

Det er akavet at arbejde med singulære objekter i de fleste klasse-baserede sprog.

Det er svært at sikre, at der ikke bliver lavet flere instanser af en klasse.



PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 11

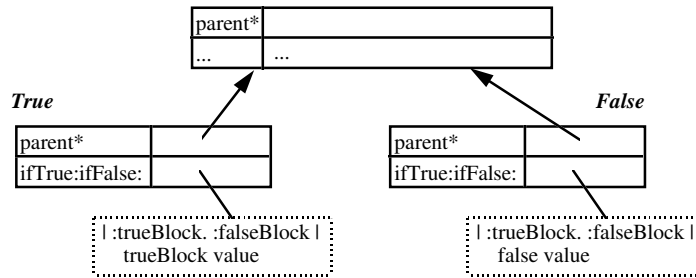
## Noter

Denne slide viser hvordan man håndterer singulære objekter i et Smalltalk-agtigt system.

Næste slide viser den tilsvarende situation i Self.

## Singulære objekter i klasseløse sprog: Self.

Det er umiddelbart naturligt at arbejde med singulære objekter i et klasseløst sprog.



### PS3 - OOP uden klasser

## Noter

Empty dashed box for notes.

## Self: Et sprog “uden variable”.

I Self tilgås slots (variable) ved at sende en besked til objektet selv (self).

Variable er intetsteds leksikalsk tilgængelige.

På det sted, hvor man refererer til en variabel kan man ikke erkende om der er tale om

- en aflæsning af variabelens værdi
- en aktivering af en metode, som refereres fra den slot, som repræsenterer variabelen.

Det er ikke nødvendigt eksplicit at angive et modtagerobjekt for en besked.

- En ‘implicit receiver’ besked sendes til self.
  - Fremmer kortfattet udtryksform.
  - Har en semantisk konsekvens, idet method lookup starter ved ‘current activation’ i stedet for ‘current receiver’.

### PS3 - OOP uden klasser

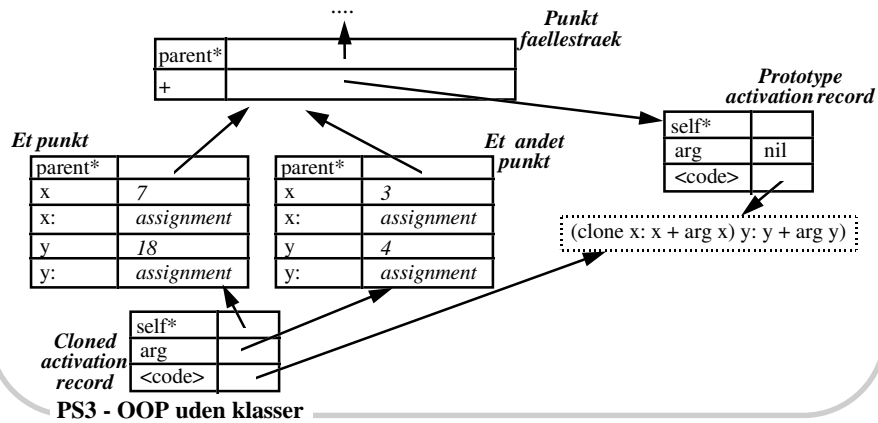
## Noter

## Aktivering af metoder i Self.

I funktionsorienteret programmering indså vi at

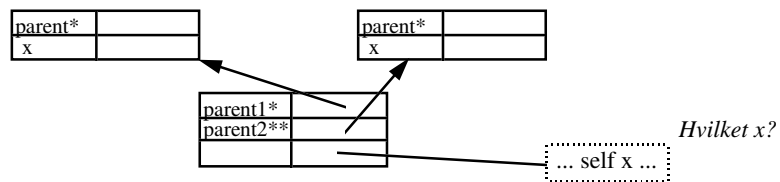
- closures tangerer klassebegrebet
- funktionskald tangerer objektbegrebet

I self unificeres metodeaktivering og objekter.



## Noter

## Principper for 'method lookup' i Self objekter (1).



I Self er det muligt at *prioritere* en 'parent' frem for en anden i forbindelse med metode opslag.

Dette er et alternativ til CLOS' totale ordning af alle superklasser.

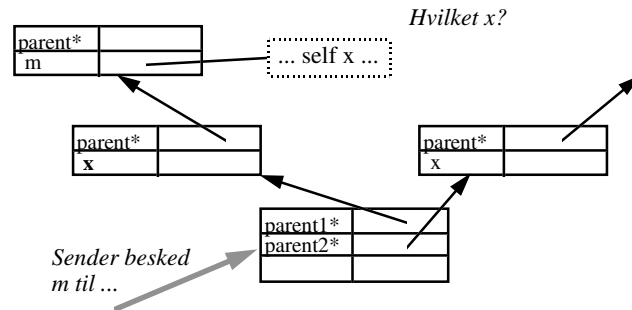
PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 15

Noter

## Principper for 'method lookup' i Self objekter (2).



Self har en 'sender path tiebreaker rule' som prioriterer slots på stien mellem modtageren og den metode, som ovenfor sender x til self. Ifølge denne regel vælges x i den venstre gren.

### PS3 - OOP uden klasser

## Noter

## Indkapsling og synlighed i Self.

Det muligt i Self at skelne mellem

- private slots, der kun kan refereres af metoder i klassen selv.
- offentlige slots, der kan refereres fra andre objekters metoder.

To former for privatliv:

- **Objekt-baseret indkapsling:** En metode kan kun tilgå instansvariable i self.
  - Eiffel
- **Klasse-baseret indkapsling:** En metode kan tilgår instansvariable i alle instanser af en bestemt klasse.
  - C++

*Kan man understøtte klasse-baseret indkapsling i et klasseløst sprog?*

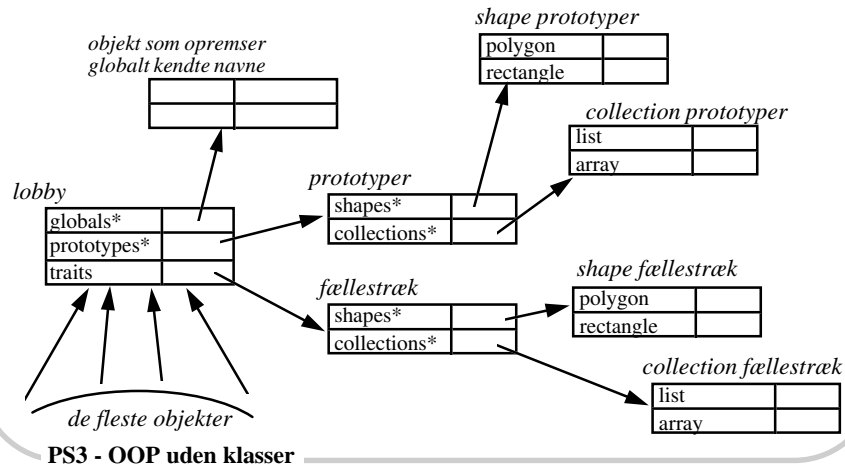
- Self forsøger dette ved at lade en metode have adgang til private slots i alle de objekter, som er fælles om metoden.

### PS3 - OOP uden klasser

## Noter

## Organisering af navnerum i Self.

Objekter kan bruges til at organisere og kategorisere navnerum i Self.  
Lobby bør opfattes som rod i objekt-hierarkiet.



PS3 - OOP uden klasser

© Kurt Nørmark, Aalborg Universitet

11/14/96 s. 18

## Noter

Det er værd at bemærke at traits ikke har en stjerne i lobby. Det betyder, at man ikke følger denne reference under metodeopslag. Man må altså sige noget i retning af

traits rectangle

for at referere til rectangles fællestræk fra et objekt. Hvis man blot siger

rectangle

vil man referere til rectangel prototypen.

Kategorisering af navne er et vigtigt emne i Smalltalk. Sproget Smalltalk er blind for dette aspekt, men emnet håndteres i programmeringsomgivelsen.

Emnet diskuteres i afsnit 4 af artiklen 'Organizing Programs Without Classes'.