7
Introduktion til Smalltalk.

Designbeslutninger.
Message passing.
Klassebegrebet.
Metodebegrebet.
Forskellige slags variable.
Nedarvning og metodeopslag.
Instans og klassevariable ift. nedarvning.
Blokbegrebet.
Blokke og kontrolstrukturer.

Noter

Nogle eksempler og slides i dette kapitel er lånt eller inspireret fra slides, som Kasper Østerbye har benyttet i tidligere versioner af dette kursus.

Et dynamisk, objekt-orienteret sprog.

- Smalltalk = Simula - Algol + Lisp.
  - Har overtaget, og videreudviklet de objekt-orienterede ideer fra Simula.
  - Populariserede de objekt-orienterede ideer dramatisk sidst i 70-erne og først i 80-erne.
  - Understøtter dynamisk typing, på samme måde som Lisp og CLOS.
  - Har ligesom flere Lisp-systemer en særdeles veludviklet programmeringsomgivelse omkring sig.

Noter

Grundlæggende designbeslutninger.

***Good design**: A system should be built with a minimum set of unchangeble parts; those parts should be as general as possible; and all parts should be held in a general framework.*

***Operating System**: An operating system is a collection of things that don´t fit into a language. There shouldn't be one.*

***Messages**: Computing should be viewed as an intrinsic capability of objects that can be uniformly invoked by sending messages.*

***Personal mastery**: If a system is to serve the creative spirit, it must be entirely comprehensible to a single individual.*

***Uniform Metaphor**: A language should be designed around a powerful metaphor that can be uniformly applied in all areas.*

***Storage Management**: To be truely object-oriented, a computer system must provide automatic storage management.*

***Polymorphism**: A program should specify only the behavior of objects, not their representation.*

***Classification**: A language must provide means for classifying similar objects, and for adding new classes of objects on equal footing with the kernel classes.*

***Factoring**: Each independent component in a system should appear in only one place.*

***Natural Selection**: Languages and systems that are of sound design will persist, to be supplanted only by better ones.*

PS4 - Smalltalk

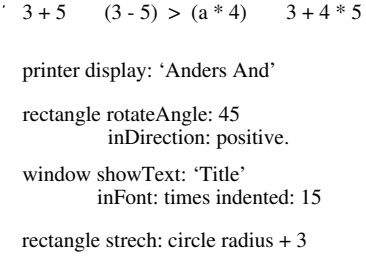
Noter

I det ovenstående har jeg i spredt orden fastholdt nogle udsagn om Smalltalk design principper, som stammer fra en artikel skrevet af Daniel Ingals ("*Design Principles Behind Smalltalk*", BYTE, August 81).

Polymorfi: Synspunktet er, at et program ikke skal erklære variable og parametre statisk af en bestemt type. Det er mere fleksibelt at "erklære", at det kan svare på en nærmere bestemt mængde af beskeder.

‘Message passing’ metaforen.

- Objekter interagerer med hinanden, udelukkende ved at sende beskeder.
- En besked lokaliserer (ved dynamisk opslag) en metode.
- En metode returnerer altid en værdi til afsenderen.
- Forskellige former af beskeder:
  - Unær besked
  - Binær besked
  - Nøgleordsbesked



5 fak 7 fak negate

3 + 5 (3 - 5) > (a * 4) 3 + 4 * 5

printer display: ‘Anders And’

rectangle rotateAngle: 45
inDirection: positive.

window showText: ‘Title’
inFont: times indented: 15

rectangle strech: circle radius + 3

- Afsendingsregler:
  - Unære beskeder sendes først, og associerer fra venstre.
  - Binære beskeder sendes dernæst, og associerer fra venstre. Ingen indbyrdes prioriteringer.
  - Nøgleordsbeskeder sendes tilsidst. Paranteser nødvendige for kombination med andre nøgleordsbeskeder.
  - Paranteser kan ændre den indbyrdes ordning af unære, binære og nøgleordsbeskeder.

Noter

Beskeden

rectangle strech: circle radius + 3

er interessant hvad angår “parsning”. Først sendes den unære besked radius til det objekt, der refereres af circle. Dernæst sendes beskeden + til resultat af circle radius. Slutteligt sendes nøgleordsbeskeden strech: til rektanglet, som refereres af rectangle, med parameteren, som er resultatet af additionen.

Nøgleordbeskeder er interessante. Der er ligheder til nøgleordsparametre i andre sprog, men også forskelle. Konkatenationen af nøgleordene kan opfattes som metodens navn. Det er ikke muligt (som i Common Lisp) at benytte nøgleordene i vilkårlig rækkefølge i et “kald”. Eksempelvis er showtext:indented:inFont en anderledes besked end showText:inFont:indented.

Klassebegrebet.

- En klasse karakteriseres primært af
  - Én superklasse.
  - En mængde af instansvariable.
  - En mængde af klassevariable.
  - En uformel kategori, hvortil klassen hører.
- Information hiding
  - Alle instansvariable er private i klassen.
  - Alle metoder, som tilføjes klassen, er tilgængelige for alle klienter.
  - Pr. konvention kan der være en kategori af metoder, der er private.

Klassedefinition via nøgleords**besked** til eksisterende superklasse.

superclass **subclass:** #Class
instanceVariableNames: 'v1 v2'
classVariableNames :''
poolDictionaries: ''
category: 'class category'

Noter

Ovenstående viser klasseanatomien på to forskellige måder:

- Som en prosabeskrivelse og
- som en besked til superklassen af den nye klasse

Vi vil senere vende tilbage til hvordan klasser (og metoder) kan tilgåes via værktøj i programmeringsomgivelsen.

Metodebegrebet.

- En metode består af:
  - Et 'message pattern' (~ metodenavn og parameterprofil).
  - En kommentar: "*kommentar*"
  - En sekvens af udtryk: afsendelse af beskeder.
    - Mulighed for *assignment* (med assignmentprefix).
    - Mulighed for angivelse af *returværdi*.
      - Default: modtagerobjektet af den besked, hvis metode vi udfører.
    - Mulighed for *kaskade* af beskeder til samme objekt.
- Metoder specialiserer kun på én "parameter".
  - En metode tilhører netop én klasse.
  - En metode aktiveres ved at sende en besked
    - til en instans af denne klasse (eller til en instans af en subklasse)
    - med en besked der modsvarer metodens selektor.
- Der anvendes imperativ metodekombination (mellem metoder med samme selektor).

Noter

Eksempler på klasse og metoder.

class name Complex
superclass name Object
instance variable names real imag

class methods
instance creation

newWithReal: rl andImaginary: im
"Return a new complex number, corresponding to *rl* + i *im*."
^Complex **new realPart:** rl; **imaginaryPart:** im

instance methods
accessing

realPart
"return the real part of a complex number"
^real

realPart: rl
"assign a new real part of this complex number"
real := rl

arithmetic

+ anotherComplex
"add anotherComplex to this complex number"
| realSum imagSum |
realSum := real + anotherComplex **realPart**.
imagSum := imag + anotherComplex **imaginaryPart**.
^Complex **newWithReal:** realSum **andImaginaryPart:** imagSum

PS4 - Smalltalk

Noter

Ovenstående viser et skematisk eksempel på en (ukomplet) klasse i Smalltalk samt et antal metoder i denne klasse. Klassen er Complex (komplekse tal).

Man ser at der er to hovedafdelinger af metoder: klasse metoder og instansmetoder. Klassemetoderne anvendes primært til at instantiere et objekt. Dette har vi meget mere at sige om i næste kapitel (Metaklasser i Smalltalk). I dette kapitel vil vi koncentrere os om instansmetoderne (her realPart, realPart: og +).

Man ser endvidere at der er underafdelinger af instans metoder, her *accessing* og *arithemtic*. Dette er en uformel opdelning af metoderne i kategorier, som afspejles i bl.a. systembrowseren (se tidligere i dette kapitel). At opdelningen er uformel betyder, at kategorierne ikke har nogen semantisk konsekvens for et Smalltalk program.

Bemærk at der ovenfor benyttes en skematisk opstilling af klassen, dens instansvariable, og dens metoder. Der findes ikke en syntaktisk form, som tillader os at udtrykke alt dette. I praksis benytter vi et browserværktøj til at definere de enkelte bestanddele. Lidt mindre bekvemt kunne vi også definere det hele ved at sende beskeder til klasser.

Variable.

- Lokale variable i metoder.
  - De formelle parametre er ikke lokale variable: De er 'read-only'.
  - Med lille begyndelsesbogstav.
- Instansvariable i objekter.
  - Navngivne eller indicerede.
  - Navngivne instansvariable kun leksikalsk tilgængelige i metoder.
  - Med lille begyndelsesbogstav.
- Globale variable.
  - Hvert klassenavn bliver automatisk en global variabel, der refererer klassen, som objekt.
  - Med stort begyndelsesbogstav.
- Klassevariable
  - Med stort begyndelsesbogstav.
- Pool variable.
  - Default puljer tilgængelige for alle klasser:
    - Smalltalk (pulje delt af alle klasser): de globale variable.
    - Klassevariablene (klassespecifik pulje).
    - Andre puljer kan skabes efter behov.
  - Med stort begyndelsesbogstav.

Noter

Hvorvidt et objekt har indicerede instansvariable eller ej styres af den besked, som sendes til superklassen. Når beskeden

variableSubclass:instanceVariableNames:classVariableNames:poolDictioraries:category:

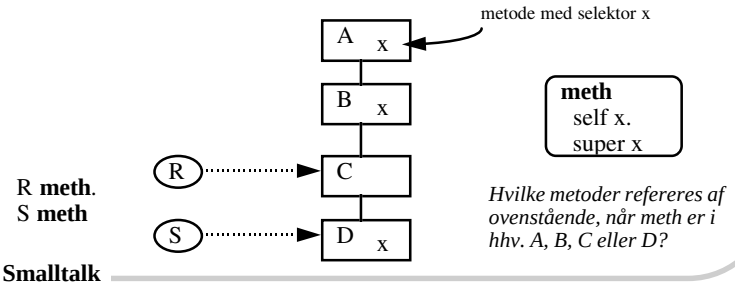
imodsætning til

subclass:instanceVariableNames:classVariableNames:poolDictioraries:category:

sendes til en klasse skabes en ny klasse, hvori der er indicerede instansvariable.

Nedarvning og Metodeopslag.

- Smalltalk tillader udelukkende enkelt-nedarvning.
- Self og super er *pseudo-variable*, der kan anvendes i metoder.
  - Self refererer til modtager-objektet af en besked, uanset den aktiverede metodes klasse-tilhørsforhold.
  - Super referer ligesom self til modtager-objektet af en besked. Men beskeder sendt til super starter metodeopslaget i *superklassen til den metode, hvori super forekommer*.
  - Super tillader imperativ metodekombination mellem metoder med samme selektor.



PS4 - Smalltalk

Noter

Fortolkningen af super må ikke forveksles med

> beskeder sendt til super starter metodeopslaget i superklassen til modtagerobjektets klasse.

Dette er en forkert fortolkning!

Svar på spørgsmål fra sliden:

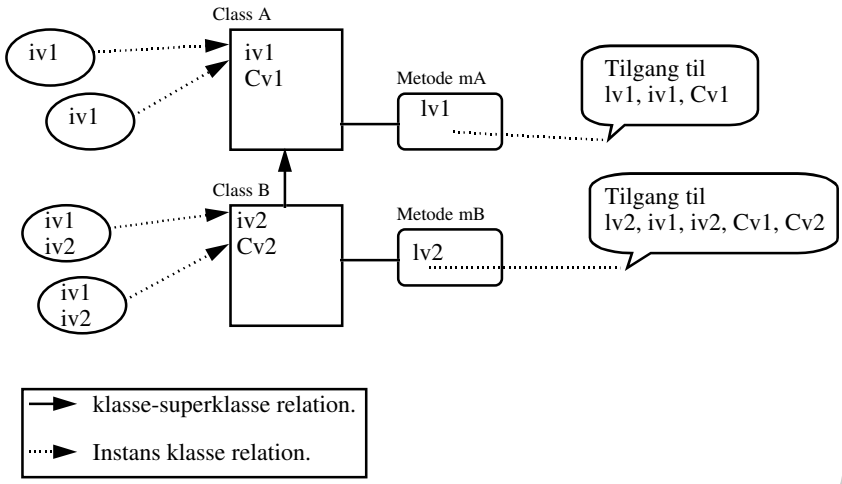
Meth placeret i klasse

	D		C		B		A	
	self **x**	super **x**	self **x**	super **x**	self **x**	super **x**	self **x**	super **x**
r **meth**	fejl	fejl	B	B	B	***A***	B	fejl
s **meth**	D	B	D	B	D	***A***	D	fejl

Tabellen indeholder klassen af metoden, som aktiveres af self **x** hhv super **x** fra hhv r meth og s meth. Som illustreret er r en instans af C, og s en instans af D i hele scenariet. Fra søjle til søjle "flytter" vi på metoden **meth.**

De ***fede klassenavne*** i tabellen er specielt bemærkelsesværdige.

Instans og klassevariable ved nedarvning.



Noter

Denne slide illustrerer, hvilke instans variable (iv..), klassevariale (Cv..) og lokale variable (lv..) der er tilgængelige i de to metoder i hhv. klassen A og klassen B.

Bemærk, at uden for metoderne i klasserne, er der ikke leksikalsk adgang til variablene. Der skal laves en metode for hver variabel, hvis værdi blot ønskes returneret til en en klient.

Blokbegrebet.

- En blok svarer til en *closure*:
  - Et parametriseret udtryk (en sekvens af beskeder), samt
  - den omgivelse, hvori blokken er defineret.
  - En blok er (ligesom alle andre objekter) et 1. klassesobjekt.
  - Statisk navnebinding: Frie navne i en blok bindes i blokkens statiske omgivelser.
- Ifølge Smalltalk tankegang er en blok
  - en sekvens af handlinger, hvis udførelse er udsat.
- En blok er et objekt, som kan udføres ved at sende en bestemt besked, **value**, til blokken.
- Blokke kan benyttes til at definere traditionelle og egne kontrolstrukturer.

***Eksempler*:**

Blok uden parametre.

```
b1 := [index := index + 1.
       array at: index put: 0].

b1 value
```

Blok med parametre.

```
b2 := [ :newElement |
        index := index + 1.
        list at: index put: newElement].

b2 value: 3
```

PS4 - Smalltalk

Noter

En blok er instans af klassen BlockContext.

Det skal bemærkes, at hvis der returneres en værdi fra en blok med ^x, terminerer og returnerer metoden, hvori blokken er defineret (altså metoden, hvori blokke findes som et udtryk). Dette sker uanset hvorfa blokken dynamisk set bliver aktiveret. (Man skal erindre, at en blok kan "flyde omkring", lige som alle ande objekter, og derfor kan konventionen omkring returnering forekomme overraskende).

Kontrolstrukturer med blokke.

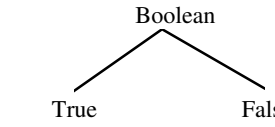
- *Selektion* (ala if-then-else) realiseres ved at sende en besked til en boolsk værdi med to blokke som parametre.

```
(a < b) ifTrue: [ max := b]
        ifFalse: [ max := a]
```

- *Iteration* (ala while) realiseres ved at sende en besked til en blok, med en anden blok som parameter.

```
[found not] whileTrue:
        [index incr.
         found := (self at: index) = x]
```

```
ifTrue: trueBlock
ifFalse: falseBlock
  self subClassResponsibility
```




```
ifTrue: trueBlock
ifFalse: falseBlock
  trueBlock value
```

```
ifTrue: trueBlock
ifFalse: falseBlock
  falseBlock value
```

I klassen blockContext:

```
whileTrue: aBlock
  (self value) ifTrue: [aBlock value.
                        self whileTrue: aBlock]
```


Noter

Bemærk at True og False i figuren ovenfor er klasser (subklasser af Boolean). De boolske metoder annonceres i klassen Boolean, men subklasserne gøres ansvarlige for deres egentlige realisering. Subklasserne redefinerer derfor disse metoder.

Flere eksempler på metoder: factorial i klassen Integer.

```
factorial
    "Answer the factorial of the receiver. Fail if the
    receiver is less than 0.

    For example, 6 factorial == 6*5*4*3*2*1."

    | tmp |
    self < 0
        ifTrue: [^self class
            raise: #domainErrorSignal
            receiver: self
            selector: #factorial
            errorString: 'Factorial is invalid on negative numbers'].
    tmp := 1.
    2 to: self do: [:i | tmp := tmp * i].
    ^tmp
```


Noter

Flere eksempler på metoder: gcd i klassen Integer.

```
gcd: anInteger
        "Answer the greatest common divisor of the receiver and anInteger."
        "Euclid's algorithm."

        | m n t |
        m := self abs max: anInteger abs.
        n := self abs min: anInteger abs.
        [n = 0]
                whileFalse:
                        [t := n.
                        n := m \\ n.
                        m := t].
        ^m
```


Noter

Flere kontrolstrukturer med blokke.

0.7 perhaps: [x := x + 1]

I klassen Float:

```
perhaps: aBlock
"Udfør blokken med sandsynlighed self"
(Float random < self)
   ifTrue: [aBlock value]
```

3 to: 5 do: [:i | res := i * i + res]

I klassen Integer:

```
to: last do: ablock
  "Udfør blokken for indeces fra  self til last"
 | i |

      i := self.
     [i <= last] whileTrue: [aBlock value: i.
                                    i := i + 1]
```

PS4 - Smalltalk

Noter

Det øverste eksempel er en lidt særartet kontrolstruktur, i hvilken vi udfører en blok med en ønsket sandsynlighed.

Det nederste eksempel svarer til Pascal´s **for** i := a **to** b **do** statement.

Mapning og filtrering af samlinger.

- Smalltalk indeholder en rig samling af klasser, som realiserer forskellige former for samlinger af objekter.
- Alle disse er subklasser af klassen Collection.
- Der findes metoder i klassen Collection, der svarer nøje til de klassiske højere-ordens funktioner på lister.

mapning:	#(2 4 6) collect: [:el | el * 2]	*#(4 8 12)*
filtrering:	#(1 2 3 4 5) select: [:el | el even]	*#(2 4)*
venstre akkumulering:	#(2 3 4 5) inject: 1 into: [:sum :el | sum - el]	*-13*

Noter

Venstre akkumuleringen svarer til ((((1-2) -3) -4) -5).

Metoder for mapning, filtrering og akkumulering.

```
collect: aBlock
        "Evaluate aBlock with each of the values of the receiver as the
       argument.  Collect the resulting values into a collection that is like
       the receiver.  Answer the new collection."

        | newCollection |
        newCollection := self species new.
        self do: [:each | newCollection add: (aBlock value: each)].
        ^newCollection

select: aBlock
        "Evaluate aBlock with each of the receiver's elements as the argument.
       Collect into a new collection like the receiver, only those elements for which
       aBlock evaluates to true.  Answer the new collection."

        | newCollection |
        newCollection := self species new.
        self do: [:each | (aBlock value: each) ifTrue: [newCollection add: each]].
        ^newCollection

inject: thisValue into: binaryBlock
        "Accumulate a running value associated with evaluating the argument,
       binaryBlock, with the current value and the receiver as block arguments.
       The initial value is the value of the argument, thisValue."

        | nextValue |
        nextValue := thisValue.
        self do: [:each | nextValue := binaryBlock value: nextValue value: each].
        ^nextValue
```

PS4 - Smalltalk

Noter

Alle metoder ovenfor findes i den meget generelle klasse Collection. Læg mærke til, hvordan alle metoderne benytter metoden med selektor **do:.**

Metoden **do:** i klassen Collection ser således ud:

```
do: aBlock
    "Evaluate aBlock with each of the receiver's elements as the argument."
    self subclassResponsibility
```

Det vil altså sige, at vi ikke kan implementere denne uden at vide noget mere specifikt om, hvilken slags Collection vi har med at gøre. Hvis vi ved, at vi har men en sekventiel samling at gøre, kan vi implementere **do:** som følger:

```
do: aBlock
    "Evaluate aBlock with each of the receiver's elements as the argument."
    1 to: self size do: [:i | aBlock value: (self at: i)]
```

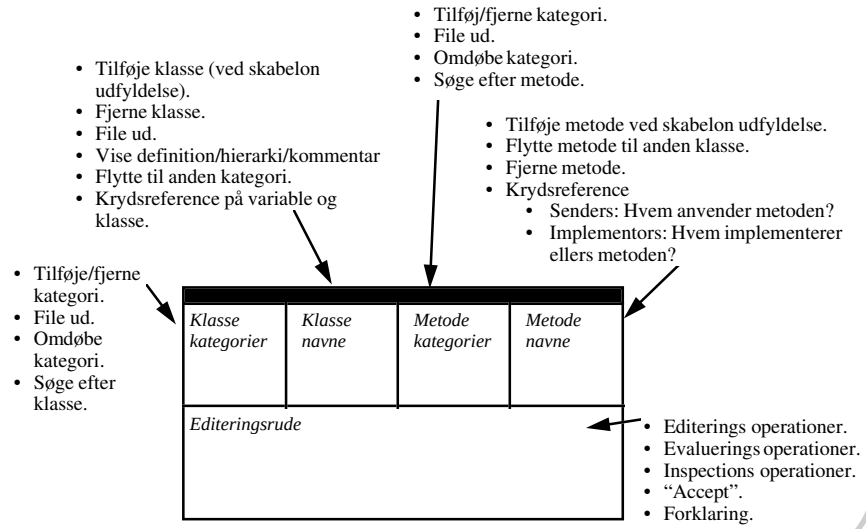
Beskeden to:do:, som sendes til et tal, har vi tidligere studeret.

Smalltalk programmeringsomgivelsen.

- Smalltalk er en **workspace-baseret** omgivelse (“image”).
  - Et fælles rum for objekter, værktøj og programbeskrivelse.
  - Workspacet kan gøres persistent mellem interaktive sessioner.
  - Programbegrebet er svagt.
  - “File ind” og “file ud” transporterer udvalgte dele af workspacet til og fra kildefiler.
- Værktøj i omgivelsen:
  - Systembrowseren og andre former for browsere.
  - Workspaces.
  - Debugger.
  - mv.

Noter

Systembrowseren.



PS4 - Smalltalk

Noter