OOMPA 2D1359 & 2D1360
Föreläsning 17: Smalltalk

Innehåll

- Olika språk
  - En kort översikt av några olika OO-språk (se separata bilder, dvs ej i denna bunt)
- Smalltalk
  - En introduktion till språket Smalltalk
    - Historik, syntax, meddelanden, grundläggande klasser, styrstrukturer
  - Klassdefinition (med instans- och klassvariabler)
  - Metoder
  - ”konstruktörer”
  - Kort-kort om Smalltalk (nio sidor som sammanfattar Smalltalk, i slutet av anteckningarna)

previous next

Historik

- Arbetet med Smalltalk började på Xerox PARC i början av 70 - talet. (3000 rader BASIC 1972)
- Skapade begreppet objektorienterat.
- Avsikten var att bygga ett kraftfullt informationssystem som skulle kunna växa och förändras i takt med med användarens idéer och önskemål.

previous next

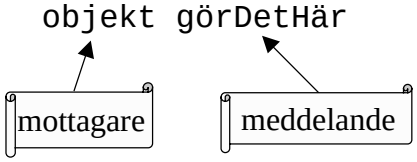
Smalltalk var en vision

- Man valde två utgångspunkter för att realisera denna idé:
  - ett beskrivnigsspråk (programspråk) som kan användas för att beskriva användarens mentala bild av tillämpningen för datorn.
  - ett interaktionspråk som används för att beskriva gränssnittet mellan användare och dator.

I Smalltalk är allt objekt

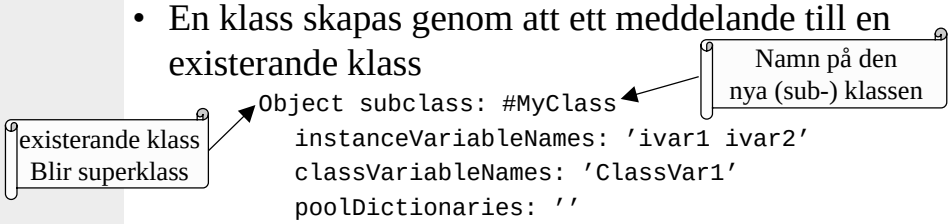
- Smalltalk är helt objektorienterat och uppbyggt kring de grundläggande begreppen: *objekt*, *instans*, *meddelande*, *klass* och *metod*.
- Allt i Smalltalk är objekt:

  ex. Tal, strängar, grafik, fönster, kompilator, klasser.
- En operation (metod) utförs genom att ett objekt (mottagaren) får ett meddelande. Objektet själv avgör vad som skall hända.



Definiera klass

- En klass skapas genom att ett meddelande till en existerande klass




```
Object subclass: #MyClass
    instanceVariableNames: 'ivar1 ivar2'
    classVariableNames: 'ClassVar1'
    poolDictionaries: ''
    category: 'MyCategory'
```

- Där man anger den nya subklassens namn, instansvariabler, klassvariabler och en del annat

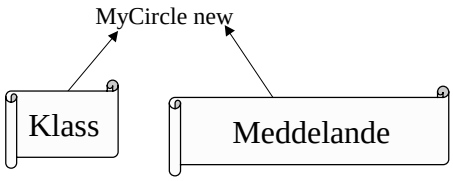
Exempel: klassdefinition MyCircle

- Vi definierar en cirkel med centrum i `center` (en instans av `Point`) och radien radius.

```
Object subclass: #MyCircle
  instanceVariableNames: 'center radius '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'OOMPA'
```

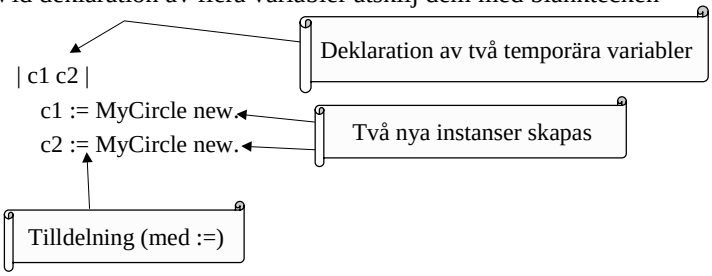
Instansiera klass

- Förutsättningar
  - Vi har definierat en klass `MyCircle`
- Problem
  - Hur skapar vi ett objekt tillhörande `MyCircle`?
- Lösning
  - Skapa instanser av klassen med hjälp av meddelandet `new`



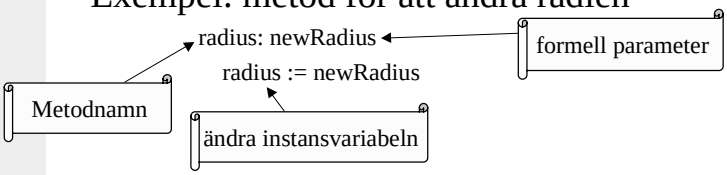
Temporär variabel

- Problem
  - Hur kan jag referera till en instans?
- Lösning
  - Använd temporär variabel
    - En temporär variabel deklareras inom ett par av vertikala "streck".
    - En variabel deklareras utan typinformation
    - Vid deklaration av flera variabler åtskilj dem med blanktecken



Ändra en instansvariabels värde

- Förutsättningar
  - Vi har deklarerat en klass med instansvariabler men kan varken läsa eller ändra deras värden
- Problem
  - Hur kan jag ändra en instansvariabel?
- Lösning
  - Skriv en mutator, dvs en metod som förändrar
- Exempel: metod för att ändra radien



Läsa instansvariabel

- Förutsättningar
  - I Smalltalk kommer man inte åt attribut i ett objekt om inte klassen erbjuder sätt att göra detta (inkapsling).
- Problem
  - Hur kommer jag åt en instansvariabel?
- Lösning
  - Skriv en inspektor. För att explicit returnera något från en metod används "stiliserad uppåtpil" (^).
- Exempel: metod för att läsa radien

```
radius
  ^radius
```

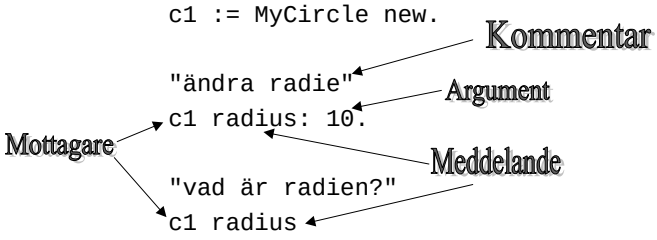
Meddelande

- Problem
  - Hur skickar jag meddelanden till objekt, dvs anropar en av dess metoder?
- Lösning
  - Meddelanden har samma namn som metoderna

```
|c1 |
  c1 := MyCircle new.

  "ändra radie"
  c1 radius: 10.

  "vad är radien?"
  c1 radius
```



Andra metoder än inspektorer

- Problem
  - Hur skriver jag en metod som använder en instansvariabel för att beräkna något?
- Lösning
  - Instansvariabeln kan alltid refereras innifrån den aktuella instansen
- Exempel: omkrets

```
circumference
  ^2 * 3.14 * radius
```

- Testa

```
c1 circumference
```

Anropa annan metod i klassen

- Förutsättningar
  - I det förra exemplet använde vi instansvariabeln radius direkt trots att det fanns en inspektor
- Problem
  - Hur använder vi en annan metod i objektet själv?
- Lösning
  - Pseudovariabeln self refererar till det aktuella objektet. För att skicka meddelanden till det aktuella objektet utnyttjas detta.

```
circumference
  ^2 * 3.14 * self radius
```


Reflektion inspektorer

- Fördelar med att använda inspektorer istället för instansvariabler direkt
  - andra metoder blir mindre bereonde av hur radien verkligen lagras. Om lagring eller andra detaljer ändras (tex om radien lagras i en databas eller i en vektor) behöver endast mutatorn förändras
  - förändringar i subklasser blir enklare
- Det ovan beskrivna gäller också mutatorer!

Initiala värden för objekt

- Förutsättningar
  - Hittills har vi använt meddelandet new för att konstruera instanser och accessmetoder för att ändra och läsa attribut.
  - Alla variabler har värdet nil om inget annat angivits
- Problem
  - Om vi använder meddelandet circumference utan att ange värde för radius får vi ett felavbrott.
  - Hur kan vi ange skönsvärden (defaultvärden) för instanvariablerna
- Lösning
  - Definiera en speciell instansmetod som ansätter defaultvärden

Forts (initieringsmetod)

- Exempel
  - Initieringsmetod

```
initialize
    self center: Point zero.
    self radius: 1.
```

  - Testa

```
| c |
    c := MyCircle new.
    c initialize.

    c circumference
```

6.28

Forts (ändra konstruktor)

- Förutsättningar
  - Programmeraren måste fortfarande komma ihåg att initiera
- Problem
  - Hur kan vi tvinga fram en initiering av skönsvärdena?
- Lösning
  - Definiera om konstruktorn new

```
new
        | newCircle |
        newCircle := self basicNew.
        newCircle center: 0@0.
        newCircle radius: 1.
        ^newCircle
```

Eller om vi utnyttjar att initialize redan är skriven

```
new
        ^self basicNew initialize
```


Vi testar MyCircle

```
c1 := MyCircle new.
c1 circumference
```

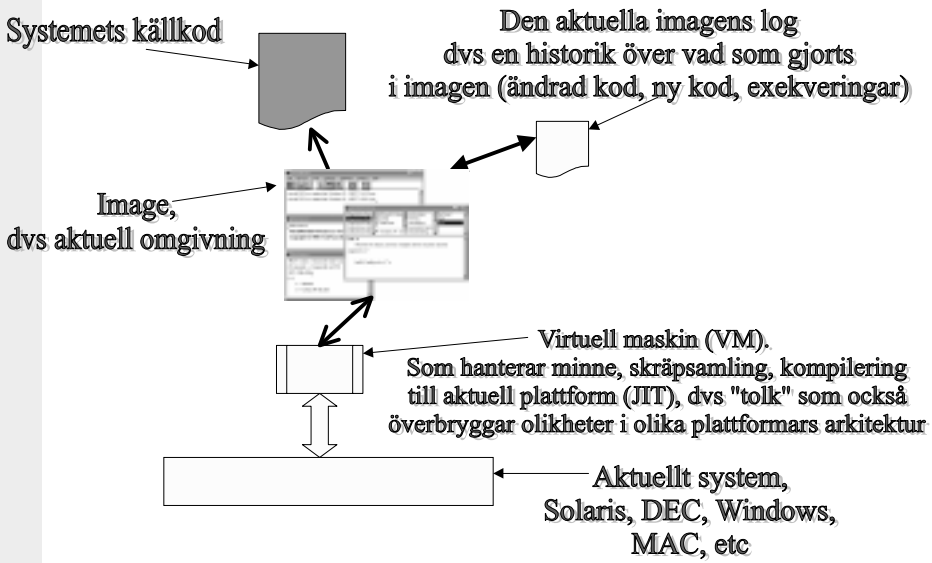
6.28

- Fungerar utan felavbrott och med förväntat resultat

VisualWorks

- Förutsättningar
  - Vi vet hur vi konstruerar Smalltalkklasser med konstruktörer och metoder.
- Problem
  - Hur implementerar vi och testar det hela i VisualWorks?

Arkitektur VisualWorks, VM, image och filer



Forts

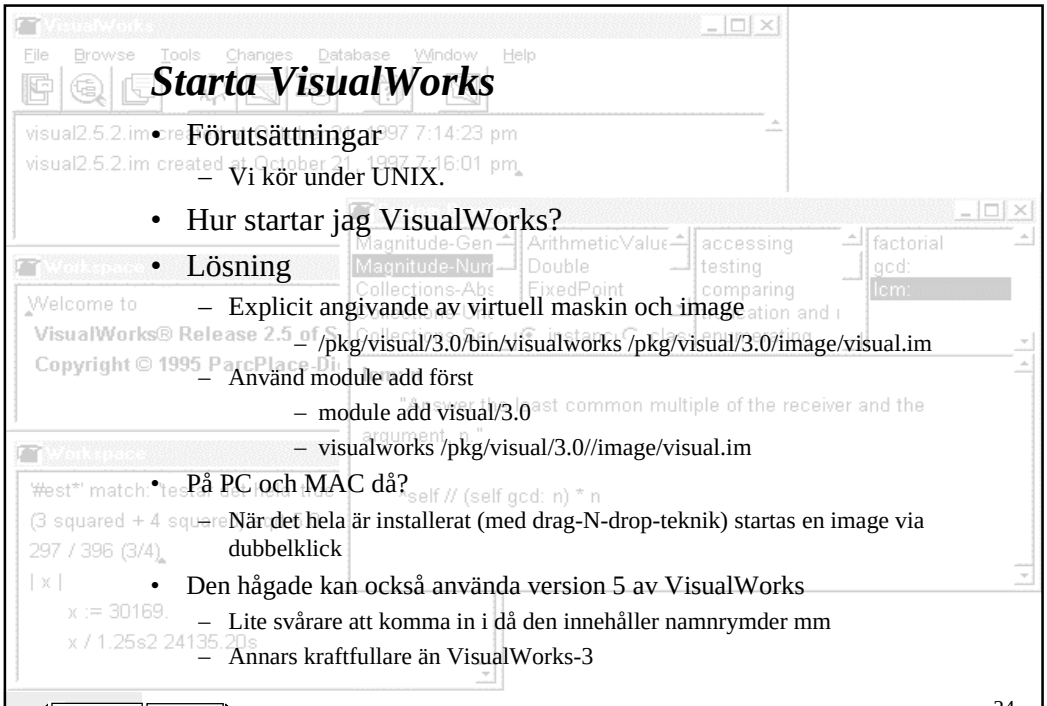
- Sammanfattning system
  - Image
    - den omgivning med fönster och kod "vi kör i"
    - namnet slutar på ".im", tex minImage.im
  - Källkod
    - systemkällkod, en fil i systemets katalog
    - egen kod + log, i en fil med den kod vi skrivit eller ändrat samt en log av vad vi gjort. Namnet slutar på ".cha"´, tex minImage.cha.
  - Virtuell maskin (VM)
    - Den enda plattformsberoende delen som översätter plattformsoberoende beskrivningar till aktuell plattform, minneshantering och tar hand om objektens "liv och död"
    - En VM per plattform (en för Solaris 5, en för Windows, osv)

För- och nackdelar med en VM

- Fördelar
  - Kod och image flyttbar mellan olika plattformar
  - Koden kan skrivas på en hög platformsoberoende nivå
  - Koden blir kompakt
- Nackdelar
  - Långsammare exekvering än om koden optimeras för aktuell arkitektur
  - Svårt att konstruera speciellt gränssnitt på ett helt plattformsoberoende sätt (fast det är ju egentligen inget VM-problem, utan mer generellt)
  - Vissa typer av program kräver plattformsspecifika "finesser"

Forts

- Överbryggande av vissa nackdelar
  - Just in time compiling (JIT)
    - dvs en översättning till riktig binärkod just före exekveringen
      - VisualWorks gör på detta sätt och många av Javas VM också
  - Anropa kod skriven i andra språk
    - I VW kan man dynamiskt länka in C-kod
  - Olika versioner av gränssnitt
    - I VW kan man känna av vilken plattform som körs och göra anpassningar (på hög nivå!) till detta.



Starta VisualWorks

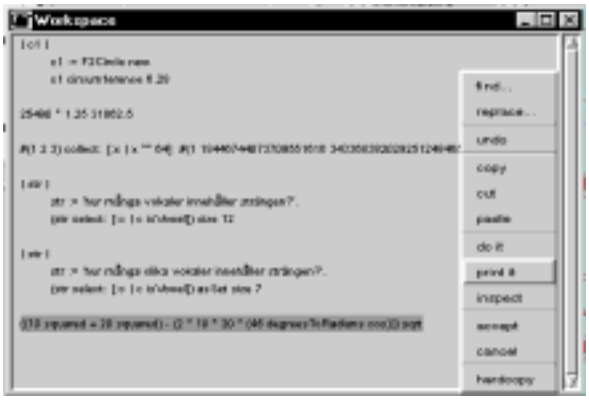
- Förutsättningar
  - Vi kör under UNIX.
- Hur startar jag VisualWorks?
- Lösning
  - Explicit angivande av virtuell maskin och image
    - /pkg/visual/3.0/bin/visualworks /pkg/visual/3.0/image/visual.im
  - Använd module add först
    - module add visual/3.0
    - visualworks /pkg/visual/3.0//image/visual.im
- På PC och MAC då?
  - När det hela är installerat (med drag-N-drop-teknik) startas en image via dubbelklick
- Den hågade kan också använda version 5 av VisualWorks
  - Lite svårare att komma in i då den innehåller namnrymder mm
  - Annars kraftfullare än VisualWorks-3

Konstruera klass i VW

- Vi har startat VisualWorks.
- Hur konstruerar vi ny klass.?
  Enklast använda Browsern.

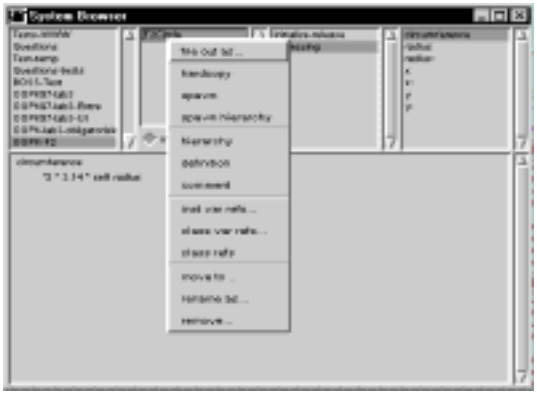
Testa kod

- Hur testar vi kod och nyskapade klasser?
- Använd ett Workspace.



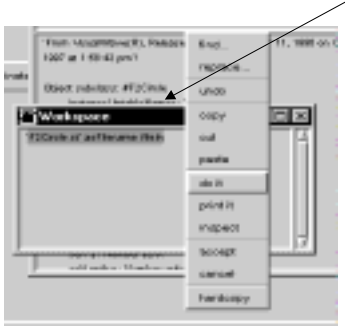
Spara kod

- Hur sparar jag min kod?
- Gör file out
  - Från Browsern



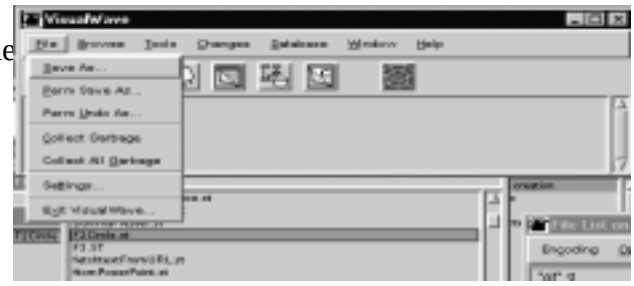
Läsa in kod

- Hur läser jag in kod som finns på fil?
- Lösning
  - Använd en Fillista
  - eller explicit meddelandet fileIn.



Spara image

- Dagen är slut vi vill gå hem och helst vill vi kunna forsätta från exakt samma utgångsläge (med öppna fönster osv) imorgon.
- Hur sparar jag mitt arbete inklusive omgivning?
- Spara image.
  - Från Launche



Starta egen image

- Hur startar jag en egen image igen?
- Lösning
  - (UNIX) visualworks minImage.im
  - (MAC, PC) dubbeklicka på ikonen

Föredefinierade klasser

- Hur använder jag fördefinierade klasser och metoder?
- Sök reda på dem i systemet. Titta vad du kan göra. Testa i tex ett Workspace.
- Exempel (vi gör dessa direkt i VW)
  - Övningar i klassen `MyCircle`
    - 3.14 mot Float pi
    - Float pi mot Double pi.