I-110 : Pensumliste

våren 1996

- B.Kirkerud: “**Object-oriented programming with SIMULA**”
  - kap. 1-6
  - kap. 8
  - kap. 9: 9.1-9.12, 9.15
  - kap. 11
  - kap. 12: 12.1-12.5, 12.7
  - kap. 14: 14.1-14.4
- **Kompendier**
  - “Eksterne prosedyrer og klasser” (tilsv. 6.4 og 9.16)
  - “Klasser og subklasser” (tilsv. 9.11-9.12)
  - “Settools” (tilsv. 14.5)
- **<u>Bruk av</u> verktøykasser**
  - Promptools, Texttools, Settools

I-110 : Innhold

I. Elementaer programmering i SIMULA
- typet språk
- imperativer
- prosedyrer + parameteroverføring
- rekursjon

II.Strukturering

III. Klasser og objekter
- klasser vs. objekter
- ref-variabler, :- vs. :=
- parametre, imperativer, THIS

IV. Klasser og subklasser
- ref-variabler, IS, IN, QUA
- arv av alle attributter
- virtuelle prosedyrer

V. Pekerkjeder
- enkle lister, to-veis/m.hode&hale
- generiske lister

VI. Settools
- Set, Ordered_Set, Sequence

VII.Filer
- Infile, Outfile
- Filformat

VIII.Tekst

I. Simula er et typet språk

(Verdi) type	Uttrykk (har en verdi) av typen
T	• variabel, (konstant,) literal • funksjonskall, typespesifikke uttrykk
integer (0)	i, 2, 97 inint, rem(8, 3), rank(‘a’) 8+i, (12*4) // 10
real (0.0)	r, 3.14, 0.5, 2.035, 2.667 inreal, cos(0), sqrt(3.14+1) 8+r-i, (2*4) / 3
boolean (false)	b, **true**, **false** letter(‘c’), digit(‘c’), not(ferdig) i=3, (3>r) **or** i=1, xx **is** yy
character (char(0))	t, ‘z’, ‘+’ inchar, char(97)
text (notext)	navn, “en tegnstreng”, **notext** intext(20) . strip, navn_1 & navn_2
referanse (none)	**ref**(X) xx, **none** **new** X, **this** X, xx **qua** yy

Imperativer

tilordning — verdi: navn **:=** uttrykk

referanse: navn **:–** referanse-uttrykk

---

input — inimage
inint, inreal, inchar

output — outimage, breakoutimage
outtext, outint, outfix, outreal, outchar

---

betinget — **if** <boolsk uttrykk> **then** <imperativ-1>
(**else** <imperativ-2>)

if b **then begin if** c **then** imp **end**
else imp

løkke — **while** <boolsk uttrykk> **do** <imperativ>

for i := <arit uttr> **step** <arit uttr> **until** <arit uttr> **do** imp

sammensatt — **begin** (<dekl.>) <imp.>; ... ; <imp.> **end**

(indre blokker)

Prosedyrer – parameteroverføring

```
begin
   integer x;
   integer array tab(1:20);

   procedure nullstill (y); integer y; y := 0;

   procedure nullstill_n (y); name y; integer y; y := 0;

   procedure init(tb, lav, hoy); integer array tb; integer lav, hoy;
      begin integer i;
         for i:= lav step 1 until hoy do tb(i) := i ;
      end;

   x := 100 ;
   nullstill(x) ;          ! x = 100;
   nullstill_n(x) ;        ! x = 0;
   nullstill(x+2) ;        ! x = 100;
   nullstill_n(x+2) ;      ! Feil ;
   init (tab, 5, 10) ;
end
```

value - “input” parametre (**refereanse** - for objekter)
name - “output” parametre (pekere - til objekter)

Parametre av enkle typer **– integer, real, character, boolean –** overføres “by default” som verdier med mindre man angir **name** overføring !!!!

Parametre av sammensatte typer **– array, text, ref**(X) **–** overføres ved **referanse** med mindre man angir **name** eller **value** overføring !!!!

Programmering og
II. Strukturering

begin

< Datastruktur
- enkle variabler
- tabeller
- klasser:
  - · subklasser
- objektmengder:
  - · tabeller
  - · pekerkjeder
  - · Settools

og hjelpedeklarasjoner >

strukturering

< Initialisering >

< Hovedløkken >

programmering

end

Fremgangsmåten ved skriving av et program

1. Problembeskrivelse (tlf.nr til ord)

Lag et program som leser inn et telefonnummer og skriver det ut som 8 ord. F.eks. når bruker skriver

`55544236`

skal programmet skrive på skjermen

`fem fem fem fire fire to tre seks`

2. Problemanalyse og presisering

Hva mener brukeren ?
Formatet på data: 55-54-42-36 eller 55544236

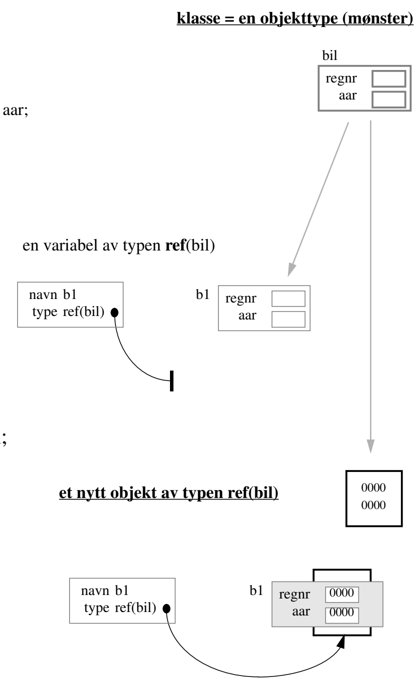
3. Grov skisse:

Datastrukturen
(hvordan lese inn og lagre data)
↓
Algoritmeanalyse
(hvordan behandle data)
↓
Grensesnitt og robusthet

4. Koding

III. Klasser og objekter

```
begin

  class bil;
   begin
      integer regnr, aar;
   end ;

ref(bil) b1;

b1 :- new bil;
```



Klasser med parametre og imperativer

```
class S_K(maks); integer maks; . . . ;
ref(S_K) s;

s :- new S_K(500);

s :- new S_K(be_om_int("Hvor stor?"));

s . maks        ! maks er et vanlig attributt ;
```

Et *aktuelt parameter* må oppgis hver gang det genereres et **nytt** objekt.

Parametre til klasser (som til prosedyrer):
- variabler eller
- tabeller
- (– men ikke prosedyrer)

overføres (som til prosedyrer) ved:
- verdi eller
- referanse
- (– men ikke ved navn)

Brukes for å:
- angi størrelsen på tabeller deklarert i klasser
- tvinge eksplisitt initialisering av noen attributter

En alternativ initialisering av objektene :

```
class S_K;
  begin
     integer nr; . . . ;
     nr := be_om_int("Skole nummer:");
  end;
ref(S_K) s;      s :- new S_K;
```

Imperativene utføres bare *en gang:* når det genereres et **nytt** objekt.

De fører til bedre modularisering enn parametre !

NB !!!

```
class S_K;
  begin
     integer maks;
     integer array tab(1:maks);
     maks := be_om_int("Maks:");
  end;
```

```
class Navn (formelle parametre) ;
 <Spesifikasjon av formelle parametre>;
begin
   < Deklarasjoner av attributtene > ;
   < Imperativer > ;
end;
```


Referanse-variabler

regnr :
aar :

integer x;
ref(bil) bil1, bil2;

bil2 **:–** **new** bil ;

Minne

Adresse	Navn	Type	Verdi
0			
1			
2			
.			
.			
.			
250	x	integer	0
251	bil1	**adresse**	**none**
252	bil2	**adresse**	373
253	...	...	...
.			
.			
.			
.			
373	regnr	integer	0
374	aar	integer	0
. . .	.		

THIS *klassenavn*

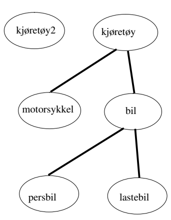
```
class bil(regnr); integer regnr;
begin
  integer aar;
  procedure parker(PP); ref(parkering) PP ;
     . . .
     PP . biler . add_element_ok ( THIS bil ) ;
  end ;
  . . .
end;
ref(bil) bil1, bil2 ;

bil1 :– new bil(123) ;
bil2 :– new bil(456) ;
```

Adresse	Navn	Type	Verdi
0			
1			
.			
.			
251	bil1	**adresse**	373
252	bil2	**adresse**	485
253	...	...	...
.			
.			
.			
373	regnr	integer	123
. . .	. . .	. . .	. . .
. . .	**This**	**adresse**	373
485	regnr	integer	456
. . .	. . .	. . .	. . .
. . .	**This**	**adresse**	485
.			

! Brukes når ***et objekt*** må sende "***seg selv***" (***referansen til seg selv***) ut i verden (til andre objekter eller til hovedprogrammet) ! (Kan også brukes inne i et inspect-imperativet.)

IV. Klasser og subklasser

klassehierarki

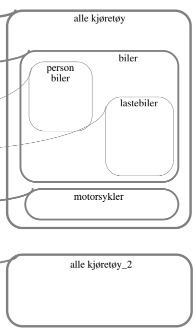


en bil **er et** kjøretøy / en lastebil **er en** bil
bil er en **umiddelbar subklasse** av kjøretøy
lastebil er en **subklasse** av kjøretøy
hver klasse er **sin egen subklasse**

klassedeklarasjoner

```
class kjøretøy;
  begin integer regnr; end;

kjøretøy class bil;
  begin integer aar; end;

bil class persbil;
  begin integer pers; end;
bil class lastebil;
  begin integer vekt; end;

kjøretøy class motorsykkel;
  begin text type; end;

class kjøretøy_2;
  begin integer regnr; end;
```

objekter av klassen



ref-"vinduet" definerer hva som er tilgjengelig

class kjøretøy ;
  begin integer regnr; **end**;

kjøretøy **class** bil ;
  begin integer aar; **end**;

bil **class** personbil ;
  begin integer pers; **end**;

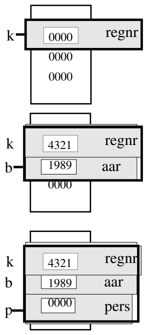
ref(kjøretøy) k **;**
ref(bil) b ;
ref(persbil) p ;

k :– **new** persbil;

k . regnr := 4321 ;
k . aar := 1989; ! Feil ;

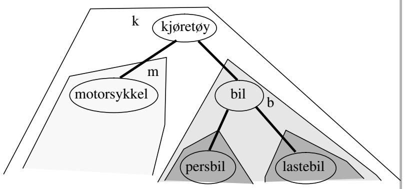
b :– k ;
b . aar := 1989
k . pers := 4; ! Feil ;
b . pers := 4; ! Feil ;

p :– k ;
p . pers := 4 ;



Subklasser

class kjøretøy ;
  begin integer regnr; **end**;
kjøretøy **class** motorsykkel ;
  begin text type; **end**;
kjøretøy **class** bil ;
  begin integer aar; **end**;
bil **class** personbil ;
  begin integer pers; **end**;
bil **class** lastebil ;
  begin integer vekt; **end**;

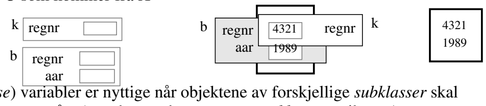


* definerer et klassehierarki

* en **ref**(X) variabel kan peke på objekter av alle subklasser av X

* en **ref**(X) variabel er et "vindu" med struktur som tilsvarer dens (**statiske**) klasse X: har vi et objekt O av en *subklasse* av X, vil en slik variabel gi oss adgang bare til de attributtene til O som kommer fra X

ref(kjøretøy) k ;
ref(bil) b ;



* **ref**(*superklasse*) variabler er nyttige når objektene av forskjellige *subklasser* skal betraktes på samme måte (som bare avhenger av *superklasse*-attributter)

* dersom en **ref**(X) variabel *k* peker på et objekt av en *ekte subklasse* av X og vi trenger adgang til objektets *subklasse*-attributter, må vi

finne subklassen til objektet	og deretter "åpne" objektet	eller lage et passende vindu

if k **in** motorsykkel **then**
else if k **in** bil **then**
else
	k **qua** motorsykkel.type:–"Honda"
	k **qua** bil.aar := 1993
	<k er ikke noe skikkelig kjøretøy> ;
		begin m:–k; m.type:–"Honda"; **end**
		begin b:–k; b.aar:=1993; **end**
		<k er ikke noe skikkelig kjøretøy> ;

inspect k **when** motorsykkel **do** **when** bil **do** **otherwise**	type :– "Honda" aar := l993 <k er ikke noe skikkelig kjøretøy> ;	

"*virtual*" : navnet/bruksmåten er felles, men verdien avhenger av den aktuelle (sub)klassen

```
element class kjøretøy(regnr) ; integer regnr;
 virtual : integer procedure avgift ;
  begin  text procedure key; key :– int_as_text(regnr) ;
         integer procedure avgift; avgift := 50 ;          ! "default" for alle kjøretøy ;
  end;
kjøretøy class motorsykkel ;
  begin  text type;
         integer procedure avgift; avgift := 30 ;
  end;
kjøretøy class bil ;
  begin  integer aar; end;
bil class personbil ;
  begin  integer pers; end;
bil class lastebil ;
  begin  integer vekt;
         integer procedure avgift; avgift := vekt * 5;
  end;
class Parkering;
 begin
  ref(Set) lasteb, motors, personb, andre;
  integer kasse;
  boolean procedure parker(k); ref(kjøretøy) k ;
   if <flere parkeringsplasser> then
    begin
      if k is personbil then  personb . add_element_ok(k) else
      if k is lastebil then  lasteb . add_element_ok(k) else
      if k is motorsykkel then  motors . add_element_ok(k)
      else andre . add_element_ok(k) ;
      kasse := kasse + k . avgift ;
      parker := true ;
    end else parker := false ;
 end Parkering ;
```