# Settools

## Verktøy for å håndtere objektmengder

Bjørn Kirkerud

Oppdatert 28. oktober 1992

## 1 Objekter og objekt-mengder

Mange programmer holder styr på objekter og objekt-mengder. Ofte når man skal konstruere et program, kan det være fornuftig ganske tidlig i arbeidet – før man går igang med å skrive deklarasjoner og imperativer, men vanligvis etter at man har funnet ut hvilke handlinger programmet skal kunne utføre – å bestemme hva slags objekter og hvilke objekt-mengder programmet skal holde styr på. Et program er ofte en modell av en del av virkligheten, og man kan som regel få gode idéer til hvilke objekter og objekt-mengder programmet skal håndtere ved å se på den virkligheten programmet skal modellere.

Noen eksempler:

**Et skole-program:** Anta man skal lage et kommando-styrt program som skal holde oversikt over ymse data om barna i en skole. Det kan være fornuftig å lage programmet slik at det oppretter et objekt for hvert barn i skolen, og at det altså skal holde styr på barne-objekter, d.v.s. objekter som inneholder data som beskriver enkelt-barn. Gitt en oversikt over hvilke kommandoer programmet skal kunne utføre (vi tar ikke med noen beskrivelse av disse kommandoene her), bestemmer man at programmet for det første skal holde styr på en objekt-mengde som skal ha som medlemmer et objekt for hvert barn, dernest en objekt-mengde for hver klasse på skolen (som selvfølgelig skal ha som medlemmer et objekt for hvert barn i klassen).

**Et student/kurs-program:** Et program som skal holde oversikt over data om studenter og kurs vil rimeligvis operere med objekter av to slag: Student-objekter og kurs-objekter. Gitt at programmet bl.a. skal holde rede på hvilke kurs hver student følger og også hvilke studenter som følger hvert kurs, bør det trolig holde styr på ihvertfall følgende objekt-mengder:

1. En objekt-mengde med alle student-objektene som medlemmer.
2. En objekt-mengde med alle kurs-objektene som medlemmer.
3. For hvert student-objekt: mengden av alle kurs studenten følger.
4. For hvert kurs-objekt: en mengde som har som medlemmer alle studentene som følger kurset.

**Et observasjons-program:** Dette programmet skal holde styr på data om en del målestasjoner og observasjoner (av en eller flere størrelser) foretatt ved disse stasjonene. Programmet bør trolig bruke objekter av to slag: Stasjons-objekter (ett slikt objekt for hver målestasjon) og observasjons-objekter (ett for hver observasjon). Videre bør det holde rede på følgende objekt-mengder:

1. En mengde med alle stasjons-objektene som medlemmer.
2. For hver stasjon en mengde som har som medlemmer et objekt for hver observasjon som er foretatt ved stasjonen.

Etter at man har bestemt seg for hva slags objekter og hvilke objekt-mengder et program skal holde styr på, og også hvilke handlinger programmet skal kunne utføre, er det gjerne på tide å definere (deklarere) en passende data-struktur med tilhørende operasjoner. Dersom man bruker Simula eller lignende programmerings-språk, vil en slik definisjon vanligvis bestå av:

- Deklarasjon av en klasse for hvert objekt-slag. Denne deklarasjonen inneholder deklarasjoner av lokale variable og prosedyrer, d.v.s. deklarasjoner som beskriver data-strukturen som alle objekter av dette slaget har, og av prosedyrer som gjør noe med eller observerer data-strukturen til et enkelt objekt.
- Deklarasjon av variable som skal brukes til å holde styr på objekt-mengdene. Dette kan gjøres på mange ulike måter, noen av disse måtene beskrives i dette notatet.
- Deklarasjon av prosedyrer som brukes til å utføre nødvendige operasjoner på objektmengdene. Hvilke operasjoner som er nødvendige avhenger selvfølgelig av hvilke handlinger programmet skal kunne utføre, men det er vanlig at man trenger prosedyrer for å utføre ihvertfall følgende operasjoner:

  1. *Melde inn* et nytt objekt i en objekt-mengde,
  2. *Finne fram* til et spesifisert medlem av en objekt-mengde (f.eks. finne et medlems-objekt med et spesifisert navn),
  3. *Melde ut* (fjerne) et spesifisert medlem fra en objekt-mengde,
  4. *Gå sekvensielt gjennom* medlemmene i en objekt-mengde, d.v.s. gå gjennom medlemmene ett for ett),
  5. *Observere* en objekt-mengde på ulike måter, f.eks. telle antall medlemmer i mengden eller finne gjennomsnittlig verdi på et eller annet data-element som finnes hos alle objektene i mengden.

I I110 beskrives tre ganske ulike metoder for å håndtere objekt-mengder:

**Array-metoden:** Her deklareres det et **ref**-array og en heltalls-variabel for hver objekt-mengde. Array'et skal brukes til referanser til objektene som er medlemmer i mengden, heltalls-variablen skal til enhver tid ha som verdi antall elementer i mengden. Dersom array-metoden brukes i skole-programmet beskrevet i det første eksemplet ovenfor, vil programmet kunne inneholde f.eks. følgende deklarasjoner (vi forutsetter at det maksimalt er 500 elever på skolen, at det er seks klassetrinn hver med opp til 4 parallelle klasser og at hver klasse har høyst 30 elever):

```
ref(Barn) array Skole(1 : 500),
                Klasse(1 : 6, 1 : 4, 1 : 30);
integer         antall_barn_i_skolen;
integer array   antall_barn_i_klasse(1 : 6, 1 : 4);
```

**Pekerkjede-metoden:** Idéen i denne metoden er at for hver objekt-mengde deklareres det for det første en **ref**-variabel som skal ha som verdi en referanse til det første medlemmet i mengden. I tillegg sørger man for (ved en passende deklarasjon i klassen som beskriver medlemsobjektene) at hvert objekt som kan være medlem av en mengde inneholder en **ref**-variabel som kan brukes til å inneholde en referanse til det neste objektet i mengden. Dersom et objekt kan være medlem av flere ulike objekt-mengder, må det inneholde en slik neste-peker for hver av de mulige objekt-mengdene det kan være med i.

Dersom pekerkjede-metoden brukes i skole-programmet, vil det kunne inneholde f.eks. følgende deklarasjoner:

```
ref(Barn)       forste_barn_i_skolen;
ref(Barn) array forste_barn_i_klasse(1 : 6, 1 : 4);
```

I tillegg utvides deklarasjonen av klassen Barn med følgende deklarasjoner:

```
class Barn;
  begin
    ...
    ref(Barn) neste_barn_i_skolen,
              neste_barn_i_klassen;
    ...
  end Barn;
```

**Settools-metoden:** Settools er en ferdig-deklarert klasse som inneholder verktøy som kan brukes til å holde styr på objektmengder. Resten av dette notatet beskriver det viktigste av dette verktøyet og hvordan det kan brukes.

## 2 Sett og sekvenser

Settools inneholder verktøy for å håndtere objekt-mengder av tre slag:

- Sekvenser
- Sett
- Ordnete sett

Objekter som skal kunne være medlemmer av et sett eller et ordnet sett må være *identifiserbare*, hvilket i denne forbindelse betyr at det må finnes en såkalt *identifikasjons-nøkkel* – vanligvis et navn, fødselsnummer, identifikasjons-nummer eller lignende – som er slik at alle medlemmene i settet har ulike nøkkel-verdier. Objekter som skal være medlemmer av en sekvens trenger ikke å være identifiserbare, de trenger altså ikke ha noe entydig navn eller lignende.

Vi kan si at et *sett* (gjerne ordnet) er en mengde av objekter som har entydig identifikasjonsnøkkel, mens en *sekvens* er en mengde av objekter som ikke trenger ha noen slik nøkkel. Forskjellen på *sett* og *ordnete sett* er at i ordnete sett ligger medlemmene sortert etter stigende nøkkel-verdi, mens de i sett ligger etter innmeldings-rekkefølge.

Dersom et program skal holde styr på en objekt-mengde der medlems-objektene ikke er identifiserbare, d.v.s. ikke har noe navn eller annet som er slik at to medlemmer alltid har ulik verdi, så må man la objekt-mengden være en sekvens. Dersom objektene er identifiserbare, kan man – om man vil – la objekt-mengden være et sett (gjerne ordnet)

En viktig fordel sett (medregnet ordnete sett) har framfor sekvenser, er at identifikasjons-nøkler kan brukes til å spesifisere hvilket medlem av et sett man vil se på eller fjerne fra settet. Kjenner man hvilken nøkkel et bestemt medlem av et sett har, kan man umiddelbart få tak i det medlemmet. Skal man ha tak i et bestemt medlem av en sekvens, må man gå sekvensielt gjennom alle medlemmene av sekvensen, ett for ett, inntil man finner det man leter etter.

Noen eksempler som viser bruk av sett og sekvenser:

**Skole-programmet** (forts.): Dersom man antar at alle barn på skolen har entydig navn, kan man bruke navn som identifikasjons-nøkkel i skole-mengden og klasse-mengdene. Det betyr at man kan la disse mengdene være sett, ordnete sett eller sekvenser. Vi velger å bruke ordnete sett, med navnet som identifikasjons-nøkkel. Årsaken til at vi ikke lar mengdene være sekvenser, er at flere av de kommandoene programmet skal kunne utføre, er slik at man skal finne fram til barn med bestemte navn (enten for å se på eller endre data som er lagret om barnet, eller for fjerne alt som er lagret om barnet), og da er det fordelaktig å bruke sett eller ordnete sett.

Vi velger å bruke ordnete sett (og ikke bare vanlige sett) fordi vi gjerne vil ha lister ordnet alfabetisk etter navn.

Dersom barna på skolen ikke har entydige navn, kan man bruke person-nummer som identifikasjons-nøkkel. Da er det imidlertid ingen grunn til å la settene være ordnete, da det vel neppe er noen god grunn til å ha lister som er ordnet etter stigende person-nummer.

**Student/kurs-programmet** (forts.): Person-nummer kan brukes som nøkkel for studenter, kursbetegnelse (f.eks. I110) som nøkkel for kursene. Vi lar alle student-mengder være sett, kursmengdene lar vi være ordnete sett.

**Observasjons-programmet** (forts.): Det er rimlig å anta at målestasjonene har entydige navn som kan brukes som nøkkel. Mengden av alle målestasjons-objekter lar vi være et ordnet sett. Observasjonene er neppe vanligvis identifiserbare, hvilket betyr at mengden som består av alle observasjoner ved en stasjon må være en sekvens. I noen situasjoner kan det selvfølgelig tenkes at det kan lages entydige nøkler for observasjonene, f.eks. kan kanskje tidspunktet brukes (sett sammen år, måned, dag, time, minutt og sekund til en nøkkel), eller kanskje finnes det løpenummere i observasjons-protokollene som identifiserer observasjonene. I slike situasjoner er det mulig å la mengden av observasjoner foretatt ved en stasjon være et sett.

# 3 Settools: Grov struktur

Klassen Settools har følgende grov-struktur:

```
class Settools;
  begin
    class element;
      begin
        <Deklarasjoner av to prosedyrer (se nedenfor) >;
      end element;

    element class set;
      begin
        <Deklarasjoner av verktøy for å håndtere sett (se nedenfor) >;
```

```
    end set;

  element class ordered_set;
    begin
      <Deklarasjoner av verktøy for å håndtere ordnete sett (se nedenfor) >;
    end ordered_set;

  element class sequence;
    begin
      <Deklarasjoner av verktøy for å håndtere sekvenser (se nedenfor) >;
    end sequence;

  <Deklarasjoner av noen hjelpe-klasser >;

end Settools;
```

Ethvert objekt som skal være medlem av et sett, et ordnet sett eller en sekvens, må være en instans av klassen element eller av en subklasse av element. Det betyr at en klasse som beskriver objekter som skal kunne være medlemmer av sett, ordnete sett eller sekvenser, må deklareres som en subklasse av element. Eksempler gis nedenfor. Legg merke til at klassene set, ordered_set og sequence er subklasser av klassen element. Det betyr at sett, ordnete sett og sekvenser kan være medlemmer i sett, ordnete sett og sekvenser.

Dersom et objekt skal være medlem av et sett eller et ordnet sett, må klassen som objektet er en instans av, inneholde deklarasjon av en **text**-prosedyre med navn key som for ethvert objekt skal ha som verdi objektets entydige identifikasjons-nøkkel.

Noen eksempler:

**Skole-programmet** (forts.):

Data-struktur for skole-programmet kan f.eks. deklareres slik dersom Settools benyttes:

```
element class Barn;
  begin

    text navn;
    <Deklarasjoner av andre data-attributter >;

    text procedure key; key :– navn;

    <Deklarasjoner av diverse andre prosedyrer >;

  end Barn;

ref(ordered_set) Skole;
ref(ordered_set) array Klasse(1 : 6, 1 : 4);
```

**Student/kurs-programmet** (forts.):

En data-struktur for dette programmet kan deklareres som følger:

```
element class Student;
  begin
    text persnr, navn, adresse;
    ref(ordered_set) kurs_liste;
    <Deklarasjoner av andre data-attributter>;
    text procedure key; key :– persnr;
    <Deklarasjoner av diverse andre prosedyrer>;
  end Student;
element class Kurs;
  begin
    text kursnummer;
    ref(ordered_set) studenter;
    <Deklarasjoner av andre data-attributter>;
    text procedure key; key :– kursnummer;
    <Deklarasjoner av diverse andre prosedyrer>;
  end Kurs;
ref(ordered_set) Alle_kurs, Alle_studenter;
```

**Observasjons-programmet** (forts.):
En mulig data-struktur:

```
element class Stasjon;
  begin
    text stasjons_navn;
    ref(sequence) observasjoner;
    <Deklarasjoner av andre data-attributter>;
    text procedure key; key :– stasjons_navn;
    <Deklarasjoner av diverse andre prosedyrer>;
  end Stasjon;
element class Observasjon;
  begin
    <Deklarasjoner av data-attributter>;
    <Deklarasjoner av prosedyrer>;
  end Observasjon;
ref(ordered_set) Stasjoner;
```

# 4 Hvordan ta Settools i bruk i et program

Filen `/Home/kurs1/i110-adm/lib/settools.sim` inneholder en deklarasjon av klassen Settools. Den greieste måten å bruke Settools i et program er å deklarere Settools som en ekstern klasse. Det kan gjøres slik:

```
external class Settools;

Settools begin   ! Dette er en blokk med Settools som såkalt prefix
                 ! Det betyr at alt som er deklarert i Settools (men ikke skjult)
                 ! er tilgjengelig inne i denne blokken;

    <Eget program.   Her er alle Settools-verktøyene tilgjengelige>;

end bruk av Settools;
```

For å kompilere et program som bruker en ekstern klasse som hentes fra `/Home/kurs1/i110-adm/lib`-området (f.eks. Settools), må man bruke kommandoen `sc -tools`. Dette er beskrevet nokså utførlig i notatet om eksterne prosedyrer og klasser.

# 5 De viktigste verktøyene i Settools

## 5.1 Klassen element

Klassen element er deklarert som følger:

```
class element;
    virtual: text procedure key;
             procedure display;
  begin
    text procedure key; key :- notext;
    procedure display; outtext(key);
  end of element;
```

Virkningen av at prosedyrene key og display er spesifisert som å være virtuelle prosedyrer, er at det kan deklareres egne versjoner av disse to prosedyrene i alle subklasser av element.

## 5.2 Det viktigste verktøyet i ordered_set og set

Skisse av klassen ordered_set:

```
element class ordered_set;
  begin
    <Deklarasjon av data-struktur (skjult) >;
    Boolean procedure add_element_ok(an_element); ref(element) an_element;
      if <dette settet allerede har et medlem med key = an_element . key>
        then add_element_ok := false
        else begin
            <Meld an_element inn i dette settet>;
            add_element_ok := true;
          end;
    Boolean procedure remove_element_ok(rkey); text rkey;
      if <dette settet har et medlem med key = rkey>
        then begin
```

```
            <Meld dette medlemmet ut av dette settet>;
            remove_element_ok := true;
          end
        else remove_element_ok := false;

    ref(element) procedure find_element(fkey); text fkey;
      if <dette settet har et medlem med key = fkey>
        then find_element :- <dette medlemmet>
        else  find_element :- none;

    Boolean procedure is_empty;
      is_empty := <sann om dette settet ikke har noen medlemmer>;

    integer procedure size;
      size := <antall medlemmer i dette settet>;

    ref(element) procedure first_element;
      first_element :- <det første medlemmet i dette settet>;
          ! NB: Medlemmene i et ordered_set er ordnet
          !       alfabetisk på stigende key-verdi;

    ref(element) procedure next_element;
      next_element :- <det neste medlemmet i dette settet>;

    procedure display;
      <Skriver en liste med alle medlemmene i dette settet>;

  end of ordered_set;
```

Klassen set inneholder nøyaktig de samme prosedyrene som ordered_set, men her er medlemmene ordnet etter innmeldings-tidspunkt.

### 5.3 Det viktigste Sequence-verktøyet

Skisse av klassen sequence:

```
element class sequence;

  begin

    <Deklarasjon av data-struktur (skjult) >;

    procedure append(el); ref(element) el;
      <Plasserer el sist i denne sekvensen>;

    Boolean procedure is_empty;
      is_empty := <sann om denne sekvensen ikke har noen medlemmer>;

    integer procedure size;
      size := <antall medlemmer i denne sekvensen>;

    ref(element) procedure element_number(elnr); integer elnr;
      element_number :- <medlem nr. elnr i denne sekvensen>;

    ref(element) procedure first_element;
        first_element :- <første medlem i denne sekvensen>;
```

```
    ref(element) procedure next_element;
        next_element :- <neste medlem i denne sekvensen>;

    procedure display;
      <Skriver en liste med alle medlemmene i denne sekvensen>;

    ref(element) procedure head;
      head :- <første medlem i denne sekvensen>;

    ref(sequence) procedure tail;
      tail :- <sekvensen som fås ved å fjerne
               første medlem i denne sekvensen>;

    procedure remove_current;
      <Fjerner det elementet som siste bruk av
       first_element/next_element pekte ut, fra sekvensen>;

    procedure remove_head;
      <Fjerner første element i sekvensen>;

    procedure remove_last;
      <Fjerner siste element i sekvensen>;

    ref(sequence) procedure last_element;
      last_element :- <siste element i sekvensen>;

  end of sequence;
```

# 6 Eksempler på bruk av Settools

Filene

- `/Home/kurs1/i110-adm/filer/sim/elev2.sim`
- `/Home/kurs1/i110-adm/filer/sim/bolig.sim`
- `/Home/kurs1/i110-adm/filer/sim/observ.sim`
- `/Home/kurs1/i110-adm/filer/sim/juleprog.sim`

inneholder fullstendige eksempler på bruk av Settools.

# 7 Implementasjon av Settools

Filen `/Home/kurs1/i110-adm/lib/settools.sim` inneholder som nevnt ovenfor deklarasjonen av Settools. Akkurat som man ikke trenger å forstå innmaten av en radio for å kunne bruke den, er det ikke på noen må nødvendig å forstå detaljene i implementasjonen av Settools for å kunne bruke verktøyet i Settools. Ta gjerne en titt på filen `/Home/kurs1/i110-adm/lib/settools.sim`, men gjør ikke noe forsøk på å forstå alt sammen (dersom du da ikke er spesielt interessert).