4
Basal Objekt-orienteret Programmering I.

Klasser i forhold til abstrakte datatyper og record-typer.

Variable og operationer.

Klasse-interfaces.

Klasser og typer.

Klasse-instantiering og initialisering.

Operationer på klasser kontra konventionelle procedure-kald.

Klienter og suppliers: relationer mellem klasser.

Eksempel på en klasse i Eiffel.

PS1 -- Basal objekt-orienteret programmering I

Noter

I nærværende kapitel i disse noter tales om objekt-orienteret programmering generelt, og stort set uafhængigt af et konkret programmeringssprog. Der vil forekomme syntaks for klasser og operationer i klasser, i en notation, som ikke stammer fra noget bestemt programmeringssprog. Dette kapitel handler altså ikke om Eiffel, men om objekt-orienteret programmering i almindelighed.

Det vil klart blive angivet i titlen af slides, når der tales om Eiffel-specifikke emner. *Med mindre andet er nævnt, vil vi i resten af noterne referere til Eiffel3, når vi blot skriver Eiffel.* Som omtalt i introduktionen til disse noter, er der meget væsentlige forskelle mellem Eiffel, som beskrevet i "Object Oriented Software Construction" og vores implementation af sproget. Bøgerne (manualerne) "Eiffel the Language", "Eiffel the Libraries og "Eiffel the Environment" afspejler vores implementation af sproget.

De læsere, som er meget nysgerrige efter, hvad Eiffel2 (det gamle Eiffel) er for et sprog, kan henvises til appendix B i OOSC. Dette appendix indeholder en god og hurtig læst oversigt over sproget. Den tilsvarende oversigt for Eiffel3 kan findes i kapitel 1 af bogen “Eiffel the Language”.

Klassebegrebet i forhold til ADT-er.

- En klasse er en abstraktions-form i et programmeringssprog, som realiserer en abstrakt datatype.
- Klassebegrebet involverer en række detaljer, som sædvanligvis ikke diskuteres i forbindelse med abstrakte datatyper, bl.a.:
  - Beskrivelse af tilstand.
  - Instantiering af klasser.
  - Synlighed af klassers egenskaber.
  - "Klasse-sammensætning" ved brug af nedarvning.

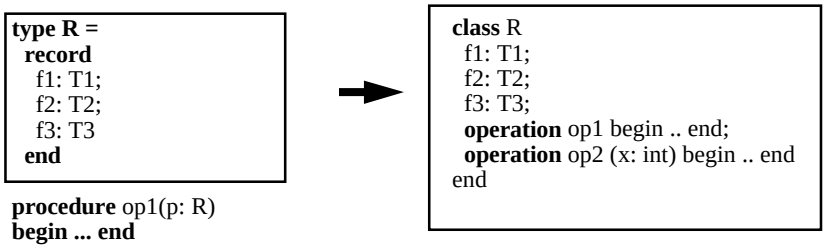
Noter

Abstraktion: Vi kender andre abstraktions-former, nemlig procedureabstraktion og funktionsabstraktion. En procedure er en abstraktion over en mængde af kommandoer (sætninger), som gives et navn, og som typisk generaliseres med et antal parametre. En funktion er tilsvarende en abstraktion over et udtryk.

Klasser som abstraktion: man kan sige, at en klasse er en abstraktion over et antal definitioner, som igen i dette tilfælde gives et navn (og måske generaliseres med et antal simple parametre eller type parametre).

Klassebegrebet i forhold til records: Øget indkapsling.

En klasse kan opfattes som en udvidelse af record-begrebet:



procedure op1(p: R)
begin ... end

procedure op2(p: R; x: int)
begin ... end

Kun data er *indkapslede* i recorden.

Både operationer og data er *indkapslede* i klassen

Noter

Forskellen mellem en record og en klasse er, at procedurerne, som tager record-typen som parametre, bogstaveligt talt er "puttet" ind i klassen (og kassen).

Endvidere er første parameter til procedurerne implicit for klassens operationer; det er nemlig altid en instans af klassen selv. Dette er en meget vigtig iagttagelse.

Senere i dette kapitel vil vi se på forskelle i anvendelse (kald) af operationerne.

Syntaksen for den viste klasse tilhører ikke noget bestemt programmeringssprog. Vi vil senere i denne forelæsning vende tilbage til, hvordan klasser ser ud i Eiffel.

Variable og operationer i klasser.

En klasse kan indeholde:

- Data
- Operationer

Data

Variable eller konstanter, som repræsenterer objekters tilstand.

- kan være tilknyttet instanser af klasser (instans-variable) eller klassen som så (klasse-variable).
- er *erklæret* af en bestemt type (statisk typing).
- kan værer *allokeret* statisk eller dynamisk.

Operationer

Procedurer eller funktioner.

- parametre og resultat er erklæret af bestemte typer (statisk typing).

Noter

Ved **statisk typing** forstår vi, at typen af ethvert udtryk i en programtekst kan bestemmes ved at analysere programteksten. Det betyder for det meste, at typen af enhver variabel og enhver parameter direkte fremgår af programteksten i erklæringen af variablen hhv. parameteren. (Se også slide om datatyper i kapitlet "Grundbegreber").

Operationerne i klassen kan kategoriseres som konstruktorer, transformatorer og accessorer på lignende måde som vi gjorde med funktionerne i en algebraisk specifikation. Konstruktorerne er vigtige i forbindelse med skabelse og initialisering af objekter. Dette vender vi tilbage til senere i dette kapitel.

Klasseinterfaces og “information hiding”.

- 'Interface' betyder 'grænseflade'.
- En klasse's interface er den mængde af klassens egenskaber, som er synlig og brugbar udadtil i forhold til klassen.
- Andre af klassens egenskaber er skjult, og dermed kun brugbare indadtil.
- I forskellige objekt-orienterede programmeringssprog er der varierende konventioner for at definere, hvilke egenskaber, der tilhører klassens interface:
  - Alle data er interne, og alle operationer er en del af klasseinterfacet.
  - Det angives eksplicit, hvilke egenskaber der tilhører klassens interface.
    - Operationer kan være interne, data kan tilhøre klasse-interfacet.
    - Klassens egenskaber kan grupperes i en privat afdeling og en offentlig afdeling.
    - Klassen angiver en liste af egenskaber, som"eksporteres" fra klassen.

Noter

Et interface udtrykker, hvilke egenskaber der kan bruges udaftil i forhold til klassen. Hvis ikke alle klassens egenskaber kan bruges af klienter, er der nogle egenskaber, der er private. Dette er udtryk for **synlighedskontrol** af klassens egenskaber. På engelsk bruges ofte termen **'information hiding'.** Synlighedskontrol (eller information hiding) ved definition af interfaces er et nøgle-emne i objekt-orienteret programmering. Årsagen til, at dette er så vigtigt, er den simple, at hvis en egenskab ikke er (direkte) synlig og brugbar for en klient, er der mulighed for at man kan ændre lokalt på egenskaben, uden at det får globale konsekvenser for alle klienter. Dette er brandmurseffekten, som vi tidligere har omtalt.

Denne slide er ikke helt præcis med hensyn til brug af ordet interface. Interfacet kan dels være af betydning for, hvilke egenskaber en klient kan bruge fra en "supplier" klasse; der kan være tale om et andet interface mellem en klasse og sub-klasser (specialiseringer) af klassen. Dette andet interface vil vi vende tilbage til i første forelæsning om nedarvning.

Sproget Smalltalk er et eksempel på et sprog, hvor alle operationer udgør klassens interface (ingen data -- instansvariable -- er en del af interfacet). I C++ kategoriseres egenskaber som offentlige og private. Også i Eiffel kategoriseres egenskaber i nogle grupper, dog ikke i en privat og en offentlig afdeling, men efter hvilke klasser, der kan bruge egenskaben. I Eiffel2 (det gamle Eiffel) indeholder hver klasser en eksport liste, der opremser navnene af de egenskaber, der synlige i andre klasser.

Klasser som typer, moduler og simple typer.

- En klasse kan anvendes som en datatype i et objekt-orienteret programmeringssprog:
  1. Der kan erklæres variable af klasse-typer.
  2. Operationers parametre kan type-erklæres med klasse-typer.
  3. Funktioner kan returnere data af en klasse-type.

- Et modul er en syntaktisk ramme om en mængde definitioner:
  - Et modul er ikke en type, men kan indeholde en eller flere typedefinitioner.
- En simpel, sprogdefineret type (integer, character, mv) ligner en klasse ved at
  - De kan bruges som forskrift for objektdannelse.
  - De har et antal tilknyttede operationer.

  og er forskellige ved at
  - Syntaksen af operationskald er ofte speciel i forhold til anvendelse af opeationer på objekter af klasser.
  - Simple typer kan typisk ikke udvides med flere egenskaber. Det kan klasser derimod.

Noter

Følgende er marginale bemærkninger i forhold til emnet:

De tre karakteristika (1--3) øverst på denne slide sammenfatter man ofte til udsagnet: "Objekter er af første klasse".

Andre elementer i programmeringssprog er ikke af første klasse, f.eks. procedurer i Pascal eller Eiffel. Man kan ikke lagre procedurer i variable, og procedurer kan ikke returneres som resultat af en funktion (men i Pascal kan de overføres som parametre).

Der findes programmeringssprog, hvor procedurer er af første klasse, f.eks. i Scheme, som er en Lisp dialekt. Det fører for vidt at komme ind på dette her.

Om moduler i forhold til klasser: Når man vil bruge et modul ligesom en klasse definerer man ofte en type i modulet, og en række operationer der tager parametre af denne type, eller returnerer objekter af typen.

I Eiffel er de simple sprogdefinerede typer definerede af klasser. Disse kaldes Basic Classes, og er beskrevet i kapitel 32 i *Eiffel the Language.*

Eksempel: anvendelse af klasser som typer.

```
class C
  procedure op1(p:D);
  ...

end
```

```
class D
  function op2(p: C): D
  ...

end
```

```
x: C;
v, w: D

  her instantieres og initialiseres objekter ;

x.op1(v);

if w.op2(x) = v
then x.op1(v) end
```

Karakteristika ved et objekt-orienteret program:
Alle operationer anvendes på objekter.

Noter

Det illustreres, at klasser anvendes i variabelerklæringer, i parameterlister og som resultat af en funktion på samme måde som simple typer.

Ovenfor antages, at proceduren op1 er del af klientinterfacet for klassen C. Tilsvarende antages, at funktionen op2 er en del af interfacet for klassen D.

Instantiering af klasser & initialisering af objekter.

- En klasse kan opfattes som en skabelon for dannelse af instanser (objekter) af klassen.
- Objekt-dannelse sker i to faser:
  - Instantiering: Lager af passende størrelse afsættes til objektet.
  - Initialisering: Det afsatte lager tilskrives passende start-værdier af data-felterne.

Det er vigtigt, at programmeringssproget understøtter, at instantiering efterfølges af en fornuftig initialisering af objektet.

Noter

Statisk og dynamisk instantiering.

Man taler om to forskellige former for instantiering af klasser til objekter:

Statisk instantiering:
• En instantiering, som foreskrives i en af programmets erklæringsdele.
• En statisk instans skabes implicit sammen med det omkringliggende objekt.
• En statisk instans kan erkendes, inden programmet udføres.

Dynamisk instantiering:
• En instantiering, som foreskrives i et af programmets handlingsdele.
• En dynamisk instans skal skabes eksplicit, typisk gennem en særlig kommando.
• En dynamisk instans kan først erkendes på udførelsestidspunktet af programmet.

Noter

I Eiffel er instanser af klasser som grundregel dynamisk instantierede, medens instanser af simple typer er statisk instantierede.

Klasse-typer kan dog godt instantieres statisk i Eiffel; hvis man ønsker at attributten 'a' bliver statisk instantieret skriver man

```
a: expanded C
```

og ikke blot

```
a: C.
```

Ovenstående er omvendt i forhold til Pascal og mange andre traditionelle programmeringssprog. I sådanne sprog er grundreglen statisk allokering af data, både hvad angår simple typer og sammensatte typer (såsom arrays og records).

Initialisering af objekter.

Forskellige muligheder for beskrivelse af initialisering:

- Klassen har en initialiseringsdel.
- Initialiseringen af objekter er beskrevet i tilknytning til variablene i klassen.
- En eller flere operationer i klassen er helliget initialisering.

Noter

Den først beskrevne initialiseringsteknik består i, at hver klasse har et stump initialiserings program tilknyttet som "kroppen af klassen". Denne stump kode udføres efter hver instantiering af klassen, bl.a. med det formål at initialisere det nye objekt. Simula og Beta er eksempler på sprog, der følger dette mønster.

Det er i nogle sprog muligt til enhver variabel i klasse at beskrive, hvilken initialværdi variable skal antage. Hvis dette ikke er en fast værdi, kan man i nogle sprog endvidere angive hvilken parameter under instantieringen, der skal bruges for at specificere en initial-værdi. CLOS (the Common Lisp Object System) er et sådant sprog.

Det mest almindelige er, at initialiseringen foretages ved brug af en initialiseringsoperation. I C++ er der såkaldte konstruktorer, som har samme navne som klassen. I Eiffel kan initialisering foretages af en såkalde *creation procedure* (som vi vender tilbage til i næste kapitel).

Operationer på objekter i forhold til konventionelle procedurekald.

Konventionel procedure-definition og -kald:

```
procedure P (x: C; y: D);      var a: C;
begin ... end                      b: D;
                               ...
                               P(a, b)
```

Operations-definition og -kald i et objekt-orienteret program:

```
class C                        var a: C;
                                   b: D;
  operation P (x: D)
  ...                          ...
end                            a.P(b);
```


Noter

I det objekt-orienterede tilfælde er det en forudsætning, at operationen P tilhører klassens interface. Med andre ord, P må ikke være en skjult, intern operation i klassen C.

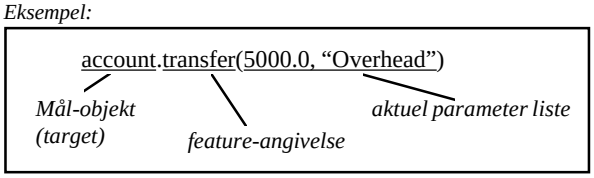
Operationskald i objekt-orienterede sprog.

- I et objekt-orienteret program anvendes enhver operation på et objekt, evt. med et antal andre objekter overført som parameter til operationen.
- Almindelige procedurekald (uden angivelse af hvilket objekt, der opereres på) forekommer ikke.
- Dot-notationen

  objekt.operation(aktuelle-parametre)

  er typisk notation for anvendelse af en operation med et antal parametre på et objekt.

Eksempel:



Noter

Når det siges, at almindelige procedurekald ikke forekommer i objekt-orienterede sprog, kan det forekomme som en sandhed med modifikationer.

I Eiffel, og også C++, forekommer der typisk mange procedurekald internt i operationer på en klasse. Sådanne procedurekald er imidlertid i Eiffel blot udtryk for en forkortelse. Meningen er, at procedurekaldet er udtryk for en operation *på det nuværende objekt.*

Der gives et eksempel på dette sidst i denne forelæsning, hvor vi ser på en Eiffel klassedefinition for en bankkonto.

Så almindelige procedurekald forekommer altså ikke i det ultimativt objekt-orienterede sprog.

Nederst vises et konkret eksempel på et operationskald, og der introduceres noget terminologi, som delvis knytter sig til Eiffel. En *feature* er i Eiffel en fællesbetegnelse for en en operation og en variabel/konstant i en klasse (mere om dette i næste kapitel).

Klasseroller: klienter og leverandører.

```
class C                    class D
                             x: C;
  data og operations
  erklæringer                op(p: C);

end;                         f(...): C
                             ....
                           end;
```

• Klassen D er en **klient (client)** af C.
"D køber fra C".

• Klassen C er en **leverandør** (**supplier**) til D.
"C sælger noget til D".

Noter

D's klientforhold til C kan skabes på fire forskellige måder:

1. Ved at D erklærer en variabel af typen C.
2. Ved at en operation i D har en parameter af typen C.
3. Ved at en funktion i D returnerer en værdi af typen C.
4. Ved at en operation i D erklærer en lokal variabel af typen C.

Kun de tre første tilfælde er illustreret på denne slide.

Man skal bemærke, at relationen, som den er indført på denne slide og i bogen "Object Oriented Software Construction", er en relation mellem to klasser. Relationen lader sig let overføre til en relation mellem et objekt og en klasse.

Et eksempel på en klasse i Eiffel.

```
class BANK_ACCOUNT
creation make

feature
  balance: real;
  interest_rate: real;
  owner: PERSON;

  make(b, ir: REAL; per: PERSON) is
  do
    balance := b;  interest_rate := ir;
    owner := per
  end;

  deposit(amount: real) is
  do
    add(amount)
  end;

 withdraw(amount: real) is
  do
    add(-amount)
  end;

  add_interest(number_of_days: integer) is
  do
    add(balance *
      (interest_rate/100) *
      (number_of_days / 360) )
  end;

  feature{NONE}

    add(amount: real) is
    do
      balance := balance + amount
    end

end -- class BANK_ACCOUNT
```

PS1 -- Basal objekt-orienteret programmering I

Noter

Klassen bank_account definerer tre klasse attributter (variable) og fire operationer. Under ét kaldes disse for features. Alle features pånær add er synlige i klienter af bank_account. De tre attributter (variable) kunne uden videre omdefineres til at være funktioner uden parametre, uden at det berører klienter af klassen.

Add proceduren er intern i klassen (kan ikke benyttes af klienter af bank_account). Dette ses derved at add findes i et afsnit af programmet, som er markeret med "feature{NONE}", hvilket vil sige at denne feature ikke eksporteres til nogen klasser.

Det er bemærkelsesværdigt, at add tilsyneladende kaldes som en konventionel procedure fra andre operationer i klassen. Meningen er imidlertid, at add kaldes på det nuværende objekt, som jo er en instans af bank_account. Objektet, der arbejdes på, er altså implicit angivet i dette tilfælde.

Klassen PERSON er ikke defineret på denne slide.

For at bringe bank_account til anvendelse, skal der laves en klientklasse, som opretter et antal bankkonti, og opererer på dem via operationerne i bank_account's klient interface.

Opgave: Udskift 'balancen' med et array af withdraw/deposit beløb. Ændringen af klassen skal foregå således, at *interfacet* af klassen bevares intakt og uforandret.

Karakteristika ved klassebegrebet i Eiffel.

- Eiffel er et objekt-orienteret programmeringssprog med statisk typing.
- Variable af klasse-typer allokeres som en grundregel dynamisk i Eiffel.
- Variable af simple typer (integer, boolean, real, character) allokeres statisk.
- Interfacet af en klasse bestemmes af en markering af grupper af definitioner i en Eiffel klasse. Operationer kan være interne i klassen; variable kan gøres til en del af interfacet.
- Variable, der eksportes, kan læses, men ikke modificeres.
- Hvis klassen B er en klient af klassen A, og hvis x er en variable af typen A i B, kan man ikke se, om udtrykket x.a refererer til en variable a eller til en funktion a i A.
- Alle variable, der beskriver data i en Eiffel klasse, initialiseres til faste, sprog-definerede værdier.
- Det er muligt at lave såkaldte creation procedurer, som initialiserer variable i klasser til program-definerede værdier.
- Klasser kan ikke indlejres i hinanden i Eiffel.

Noter

Denne slide beskriver på kort form hvordan programmeringssproget Eiffel er designet med hensyn til nogle af de emner, vi har diskuteret i dette kapitel. I den følgende forelæsning vil vi koncentrer os om detaljer i basal, objekt-orienteret Eiffel programmering.

Det er værd at hæfte sig ved, at variable, som eksporteres, kan læses udefra, men det er ikke muligt direkte at assigne til disse. Hvis variable i en klasse skal ændre værdi som følge af en ekstern begivenhed, skal det ske gennem operationsinterfacet.

Hvorfor mon denne begrænsning? Fuld skrivetilladelse udvikler sig let til anarki, idet klienter da vil kunne tilskrive dele af objektets repræsentation vilkårlige værdier. Dette vil kunne bryde klassens invariant. Vi vender tilbage til dette emne.

Det modsatte ekstrem tvinger alle variabelaflæsninger til at foregå gennem funktioner, hvilket i mange situationer kan forekomme tungt.

Terminologi

	Smalltalk	Eiffel	C++
Interface	Protokol		
		Feature	Member
Operation	Metode	Routine	Function member
Operationskald	Sende en besked		
Variabel	Instansvariabel	Attribut	Data member

Noter

I forskellige sprog og i forskellige objekt-orienterede subkulturer anvendes der forskellig terminologi for begreber, der enten er de samme, eller ligner hinanden meget. Denne slide giver en lille (og ikke særlig komplet) oversigt over beslægtet terminologi.