Arbeidsnotater fra Høgskolen i Buskerud nr. 69

ARBEIDSNOTAT

ARBEIDSNOTAT

Hendelsesorientert programmering med Visual Basic

Kompendium

Knut W. Hansson

HØGSKOLEN i Buskerud

Arbeidsnotater fra Høgskolen i Buskerud

Nr. 69

Hendelsesorientert programmering med Visual Basic

Kompendium

Av

Knut W. Hansson

Hønefoss 2012

ISSN 1893-2398 (online)

HØGSKOLEN
i Buskerud

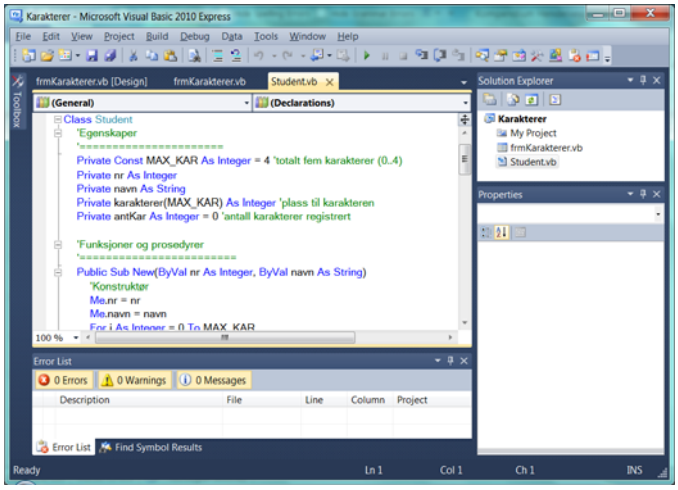
Hendelsesorientert programmering

med Visual Basic

Kompendium

Knut W. Hansson
Førstelektor IT

11 October 2012



Emneord:
Visual Basic
hendelsesorientert
programmering
kompendium

English keywords:
Visual Basic
event driven
programming
compendium

Sammendrag
Ved høyskolen i Buskerud, bachelorstudiene i IT, undervises kurset "Hendelsesorientert programmering" og gir 7,5 studiepoeng.

Kurset bygger på "Grunnleggende programmering" med ca. ni studiepoeng programmering i Visual Basic. I det underliggende kurset skal studentene ha blitt kjent med programmeringsmiljøet, kontrollers egenskaper og prosedyrer, datatyper og tilordninger. De skal kunne programmere de vanligste programstrukturene (sekvenser, seleksjoner og iterasjoner). De skal videre kunne håndtere arrays (opprette, sortere, finne største/minste, snitt og sum) og lage sine egne prosedyrer og funksjoner.

Kurset "Hendelsesorientert programmering" bygger altså på dette og skal bringe det videre med mer avanserte programstrukturer (rekursjon) og prosedyrer/funksjoner med argumenter både etter verdi og referanse og frivillige argumenter. De skal lære grafikk, dato/tid, filbehandling og programmere mot relasjonsdatabaser. De skal lære objektorientert programmering med egendefinerte klasser, kunne teste programmer og dokumentere dem, herunder lage interaktiv hjelp til brukeren.

Dette kompendiet inneholder alle forelesningene mine i kurset, samt noe tilleggsstoff for spesielt interesserte.

Det er laget en egen elektronisk lærerressurs med oppgavene og løsningsforslag.

Synopsis in English
At Buskerud College, the course "Event Driven Programming" is part of the bachelor IT education and gives 7.5 credits.

The course builds on an underlying course "Programming for beginners" with approx. nine credits. In the underlying course, the students should learn to know the IDE, controls' properties and procedures, data types and assignments. They should be able to program the most common program structures (sequence, selection and iteration). They should also be able to use arrays (create, sort, find largest/smallest value, average and sum) and make their own procedures and functions.

The course "Event Driven Programming" builds on this knowledge and aims to bring it further with more advanced program structures (recursion) and procedures/functions with arguments both by value and by reference and optional arguments. They shall learn graphics, date/time, files and program against relational databases. They shall learn object oriented programming with self-defined classes, be able to test programs and document them, including the making of interactive help for the user.

This compendium contains all my lectures in the course, and some additional subjects for interested students.

An electronic teacher's resource has been made with the assignments and suggested solution for the practical part of the course.

Innhold

Innledning

Eksamensordning/arbeidsmåter

Du bør lese emneplanen grundig, særlig vurderingsordningen. Merk deg spesielt at du ikke får gå opp til avsluttende gruppeeksamen hvis du ikke består ukeoppgavene. Den beste strategien – både for læringens skyld og vurderingen – er å levere *alle* ukeoppgavene. Du kan da velge å få vurdert den (av tre) som du mener at du klarte best.

Følgende vurderingsordning gjelder i dette faget:

> **Del 1:**
> Det gis inntil 12 individuelle ukeoppgaver, gruppert tre og tre. Studentene leverer én besvarelse i hver gruppe etter eget valg. Hver innlevering gis karakter etter gradert skala. Studenter som ikke leverer noen besvarelse i en gruppe innen angitt frist, gis karakteren F for denne gruppen.
> Gjennomsnittsvurderingen (der alle disse karakterene – inkl. evt. F – teller likt) inngår i den samlede karakter med 40 %. For å få delta ved eksamen, må studenten minst ha samlekarakteren E (dvs. bestått). Det er ikke et krav at alle besvarelsene skal være bestått.
>
> I tillegg kan faglæreren, fritt og uten begrunnelse, la enkelte studenter gå opp til en individuell prøve. Denne kan være skriftlig, muntlig, på datamaskin eller en kombinasjon av dette. For disse inngår den individuelle prøven i vurderingsgrunnlaget.
>
> **Del 2:**
> Videre gis et eksamenscase som løses i gruppe i løpet av en uke. Caset leveres skriftlig og følges opp av en muntlig, gruppevis presentasjon, med mulighet for individuelle spørsmål. Karakteren for eksamenscaset inngår i den endelige vurderingen med 60 %. Begge hovedkomponentene (vurdering gjennom studietiden og avsluttende vurdering) må være bestått.
>
> *Utdrag av emneplanen 2012/13*

Del 1 vurderes etter følgende regler:

1) Etter forelesningen legges en oppgave ut i kurset på Fronter, under ”Oppgaver”. Leveringsfrist er vanligvis kl 1600 dagen før neste forelesning. Da vises et løsningsforslag i Fronter.
2) Studentene kan levere sin besvarelse til vurdering via Fronter innen fristen. Forsinkede leveringer mottas ikke.
3) Dere bør skrive ut løsningsforslaget (vb-filene) og ta det med på forelesningen. Jeg har ofte ikke tid til å gjennomgå oppgaven, men svarer bare på konkrete spørsmål fra dem som har forsøkt å løse den.
4) Oppgavene grupperes tre og tre, og alle studenter forventes å levere minst én av de første tre oppgavene, minst én av de neste tre osv. *Det kan bli bare to oppgaver i noen grupper.* Studentene kan levere flere. Da må de – etter at de tre ukene er omme – oppgi hvilken ene de ønsker vurdert. Det blir fire slike grupper.
5) Oppgaver som skal leveres, må løses individuelt, og leverte besvarelser skal være studentens eget arbeid. Under løsningen er likevel ”alle hjelpemidler tillatt” og det er både lov til å søke råd og å samarbeide med andre studenter. Slikt samarbeid skal i så fall fremgå av leveringen. Kopiering av andre studenters kode er ikke tillatt (og heller ikke lærerikt). Be – og gi – heller generell hjelp med fenomenet. Altså ikke ”hvordan løste du dette her?”, men ”hvordan får man vanligvis testet om…?”
6) Faglæreren kan når som helst – og uten begrunnelse – innkalle enkeltstudenter til en muntlig høring om en levert besvarelse, og da inngår høring i vurderingsgrunnlaget.

7) En av de leverte oppgavene i hver gruppe blir vurdert og gis poeng, fra 1 = A til 6 = F. Av ressursgrunner kan studentene ikke forvente annen tilbakemelding enn dette. De som ikke leverer noen av de tre oppgavene i en gruppe, får karakteren 6 = F for denne gruppen.
8) Hver student vil på denne måten få fire karakterer gjennom semesteret. Disse teller likt og gjennomsnittet – avrundet til nærmeste hele etter vanlige avrundingsregler (3,5 gir 4 osv.) – inngår i fagets endelige karakter med 40 %.
9) Hvis gjennomsnittet er 5,5 eller mer (som gir karakteren F), har studenten ikke bestått del 1 og vil ikke få delta i eksamenscaset.

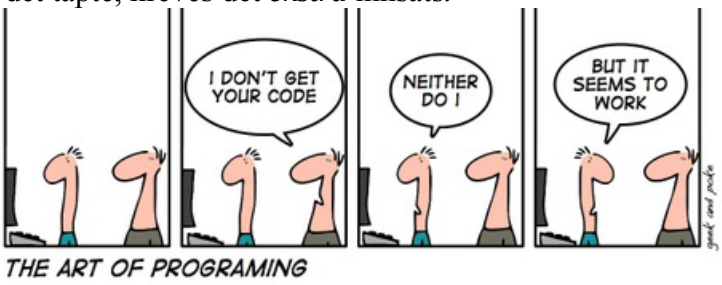
Del 2 – eksamenscaset – løses i grupper på inntil fem studenter, som alle må ha bestått del 1. Studentene setter selv sammen gruppene. Caset leveres ut gjennom Fronter og varer en ukes tid. Senere skal arbeidet presenteres for medstudenter, faglærer og sensor. Caset vil bli lagt sent i semesteret, og endelig beskjed om tidspunktet vil bli gitt senere.

Litteratur

I dette faget legges det ut omfattende forelesningsnotater i Fronter og alle oppgaver gis der. Det er derfor ikke stengt nødvendig med lærebok for å følge dette faget. Det anbefales derfor å ikke kjøpe ny litteratur, men beholde den anbefalte litteraturen fra INF150 Grunnleggende programmering.

"Sliter" du med programmering?

Det er mange som forteller at de sliter med programmering, og det er ikke uvanlig. Resultatene fra programmeringsdelen til jul tyder på at det er mange som "sliter" - hvis nå det kan være en trøst. Hvis man skal ta igjen det tapte, kreves det *ekstra* innsats.



Ikke all kode er god kode!

Dere er "lærlinger i programmering"

Foreløpig har der bare fått et lite innblikk i emnet programmering, med de aller enkleste teknikkene. Dere er fortsatt *nesten* nybegynnere, eller altså *lærlinger*. Lærlinger er yngre håndverkere som arbeider hos en mester (ja - foreleseren din, altså :-) for å lære. Lærlinger lærer ved å jobbe sammen med en mester. Læringen skjer

1. Ved å lytte til mesteren som forklarer og forteller. Mesterens erfaring er gull verd.
   Mesteren forklarer og forteller i forelesning hver uke. Vær med og følg aktivt med. Gjør notater.
2. Ved å prøve enkle oppgaver som mesteren gir. Mesteren gir oppgaver tilpasset lærlingens nivå. Etterhvert blir oppgavene vanskeligere og til slutt kan lærlingen ta svenneprøve og bli godkjent håndverker. Det nytter ikke for en lærling å bare se på mesteren - lærlingen må prøve selv. Bare da får lærlingen en forståelse av vanskelighetene og trening i å løse dem.
   Dere får oppgaver hver uke. Gjennom å arbeide med dem, lærer dere mye - både kunnskaper, teknikker og ferdigheter. For å få noe ut av gjennomgang, må man ha forsøkt seriøst selv, så man vet akkurat hvor man står fast, hva som var vanskelig og hva som gikk lett. Dere tar svenneprøve som et eksamenscase.
3. Ved å se på mesteren. Mesteren gjør selv vanskeligere oppgaver som lærlingen - og delvis også svennen - kan lære av å se på.
   Dere får se mesteren i arbeid under forelesningene, når han viser eksempler og gjennomgår oppgaver. Dere kan se mesterens ferdige arbeider som løsningsforslag. Studer løsningsforslagene og se etter ting å kopiere.

Programmering er sentralt

Programmering er sentralt i ethvert IT-studium. Én ting er å tegne modeller, tegne opp skjemaer ("forms"), skrive SQL og lage regnearkmodeller. Det er statiske ting - ting som "er" slik eller slik, og de "er" det hele tiden. Det er ikke altfor vanskelig. Det er verre å tenke dynamisk slik man må når man programmerer. IT dreier seg om begge deler, og programmering dukker opp i en eller annen form, gang på gang. Frem til IT bachelor skal dere f.eks. programmere VB (Windows), Java (alle operativsystemer), PHP/ASP (nettsider for Internett), PL/SQL (prosedyrer i databasen Oracle), C#, Flash (multimedia for Internett) og annet. Alle disse har mye felles, særlig i tankegang. Har du først mestret programmering i VB, blir det ikke altfor vanskelig å lære de andre programmeringsspråkene. Det blir litt som å sykle: Har du først lært å sykle på én type sykkel, er det ikke så vanskelig å skifte til en annen sykkeltype (andre girspaker, annet styre, andre bremser).

Skisse til en ukeplan

Alt studiearbeid bør gjøres på skolen. Det er mer motiverende å delta i et miljø, og du har andre å spørre, samt mulighet for å få veiledning av læreren. Arbeid hjemme bare som overtid kveld/week end. Regn med å arbeide tre økter med faget:

Økt 1 - Forelesningsdagen:
Møt til forelesning og vær aktiv der. Notér og spør. Bruk blyant og papir. Her gjennomgås det meste av det nye som du trenger til neste oppgave.

Økt 2 - Samme dag eller en annen dag (bestem en fast dag):
(1) Start med å se på løsningsforslaget fra forrige gang. Merk deg tekniker og "knep" som du kan bruke selv. Tenk også på om din innlevering er én du vil ha vurdert (få karakter på).
(2) Se deretter igjennom oppgaven til neste gang, lag skjemaet (det er jo raskt gjort) og tenk litt på hvordan oppgaven kan løses. Tegn gjerne en figur for strukturer som arrays, lister, fil, objekter o.l. Bruk et par timer på dette.

Økt 3 – En anen dag (velg fast dag):
Programmer oppgaven ferdig. Bli enig med andre om et passende sted på skolen og gjør oppgaven der. Søk veiledning ved behov. Ikke glem å levere besvarelsen din.

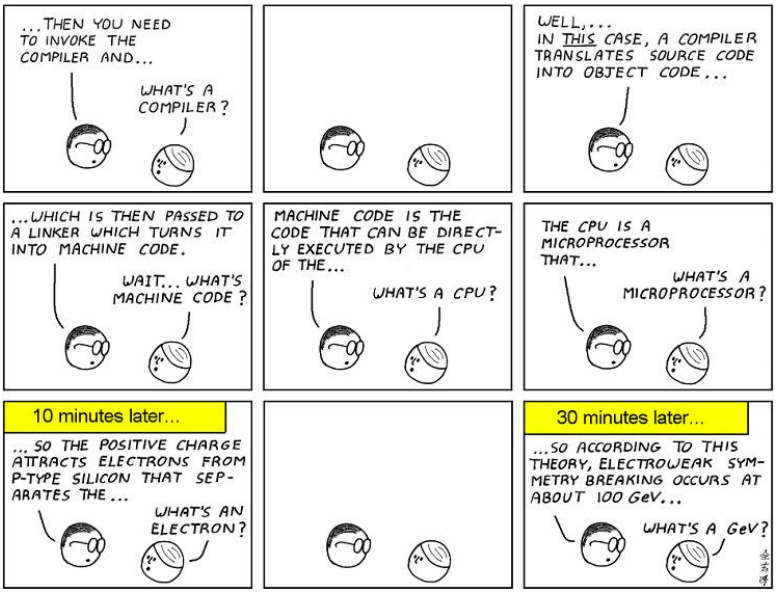
Noen mer konkrete råd

- ✓ Man lærer programmering først når man programmerer selv. Man må gjøre alt man får av oppgaver, og gjerne finne på flere selv i tillegg.
- ✓ Hvis man oppdager at man sliter med noe som man har hatt før, f.eks. funksjoner/prosedyrer, variable eller arrays ("tabeller"), kan man gå tilbake til notater, oppgaver og løsningsforslag som man har fått før, og studere dem om igjen.
- ✓ Det er også en god idé å søke på Internett - det finnes f.eks. et flott nettsted om VB[1] som du med fordel kan legge til som bokmerke i nettleseren din.
- ✓ Sjekk ut andre lærebøker. Se f.eks. på "Sam's Teach Yourself Visual Basic 2010" eller Visual Basic 2008 for Dummies (den kan innlånes gjennom biblioteket). Et *Internett-kurs* som ser bra ut er Home and Learn's kurser[2].
- ✓ Samarbeid med andre: Dann en gruppe som møtes hver uke og diskuterer (a) nytt stoff og (b) oppgaven til neste gang (c) annet som en i gruppen ikke forstår
- ✓ Spør aktivt i timen. Du kan trygt regne med at det er mange andre som lurer på det samme som deg.

Forkunnskaper

Studenter som ikke klarte "Grunnleggende programmering" får erfaringsmessig store problemer i dette kurset også og det kan ikke uten videre anbefales dem å ta dette kurset. Det har imidlertid også vist seg at enkelte *har* klart det med meget stor motivasjon og ikke minst arbeidsinnsats de første fem-seks ukene.

[1] http://www.dotnetperls.com/vb
[2] http://www.homeandlearn.co.uk/NET/vbNET.html



Dette bør dere minst kunne fra INF150 "Hendelsesorientert programmering":
- ✓ Beherske programmeringsmiljøet i VB
- ✓ Lage, vise og skjule skjemaer og diverse skjemamanipulering (f.eks. tømme felt, sett fokus)
- ✓ Datadeklarasjoner (Dim og Const)
- ✓ Tilordninger
- ✓ Operatorer (regne-, streng- og sammenlikningsoperatorer samt logiske operatorer)
- ✓ Meldingsbokser
- ✓ Programstrukturer: Sekvens, seleksjon (valg) og iterasjon (løkker)
- ✓ Prosedyrer og funksjoner
- ✓ Arrays
- ✓ Noen av de mange forhåndsdefinerte funksjoner i VB, f.eks. konvertering, strenger og tilfeldige tall
- ✓ Forhåndsdefinerte konstanter i VB, f.eks. True/False, vbYesNo og vbNewLine
- ✓ Standard programmeringsteknikker (f.eks. posit-admit, akkumulering, sortering)

Nesten alle disse punktene skal vi utvide og fordype i faget INF116. Det kan være mindre forskjeller i syntaks, men ellers vil de fleste finnes igjen i alle objektorienterte programmeringsspråk. Vi oppfatter stort sett VB som et eksempelspråk.

Dere bør også ha klar forståelse for noen viktige begreper som anvendes i dette faget:
- ✓ Mengde (definisjon) og mengdeoperasjoner (snitt, union, negering og kartesisk produkt)
- ✓ Funksjoner (definisjon, determinand = fri variabel, funksjonelt avhengige = avhengig variabel, definisjonsmengde og løsningsmengde, funksjoner med flere, frie variable)
- ✓ Konstanter og variable
- ✓ Tallsystemer
- ✓ Tegn- og fargekoder
- ✓ Tabeller og arrays
- ✓ Logikk
- ✓ Enkel trigonometri (sinus-funksjonen og radianer)
- ✓ Rekker og rekursjon
- ✓ Pseudotilfeldige tall (Rnd-funksjonen)
- ✓ Objekter

Hittil har dere i faget INF150 "Hendelsesorientert programmering" – laget applikasjoner som…
- ✓ brukes av en, enkelt bruker av gangen (= énbruker applikasjoner)
- ✓ kjører på en, enkelt maskin (= "stand alone" applikasjoner)

- ✓ mister all input når programmet avsluttes (transiente data)
- ✓ bruker statiske, grafiske brukergrensesnitt (= enkel GUI)
- ✓ er uten dynamisk grafikk og uten multimedia
- ✓ har separate data og funksjonalitet (= imperativt språk)
- ✓ har enkle programstrukturer (sekvens, seleksjon og iterasjon)
- ✓ har enkle datastrukturer (enkle datatyper og arrays)
- ✓ har brukergrensesnitt som er basert på skjemaer deklarert i programmet
- ✓ er lite dokumentert utover programkoden
- ✓ er lite testet/kvalitetssikret
- ✓ bruker noen objekter, men de er forhåndsdeklarert

Etterhvert – i løpet av bachelorstudiet – skal dere utvide dette. Noe kommer i INF116, annet kommer senere.

Litt repetisjon av tidligere kurs

I første forelesning repeteres litt sentralt stoff fra det underliggende kurset "Grunnleggende programmering" som anses kjent. Det kurset du nå tar, skal bygge på disse kunnskapene. Repetisjonen skal gi deg mulighet til å friske opp, evt. hente deg inn.

Variable og konstanter

Variable er *verdier* med en angitt **datatype** og et **navn**. Verdien kan variere mens programmet kjører. Variable må *deklareres*, dvs. man må gjøre det klart at man vil bruke den. De får da samtidig en verdi:

```
Dim tekst As String
```

Her tekst en variabel. Siden det ikke er angitt noen verdi for variabelen, blir den *initiert* av kompilatoren (programmet som gjør om *kildekode* til *maskinkode*). Tekster initieres til en tom streng, tall initieres til 0 (null) og Boolske variable til *false*. Dette har å gjøre med at det settes av plass til variabelen i RAM, og denne delen av minnet har jo en eller annen verdi uansett. Det er en fordel at vi vet hvilken verdi det er.

Variable kan *tilordnes, dvs. gis, en verdi* av riktig datatype:

```
tekst = "Heisann!"
MsgBox(tekst)
```

Variable kan også initieres av programmereren:

```
Dim tekst As String = "Hei, Knut!"
```

Allikevel kan den tilordnes verdi senere som vist i forrige eksempel.

I uttrykket

```
MsgBox("Hei, Knut!")
```

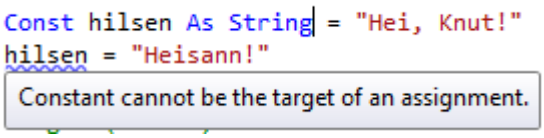
er "Hei, Knut!" en **konstant** – den kan altså ikke endres mens programmet kjører. Vi kan gi konstanten et *navn*, ved å skrive en deklarasjon:

```
Const hilsen As String = "Hei, Knut!"
```

Navnet på konstanten er da blitt *hilsen* og verdien er initiert til "Hei, Knut!". **Navngitte konstanter** må gis både *datatype* og *verdi* samtidig med at den deklareres. Når vi først har deklarert en slik konstant, kan vi bruke den der det skal være en String:

```
MsgBox(hilsen)
lblHilsen.Text = hilsen
```

Senere kan en navngitt konstant ikke tilordnes ny verdi, og kompilatoren vil gi en feilmelding hvis man prøver:



Variable og navngitte konstanter må deklareres. Det vanligste er å deklarere variable med *Dim* og navngitte konstanter med *Const*. Deklarasjonen *må* ha et navn og en datatype, slik det er vist ovenfor. I tillegg kan man angi **synlighet** som angir i hvilken grad variabelen/konstanten kan "sees" fra andre steder i programmet. Det kommer jeg tilbake til i en senere forelesning.

Variable og navngitte konstanter har også en **levetid**. Det settes av plass til dem i RAM mens programmet kjører, og denne plassen fristilles igjen mens programmet kjører. Vi kaller det gjerne at variabelen "opprettes" og "fjernes". Hvor lenge variabelen/konstanten eksisterer ("lever") er avhengig av *hvor* de deklareres.

Variable og konstanter kan deklareres forskjellige steder i programmet. Et viktig prinsipp i programmering, er å deklarere dem der de skal brukes, og ikke utenfor. En annen måte å si det på, er at variable skal ha kortest mulig ”levetid”.

Når mange programmerer sammen, noe som er vanlig for større applikasjoner, oppnår vi da at vi i størst mulig grad bruker variable som vi selv har deklarert, og deklarasjonen står så nær som mulig koden som bruker den. Her er en skisse av et program ("Applikasjonen") med flere skjemaer. Variable er angitt med fet skrift.

Applikasjonen

Public Class **skjema1** (for brukere)
*Dim **logginn** As String*
*Private Sub **lagre** (ByVal **fil** as String)*
*Dim **svar** as String*
*For **i** As Integer = 0 To 5*
…
Next i
…
End Sub
…

Public Class **skjema2** (for admin)
*Dim **antall** As Integer*
…
*Private Sub **nyStudent()***
*Dim **studnvn** As String*
….
End Sub
…

Synligheten gjelder "sideveis" og "innover" (men *ikke* "utover") som følger:

- ✓ *skjema1* er synlig overalt i applikasjonen, da den er deklarert *Public*
- ✓ *loggin* er synlig overalt i *skjema1*. Den er ikke synlig fra *skjema2*.
- ✓ Prosedyren *lagre* er synlig overalt i *skjema1* og ikke synlig fra *skjema2*.
- ✓ *fil* i prosedyren *lagre* er bare synlig innenfor denne prosedyren.
- ✓ for-løkkens *i* er bare synlig inne i løkken.

Prøv selv å angi synlighet for navnene i *skjema2*?

Om *levetiden* kan man si at variabelen/konstanten lever innenfor en **blokk**. En blokk er en gruppe kodelinjer avsluttet med *End*, *Next*, *Loop*, *Else* eller *End If*. Når det er slutt på levetiden, eksisterer ikke variabelen/konstanten lenger, og er da selvsagt heller ikke synlig. Siden synligheten varierer, kan en variabel/konstant godt eksistere ("leve") selv om den ikke er synlig.

Kontroller

Kontroller er ferdige elementer som vi legger inn i skjemaer. De har **egenskaper** inkludert et *navn*, **metoder** (funksjoner og prosedyrer) og de kan bli utsatt for **hendelser**.

Vi får tilgang til kontrollens *egenskaper* og *metoder* med **punktnotasjon**. Vi angir kontrollens navn etterfulgt av punktum og navnet på egenskapen eller metoder, f.eks.

```
txtNavn.Text = hilsen
lblHilsen.Visible = False
txtNavn.Clear()
```

1. Kontrollers egenskaper.

Egenskapene er egentlig variable/konstanter som er deklarert for kontrollene, akkurat som skjemaer[3]. De som er synlige for oss er deklarert på en måte som gjør dem tilgjengelige.

Verdien av en synlig *egenskap* – både variable og konstanter – kan endres av oss under design. Typiske eksempler på synlige variable egenskaper er bakgrunnsfarge, tekst, font og størrelse/plassering. Det kan være flere variable egenskaper som ikke synes.

[3] Skjemaer er egentlig også en type kontroll. Det skapes en slik skjemakontroll når programmet kjøres. De andre kontrollene skapes i design, når vi drar en kontroll inn på skjemaet. Da kan vi ikke legge til noe, men må ta det som er gitt av Microsofts programmerere.

Synlige *konstante egenskaper* får vi ikke lov til å endre mens programmet kjører – f.eks. kontrollens navn og en teksts lengde. Hvis de er synlig, kan vi imidlertid lese dem av.

Alle kontrollens egenskaper har verdi uansett – det settes av plass til verdiene i RAM når kontrollen skapes. Under design kan vi endre disse verdiene som da blir verdiene de får når programmet starter. De variable kan vi også endre mens programmet går.

2. Kontrollers metoder

Metodene er ting som kontrollen kan *gjøre* eller *beregne*. Dere har f.eks. sikkert brukt metoden *Show()* for et skjema, *Fokus()* for forskjellige kontroller og *Clear()* for tekstbokser. Kontroller har også funksjonen *ToString()* som gjør om kontrollen til en tekst (den er ikke særlig nyttig). Metodene angis med punktnotasjon, som variable og konstanter.

Vanlige kontrollers metoder er forhåndsdefinerte og det er angitt i manualen hva de gjør. Vi kan ikke endre dem. I skjemaer, derimot, får vi tilgang til selv deklarasjonen, og kan selv legge til egenskaper (variable) og metoder. Vi bestemmer da selv synlighet (mer om dette senere).

3. Kontrollers hendelser

Hendelsene får vi ikke tilgang til. De er automatiske, men navngitt med punktnotasjon. Når operativsystemet oppdager en hendelse, f.eks. at det ble klikket med musen, finner den ut hva markøren pekte på akkurat da. Hvis markøren pekte på f.eks. en knapp *butHils* i vårt skjema, vil hendelsen bli sendt til vårt program sammen med en del opplysninger om hendelsen. Vårt skjema får beskjeden og leter etter en hendelsesprosedyre som håndterer denne hendelsen *klikk* for *butHils*. Anta at vi har skrevet følgende hendelsesprosedyre:

```
Private Sub butHils_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) _
                Handles butHils.Click
        MsgBox(hilsen)
End Sub
```

Da har vi laget en hendelsesprosedyre som skal håndtere hendelsen *butHils.Click* (legg merke til punktnotasjon også for hendelser). Den vil bli utført og en meldingsboks vises.

Prosedyrer og funksjoner

En *prosedyre* er en navngitt gruppe programsetninger som *gjør noe*. En spesiell type er *hendelsesprosedyrer* (eksempel ovenfor). De skal gjøre noe når det har skjedd en bestemt hendelse. De andre prosedyrene gjør noe når de blir "kallet" fra et annet sted i programmet (da må de være synlige derfra).

En *funksjon* er… Ja, hva er det nå igjen[4]?

I programmet er en funksjon en navngitt gruppe programsetninger som returnerer en verdi av en bestemt type. Programsetningene utgjør den regelen som det er snakk om i definisjonen. Verdiene som oppgis når funksjonen kalles, er determinanden og må være av riktig type (les: riktig mengde). Verdien som returneres er den funksjonelt avhengige og kan være av samme eller en annen type (les: mengde). Kompilatoren vil passe på at det alltid returneres en verdi av riktig type fra en funksjon. Funksjoner tilordner en verdi bare når de "kalles".

[4] Måtte du titte? "En funksjon er en regel, som til hvert element i en mengde A tilordner ett – og bare ett – element i en mengde B. Elementene i A er determinander, elementene i B er funksjonelt avhengig av elementene i A." A og B kan være samme mengde. A kan være en kombinasjon av elementer fra flere mengder (funksjoner av flere variable).

Arrays

Ofte er det bruk for flere variable av samme type og de skal gjerne ha samme navn. Istedenfor å deklarere dem hver for seg med nesten like navn, f.eks.

```
Dim navn0 As String = "Knut"
Dim navn1 As String = "Erik"
Dim navn2 As String = "Petra"
Dim navn3 As String = "Olga"
```

Når det er mange navn, er dette tungvint, og det er enklere å deklarere en **array**:

```
Dim navn(3) As String
```

Da angir vi hvilket variabel vi mener med en *indeks*, f.eks.

```
navn(3) = "Petra"
navn(4) = "Olga"
```

Den store fordelen oppnår vi hvis alle navnene skal behandles likt i en løkke, f.eks. når navnene leses fra en fil (det første navnet som leses tilordnes til *navn(0)*, det neste til *navn(1)* osv.). Hvis alle navnene skal legges til listeboksen *lstNavn* kan det se slik ut med enkle variable:

```
lstNavn.Items.Add(navn0)
lstNavn.Items.Add(navn1)
lstNavn.Items.Add(navn2)
lstNavn.Items.Add(navn3)
```

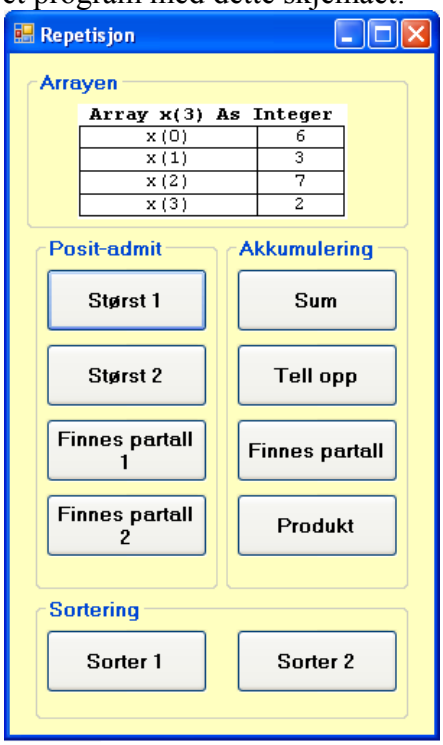
eller slik med array:

```
For i As Integer = 0 To 3
      lstNavn.Items.Add(navn(i))
Next
```

De fleste programmerere vil nok foretrekke den siste, særlig med flere like variable enn bare fire. Det kan jo fort bli tusener av navn som leses fra en fil.

Noen standard programmeringsteknikker

I eksemplene nedenfor bruker vi et program med dette skjemaet:



Programmet har deklarert en array, slik.

```
Private x(3) As Integer 'fylles med verdier i Load
```

og den fylles med følgende verdier:

Array x(3) As Integer

x(0)	6
x(1)	3
x(2)	7
x(3)	2

A. Posit-Admit
Posit = Anta, Admit = Innrømme

Teknikken brukes når man skal sjekke mange data og komme ut med ett svar, f.eks. Hvilken verdi er størst/minst? eller: Er alt i orden med denne?

Eksempel A1
Vi skal finne den største verdien i arrayen. Posit-admit-teknikken tilsier da at vi skal **anta** at den første av verdiene også er den største. Deretter går vi igjennom resten, og hvis vi finner en som er større så **innrømmer** vi tok feil. Det er to måter å gjøre det på – her vises den første, der vi "husker" den største *verdien*:

```
Private Sub butPositAdmitStørst1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butPositAdmitStørst1.Click
    '= Finner største verdi i en array

    'Metode 1 - Husk den største verdien
    Dim størst As Integer = x(0) 'Posit at element nr 0 er størst
    For i As Integer = 1 To 3 'sjekk resten av verdiene
        If x(i) > størst Then størst = x(i) 'Admit - vi fant en større verdi
    Next
    MsgBox("Den største verdien er " & størst.ToString())
End Sub
```

Hvis vi skal finne den minste verdien, er det bare å bytte ulikhetstegnet.

Eksempel A2
Vi kan oppnå akkurat det samme ved å "huske" indeksen til den verdien som er størst (istedenfor å huske verdien):

```
Private Sub butPositAdmitStørst2_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butPositAdmitStørst2.Click
    'Bruker posit-admit til å finne største verdi i en array
    'Metode 2 - Husk indeksen for den største verdien
    Dim størst As Integer = 0 'Posit at element nr 0 har størst verdi
    For i As Integer = 1 To 3 'sjekk resten
        If x(i) > x(størst) Then størst = i 'Admit - vi fant en større verdi
    Next
    MsgBox("Den største verdien er " & x(størst).ToString() _
           & " - det var verdi x(" & størst.ToString() & ")")
End Sub
```

Som du ser, har denne metoden en fordel, ved at den "husker" indeksen. Da kan vi senere oppgi både den største verdien og hvor den var plassert i arrayen.

Hvis det er flere verdier som er like store som den største (f.eks. hvis x(1) settes til 7 i arrayen ovenfor) – vil da programmet finne den første av dem eller den siste av dem? Hva må gjøres hvis vi vil ha det motsatte?

Eksempel A3
Vi vil sjekke om det er minst ett tall i arrayen som er partall, altså delelig med 2 (vi sjekker om x(i) Mod 2 = 0). Slik kan det se ut hvis vi bruker en tekst som variabel:

```
Private Sub butPositAdmitPartall1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butPositAdmitPartall1.Click
    Dim tekst As String = "Det finnes ingen partall" 'Posit
    For i As Integer = 0 To 3 'sjekk alle verdiene
        If x(i) Mod 2 = 0 Then tekst = "Det finnes minst ett partall" 'Admit
    Next
    MsgBox(tekst)
End Sub
```

Eksempel A4
Vi vil igjen sjekke om det er minst et tall i arrayen som er partall. Slik kan det se ut hvis vi bruker en Boolsk variabel:

```
Private Sub butPositAdmitPartall2_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butPositAdmitPartall.Click
    '= Sjekker om det er minst ett partall i arrayen
    Dim finnes As Boolean = False 'Posit - det er ingen partall der
    For i As Integer = 0 To 3 'sjekk alle
        If x(i) Mod 2 = 0 Then finnes = True 'Admit - vi fant et partall
    Next
    MsgBox("Påstanden om at partall finnes er " & finnes.ToString())
End Sub
```

B. Akkumulering
Akkumulering = Oppsamling

Teknikken brukes når vi skal **samle opp** noe. Det kan være en sum som skal samles opp, eller en tekst som skal vokse etter hvert. Vi starter da med en startverdi – en **initialverdi** – og så **akkumulerer** vi senere, dvs. vi føyer til mer.

Eksempel B1
Vi skal finne summen av alle verdiene i arrayen. Da må vi **initiere** en sum til 0, og så **akkumulere** etter hvert som vi går igjennom verdiene. Ved hoderegning kan vi se at

- ✓ starter med 0
- ✓ legger sammen 0 og 6 og får 6
- ✓ legger sammen 6 og 3 og får 9
- ✓ legger sammen 9 og 7 og får 16
- ✓ legger sammen 16 og 2 og får 18
- ✓ ferdig

Vi ber maskinen om å arbeide på akkurat samme måten:

```
Private Sub butAkkumulerSum_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAkkumulerSum.Click
    '= Bruker akkumulering til å beregne summen av alle verdien i arrayen
    Dim sum As Integer = 0 'initialverdi = vi starter med summen = 0
    For i As Integer = 0 To 3 'gå igjennom alle verdien
        sum += x(i) 'akkumuler = legg hver, enkelt verdi til det vi allerede har
    Next
    MsgBox("Summen er " & sum.ToString())
End Sub
```

Eksempel B2
Vi ønsker å telle opp hvor mange partall arrayen inneholder. Vi **initierer** antallet til 0, og **akkumulerer** ved å øke med 1 hver gang vi finner et partall:

```
Private Sub butAkkumulerTellopp_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAkkumulerTellopp.Click
    'Bruker akkumulering til å telle opp antall partall i arrayen
    Dim antall As Integer = 0 'initiering
    For i As Integer = 0 To 3 'sjekk alle verdiene
        If x(i) Mod 2 = 0 Then antall += 1 'akkumuler antallet
    Next
    MsgBox("Antallet partall er " & antall.ToString())
End Sub
```

Eksempel B3
Vi ønsker å se om det er partall i arrayen. Det kan vi jo se av forrige eksempel, siden antallet må bli minst 1 hvis det finnes partall. Her skal vi imidlertid bruke en akkumulering med en Boolsk variabel:

```
Private Sub butAkkumulerPartall_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAkkumulerPartall.Click
    '= Bruker akkumulering til å sjekke om arrayen inneholder partall
    Dim finnes As Boolean = False 'initialverdi
    For i As Integer = 0 To 3 'sjekker alle verdiene
        finnes = finnes Or (x(i) Mod 2 = 0) 'akkumuler med OR
    Next
    MsgBox("Påstanden om at partall finnes er " & finnes.ToString())
End Sub
```

Hvordan skulle vi sjekke om *alle* verdiene er partall?

Eksempel B4
Vi skal beregne produktet av alle verdiene i arrayen, altså x(0)*x(1)*x(2)*x(3). Her er det jo så få at vi kan skrive nettopp slik – i én setning – men hvordan skulle vi da gjøre det hvis det var 1000 verdier i arrayen. Vi gjør det i alle fall slik her:

```
Private Sub butAkkumulerProdukt_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAkkumulerProdukt.Click
    'Bruker akkumulering til å finne produktet av alle verdiene
    Dim produkt As Integer = 1 'initiering (hvorfor ikke 0?)
    For i As Integer = 0 To 3 'gå igjennom alle verdiene
        produkt *= x(i)
    Next
    MsgBox("Produktet av alle verdiene er " & produkt.ToString())
End Sub
```

C. Sortering av array
Når en array skal sorteres, kan det gjøres ved å flytte alle dataene fra en array til en annen. Man finner den minste i arrayen og flytter til plass 0 i den andre. Deretter den minste av de som er igjen til plass 1 i den andre osv.

Ulempen er at man må ha plass til begge arrayene i RAM, og det skaper problemer med store arrays, f.eks. ved sortering av filer som er for store for RAM.

Vanligvis vil man følgelig sortere innenfor arrayen. Da kreves det at to elementer i arrayen bytter plass. Slik kan de se ut når man vil bytte om element "start" med element "minst":

```
'bytt to elementer
Dim tmp As Integer
tmp = x(start)
x(start) = x(minst)
x(minst) = tmp
```

Det finnes svært mange algoritmer for sortering av arrays. De har forskjellige egenskaper, og brukes derfor i forskjellige situasjoner, f.eks. avhengig av hvor stor arrayen er, om den delvis er sortert på forhånd, om det er viktig å opprettholde rekkefølgen på poster med samme verdi i sorteringsnøkkelen osv. På høyskoler er noen spesielt populære, fordi de er enkle å huske, spesielt interessante eller enkle å forstå. Vi skal her se på én av de mest populære.

Algoritmen er slik at man først finner det minste elementet i arrayen. Det gjøres med standardteknikken posit-admit. Så bytter man element 0 med det minste (som derved er på rett plass). Deretter finner man det minste fra element 1 og utover, og bytter det med element 1 (og dermed er det kommet på rett plass). Man fortsetter med arrayen fra element 2 og utover osv. Når det bare er ett element igjen – det siste – så er det nødvendigvis på rett plass (alle de foranstående er jo det). Slik kan det se ut:

Eksempel C1

```
Private Sub butSort1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butSort1.Click
    '= Sorterer arrayen x stigende. Ingen bruk av egen prosedyrer/funksjoner
    For start As Integer = 0 To 3 - 1
        Dim minst As Integer = start 'Posit
        For i As Integer = start +1 To 3
            If x(i) < x(minst) Then minst = i 'Admit - de to skal byttes om
        Next
        'bytt første element (start) med minste (minst)
        Dim tmp As Integer
        tmp = x(start)
        x(start) = x(minst)
        x(minst) = tmp
    Next
    'Vis resultatet ved akkumulering
    picArray2.Visible = True
    'Skaper tekst til meldingsboksen ved akkumulering
    Dim tekst As String = "Ferdig sortert: " & vbNewLine 'init
    For i As Integer = 0 To 3
        tekst &= x(i) & vbNewLine 'akkumuler
    Next
        MsgBox(tekst)
End Sub
```

Legg merke til at både posit-admit og bytting av to elementer er med her. Det er ganske vanlig å sette sammen flere programmeringsteknikker til én større.

D. Prosedyrer og funksjoner

Vi bruker prosedyrer (*sub*) og funksjoner (*function*) for å dele opp programmet. Da blir det letter å overblikke, letter å forstå, lettere å teste og lettere å programmere/vedlikeholde. Dette kommer jeg grundig tilbake til i tema C om et par uker, så her minner jeg bare om enkelte ting som en ”oppvarming”.

Vi tar utgangspunkt i sorteringseksemplet ovenfor. En forenkling vil være å la en prosedyre bytte om de to verdiene når det trengs, og en funksjon bygge opp den teksten som skal vises i meldingsboksen. Da kan eksempel C1 se slik ut:

Eksempel D1

```
Private Sub butSorter2_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butSorter2.Click
    '= Sorterer arrayen x stigende. Bruker byttOm() og lagTekst()
       For start As Integer = 0 To 3 - 1
           Dim minst As Integer = start 'Posit
           For i As Integer = start To 3
               If x(i) < x(minst) Then minst = i 'Admit - de to skal byttes om
           Next
           byttOm(x(start), x(minst)) 'bytt to elementer
       Next
    'Vis resultatet
    picArray2.Visible = True
    MsgBox(lagTekst(x))
End Sub
```

Legg merke til at vi nå slipper mentalt å bli ”heftet” med ombyttingen, slik at vi kan konsentrere oss om *når* det skal skje. Videre lager teksten til meldingsboksen nærmest ”seg selv”. Sorteringsprosedyren er blitt kortere og mer oversiktlig, men vi må til gjengjeld lage *byttOm()* og *lagTekst()*.

Prosedyren byttOm()

Vi lager nå en prosedyre for bytting av to elementer. Den må få ”vite” hvilke to elementer som skal byttes – vi kaller dem a og b – og de må være heltall:

```
Private Sub byttOm(ByRef a As Integer, ByRef b As Integer)
    'Bytter elementene a og b
    Dim tmp As Integer
    tmp = a
    a = b
    b = tmp
End Sub
```

Legg her spesielt merke til

- ✓ Vi trenger ikke å vite hva de to verdiene a og b egentlig heter, i denne prosedyren kaller vi dem a og b (en form for alias-navn). Dette kalles ”formelt parameter”. Den som skal bruke denne prosedyren til å få byttet om på to tall, oppgir navnet på dem slik ”han/hun kjenner dem”, f.eks. `byttOm(x(i), x(minst))`. Da kalles de ”aktuelt argument”.
- ✓ Parametrene overføres *ByRef*. Da får prosedyren vite *hvor i RAM* de aktuelle argumentene er lagret. Prosedyren bruker samme sted og når verdiene da byttes om, får det effekt etterpå. Hvis man bruker *ByVal* får prosedyren bare oppgitt *verdien* og oppretter selv et sted i RAM der verdiene legges. Alle endringer av verdiene er da lokale i prosedyren, og disse variable fjernes fra RAM når prosedyren avslutter.

Funksjonen lagTekst()

Vi lager nå en funksjon som genererer den teksten vi skal vise i meldingsboksen. Den må få overført den arrayen som inneholder alle tallene som skal inn i meldingsboksen. Jeg kaller den *a*:

```
Private Function lagTekst(ByVal a() As Integer) As String
    'Skaper tekst ved akkumulering
    Dim tekst As String = "Ferdig sortert: " & vbNewLine 'init
    For i As Integer = 0 To 3
        tekst &= a(i) & vbNewLine 'akkumuler
    Next
    Return tekst
End Function
```

Legg her merke til
- ✓ Det formelle parameteret heter *a* lokalt for funksjonen.
- ✓ Parentesene bak *a* indikerer at *a* er en array – vi sier ikke hvor stor den er (det vil vise seg).
- ✓ Parameteret er overført med *ByVal* da vi ikke skal endre verdiene i arrayen.
- ✓ Funksjonen må returnere en verdi, og det skal være en String.
- ✓ Koden må sikre at man når *Return* ved utførelsen, ettersom funksjonen alltid skal tilordne en verdi (jfr. definisjonen av en funksjon).

Kommentarer i koden

Kommentarer er noe av det viktigste i koden. Den gjør det enklere å forstå koden, og derved lettere å finne feil og å vedlikeholde senere. Hele 70 til 80 % av alt arbeid med en applikasjon er vedlikehold. Kommentarene er ment for en senere leser, men ofte får du bruk for den selv også. Microsoft fremholder særlig to forhold som vesentlige for god, forståelig kode: Kommentarer i koden og gode variabelnavn (hørt det før☺?).

Jeg mener at man minst må kommentere slik:
- ✓ Forklar alltid hva en funksjon/prosedyre gjør – navnet sier ikke alltid alt.
- ✓ Kommentér etter *then* og etter *else*.
- ✓ Bruk ”overskrifter” som forteller hva som skal skje. Det beste er å skrive overskriftene først, slik sikkert norsklæreren din maste om når du skrev stiler (”lag disposisjon”). Overskriften forteller leseren om hensikten med koden.
- ✓ Jeg pleier alltid å skrive *’init* og *’akkumuler*, samt *’posit* og *’admit*. Det hjelper meg til å huske på det, og forklarer koden for leseren.
- ✓ Ikke kommenter det opplagte. Dette er f.eks. bortkastet og bare tilsmusser bildet, fordi enhver programmerer ser det selv:

```
If x(i) < x(minst) Then 'x(i) er mindre enn x(minst)
```

Denne kan nok være nyttig (”hvorfor testes dette?”):

```
If x(i) < x(minst) Then 'Admit - de to skal byttes om
```

I tillegg vil et godt program ha medfølgende dokumentasjon, der man har tegnet datastrukturer, komplekse valg og annet som en leser kan ha glede av for lett å forstå programmet.

Kommentarer er en kunst – du må trene på det i innleveringer, så får du tilbakemeldinger på om de er nyttige.

Tema A – Klasser og objekter

Å lære seg å bruke klasser og objekter er vanligvis en terskel for de fleste. Det er ikke det at det er så vanskelig, men det er uvant. I dette notatet skal jeg derfor gå relativt grundig inn i temaet. Senere kan bare *mye trening* hjelpe deg til å få klasser og objekter "under huden". Da vil du se at det er et meget kraftig og faktisk forenklende verktøy i programmering.

Kort historikk

Objektorientert programmering oppsto først i Norge, ved Norsk Regnesentral, med programmeringsspråket Simula rundt 1965. Det var professorene Ole-Johan Dahl og Kristen Nygård som var ledende med dette arbeidet. Begge jobbet senere ved Universitetet i Oslo[5]. Senere kom mange andre objektorienterte språk, f.eks. SmallTalk (fra Xerox PARC i 1971), C++ (fra AT&T Bell Labs v/Bjarne Stroustrup i 1985) og Java (fra Sun Microsystems i 1995). De aller fleste programmeringsspråk er nå enten objektorienterte helt fra start eller har fått objektorienterte tillegg/utvidelser. Visual Basic er et eksempel på det siste og har utviklet seg fra å være rent funksjonsorientert til å bli fullt objektorientert nå.

Nærmest alle språk – og i alle fall alle de ledende – er i dag er objektorienterte. Det er bred enighet om alle fordelene ved denne programmeringsmåten. Det finnes imidlertid fortsatt ledende eksperter som påpeker noen ulemper. De henviser bl.a. til at noen problemer egner seg dårlig for løsning med objektorientert programmering. Svært mange er også enige om at det er vanskeligheter med objektorientert programmering med relasjonsdatabaser. Dette siste skal dere selv få prøve senere i dette kurset. Vanskelighetene med relasjonsdatabaser sammen med objektorienterte programmer kan nok føre til at en ny databasetype – objektorienterte databaser – etter hvert overtar for relasjonsdatabasene (men det er ennå svært langt dit). Dere skal få bruke objektorienterte databaser senere i studiet.

Strukturerte variable = datastrukturer

Med *strukturert variabel*, mener vi en variabel som inneholder flere verdier. Arrays er et eksempel på strukturerte variable. Objekter er en spesiell form for strukturert variabel. Som introduksjon til objekter, skal jeg derfor først repetere litt om strukturerte variable.

Noen vanlige datastrukturer er:

a. *Mengde.* En mengde er en samling elementer av samme type. De er unike (ingen er lik noen annen) og de har ingen ordning.
b. *Bag.* En mengde der dubletter (flere like elementer) er tillatt.
c. *Sekvens = liste.* En bag der elementene er ordnet (de har en gitt rekkefølge – ikke nødvendigvis sortert).
d. *Array.* En sekvens der man kan få direkte tilgang til et element vha indeksen.

Mengder og bags finnes ikke implementert i VB. Sekvenser deklareres som *list (of T)* der T er en datatype. Arrays deklareres med parenteser bak variabelnavnet[6].

Kort om minnebruken

Når du deklarerer en enkel variabel, f.eks. med

```
Dim i As Integer
```

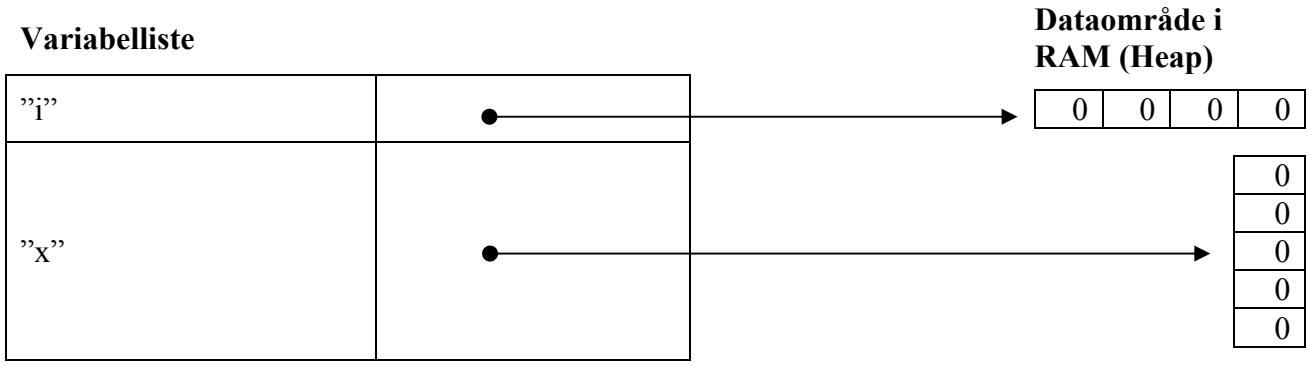
vil kompilatoren (som oversetter ditt tekstprogram til kjørbar kode) merke seg at du har tenkt å bruke en variabel som du kaller *i* og som er en *Integer*. Kompilatoren legger til kode som setter av plass til en *Integer* (fire bytes) i RAM, og legger variabelnavnet og adressen til verdien i RAM inn i en tabell. Dessuten initieres variabelverdien til 0.

[5] Gunnar Syrrist som forsker/underviser ved vår IT-seksjon har også arbeidet mye med Simulakompilatoren for PC.
[6] Merk at formelt er en *tabell* i en database en helt annen struktur enn array. Tabellen har først en *overskriftsrad* med *tekster* og deretter en *mengde* rader med *verdier* (unike, uordnet). Det å kalle en array for "tabell" er følgelig misvisende selvom det tradisjonelt har vært gjort i programmering.

Hvis du deklarerer en *array* (tabell), skjer omtrent det samme, f.eks.

```
Dim x(5) As Byte
```

men da settes det av plass til fem tall (på en byte hver) sammlet i RAM. Det første tallet får navnet x(0), det neste x(1) osv. til x(4). Hvor i RAM de fem verdiene havner, varierer for hver gang programmet kjøres. Det er avhengig av hvor det er ledig plass.



Ferdigdefinerte klasser

Et **objekt** kan enkelt beskrives som en samling data med metoder til. Metodene gjør det mulig for objektene å utføre handlinger. Objekter kan også reagere på hendelser, hvis vi definerer fornuftige hendelser for dem (det er f.eks. ikke aktuelt å klikke på et objekt som ikke er synlig på skjermen).

En **klasse** kan være enten

a. En slags *mal* som brukes når det skal lages et objekt. Malen angir hvilke data objektet skal kunne lagre, hva objektet skal kunne gjøre og hvilke hendelser det skal reagere på. Dette er den vanligste måten å bruke klasser på i Visual Basic.
b. En *del av programmet* som andre deler (og objektene) kan bruke. Klassen har et navn, og kan også lages slik at den selv kan lagre data og utføre handlinger, men det vanligste er at den bare opptrer som mal.
c. En *kombinasjon* av disse.

Microsoft har definert svært mange klasser som en del av VB. Uten at du kanskje har tenkt over det, har du brukt mange av dem allerede. Når du lager en kontroll på skjemaet ditt – f.eks. en kommandoknapp – lager du et **objekt** av **klassen** *Button* i navnerommet *System.Windows.Forms*. Da følger det en rekke **egenskaper** med ”på kjøpet”, f.eks. *Enabled, Text, Width, Font* og mange andre, som alle har en standardverdi (*Enabled = True* osv.). Det følger også **metoder** med, f.eks. *Focus()* og *Refresh()*. Dessuten kan en knapp reagere på **hendelser**, f.eks. *Click* og *GotFocus*. Alt dette er definert i *klassen Button*. Kontrollen er altså et *objekt* og i tillegg til å vises på skjermen, lagres det i RAM.

Du har også benyttet metoder knyttet til selve klassen (pkt b ovenfor), f.eks. når du skriver:

```
ant = Integer.Parse(txtAntall.Text)
```

Integer er da en *klasse*, og *Parse()* en metode som denne *klassen* tilbyr.

Alle klasser har en *konstruktør*, dvs. en metode som gjør det mulig å skape nye objekter basert på klassen som mal. Konstruktøren heter *New*[7] etterfulgt av klassens navn med eventuelle parametre. Når man lager et nytt objekt av en klasse, heter det å *instansiere* og objektet er en *instans* av klassen. Når du har trukket en kontroll fra verktøykassen og inn på skjemaet, har programmeringsmiljøet laget kode "bak kulissene" som bruker *New* til å skape kontrollen. Du kan også lage nye kontroller med programkode, f.eks. ved å skrive:

```
Dim cmdAvslutt As New Button()
```

[7] Det er spesielt for VB at konstruktøren heter *New*. I de fleste objektorienterte språk har konstruktøren samme navn som klassen. Allikevel blir kallet for å skape et nytt objekt seende svært likt ut, f.eks. slik i Java: "BankKonto nykonto = new BankKonto()" mens i VB står det "Dim nykonto As BankKonto = New BankKonto()".

Det finnes også en *destruktør* som kjøres når objektet slettes fra RAM. I VB heter destruktøren *finalize()*. Destruktøren bruker vi lite av forskjellige grunner som jeg ikke kommer inn på her. Dessuten har alle klasser automatisk noen andre metoder bl.a. *ToString()*, selv om den ikke alltid produserer noe fornuftig. En kommandoknapp med teksten "Eksempel" gir f.eks. dette som *ToString()*:

```
"System.Windows.Forms.Button, Text: Eksempel"
```

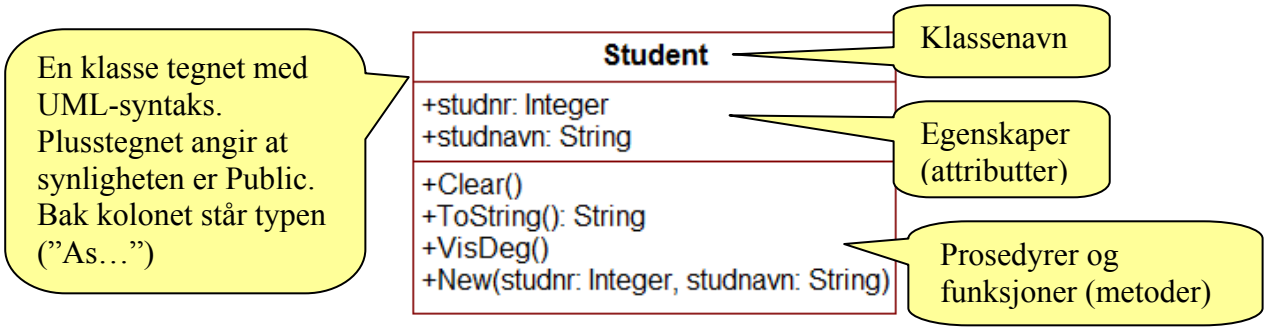
Det er neppe særlig interessant å få en slik tekst i en tekstboks.

Definere egne klasser

Det er umulig å komme særlig langt i programmering uten å forstå begrepene *klasser* og *objekter*. **Den beste måten å lære det på, er antakelig å lage klasser og objekter selv.**

Dessuten kan vi ikke vente at Microsoft har ferdige klasser for ting som vi vil bruke i våre programmer, som *kunde*, *student*, *faktura*, *bil* osv. – og i så fall kunne vi neppe ha brukt dem. Vi er derfor avhengig av å definere slike klasser selv. Vi skal se på hvordan det gjøres, og hvordan de brukes, gjennom et eksempel.

Studenteksempel



Definere studentklassen

Vi skal lagre opplysninger om studenter. Hver av dem har et studentnummer (heltall) og et studentnavn (tekst). Vi deklarerer klassen student med to egenskaper[8] (og foreløpig ingen metoder), slik

```
Public Class Student
    'Egendefinert klasse
    Public studnr As Integer
    Public studnavn As String
End Class
```

Legg merke til at *egenskaper* – altså variable på klassenivå – kan deklareres *Public* eller *Private* (og annet) som angir synligheten fra andre objekter. Da brukes vanligvis ikke ordet *Dim*, og programmeringsmiljøet vil fjerne det hvis du skriver det (det er altså lovlig, men unødvendig). Også klassen gjøres *Public* så den er synlig i hele prosjektet vårt.

Det er vanlig – og best – og deklarere klasser i en egen fil, én fil for hver klasse. Gi filen samme navn som klassen, altså her *Student.vb*. Du velger *Add Class* på *Project*-menyen. Det er også vanlig å bruke stor forbokstav i klassenavnet og det er standard i Java.

Når klassen er definert, kan vi bruke den i skjemaet vårt. Der kan vi skape en ny student og gi verdier til egenskapene, f.eks. slik:

[8] I VB kalles verdier som beskriver et objekt for *egenskaper* (*properties*), mens verdier som deklareres inne i en prosedyre/funksjon kalles *variable*. Dette er annerledes i f.eks. Java, der begge deler kalles *variable*.

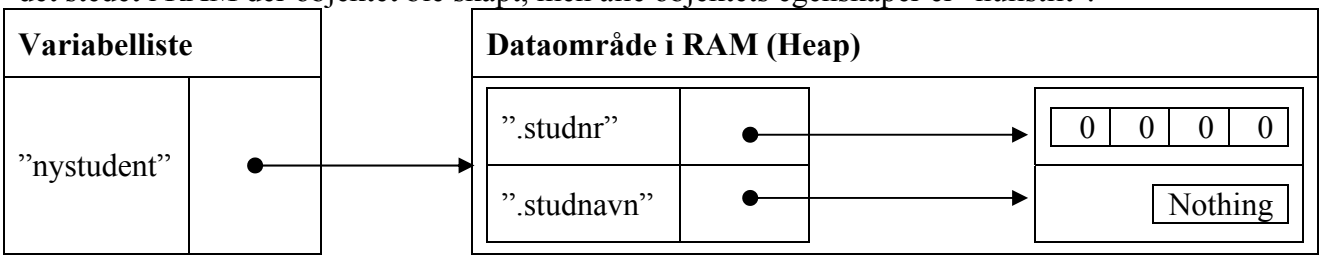
```
Private Sub butRegStudent_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butRegStudent.Click
    Dim nystudent As Student 'deklarer at vi vil bruke et objekt av klassen Student
    nystudent = New Student()'skaper objektet i RAM
    nystudent.studnr = 99
    nystudent.studnavn = "Knut"
```

Legg spesielt merke til at mens enkle variable og arrays får plass i RAM og initieres bare ved å skrive f.eks. *Dim i As Integer*, blir det ikke satt av plass i RAM bare ved å skrive *Dim nystudent As Student*. Objekter må *skapes* med *New* etterfulgt av et klassenavn. Det er dette som kalles å *instansiere* (lage et instans dvs et eksemplar av) klassen[9].

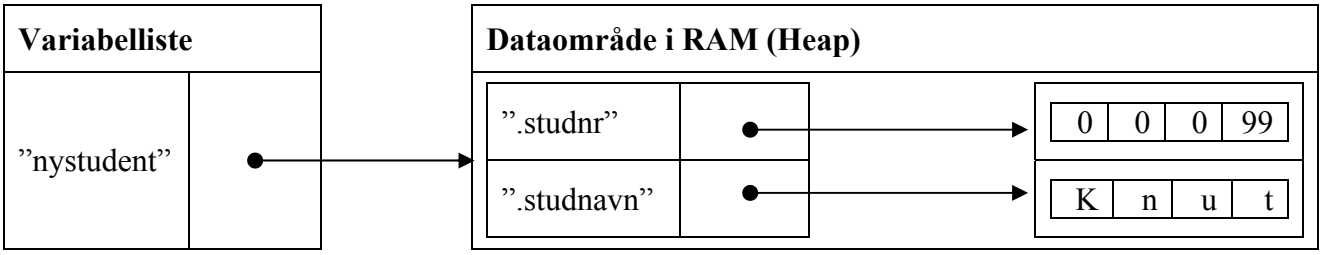
Etter den første setningen *Dim nystudent As Student* legges navnet *nystudent* i variabellisten og kompilatoren vet at det skal referere til et *Student*-objekt, men den peker ikke til noe. Pekerverdien er da *"Nothing"*:

Variabelliste	
”nystudent”	*Nothing*

Etter den andre setningen *nystudent = New Student()* skapes objektet i RAM og pekeren refererer til det stedet i RAM der objektet ble skapt, men alle objektets egenskaper er "nullstilt":



Deretter får de to egenskapene verdi med setningene *nystudent.studnr = 99* og *nystudent.studnavn = "Knut"*. Da ser det slik ut:



(Denne tegningen er forenklet – en tekst er egentlig et objekt og lagres på en annen måte enn dette.)

Variabellisten inneholder henvisning til objektet *nystudent*. Følges denne lenken, finner man en ny tabell for dette objektet, med henvisning til hver enkelt egenskap for objektet *nystudent*. Hvis du skriver f.eks. *nystudent.studnr* følges altså først referansen til objektet, og derfra videre til dette objektets verdi for *studnr*.

[9] En enklere form er å skrive *Dim nystudent As New Student()* eller bruke initiering: *Dim nystudent As Student = New Student().*

La oss nå lage en metode som ”nullstiller” en student, slik at vi kan bruke den ”om igjen”. Metoden deklareres som en del av klassen (i filen *Student.vb*):

```
Public Sub Clear()
    'Nullstiller alle egenskapene
    studnr = 0
    studnavn = ""
End Sub
```

Denne metoden er en del av klassen, og derfor "ser" den egenskapene uten noen punktnotasjon. Det gjelder uansett om de er deklarert *Public* eller *Private*.

En metode som dere har brukt mye, er *ToString()*. Den er automatisk definert for alle klasser, men den er ubrukelig i praksis. Derfor må vi deklarere vår egen *ToString()* for studentklassen:

```
Public Overrides Function ToString() As String 'overstyrer den arvede
    'ToString er definert fra før - dette er en overstyring
    Return studnr.ToString() & ": " & studnavn
End Function
```

Legg her spesielt merke til at vi her må føye til ordet *Overrides*, fordi klassen allerede har en *ToString()* automatisk definert. Vi skal altså lage vår egen definisjon, og den skal ”overstyre” den eksisterende. Siden dette er en *funksjon* må den også *returnere* noe. Her returnerer den en String. Kompilatoren vil passe på at du skriver kode som returnerer en verdi, og den krever at det skal være en *String* fordi du har deklarert funksjonen slik (*As String*).

Vi vil også at studenter skal kunne ”vise seg selv” i meldingsbokser[10]. Vi lager metoden *VisDeg()*:

```
Public Sub VisDeg()
    'Objektet "viser seg" i en meldingsboks
    MsgBox(Me.ToString(), MsgBoxStyle.Information, "Dette er meg")
End Sub
```

Uttrykket *Me* henviser til objektet selv.

Klassen får også automatisk metoden *New()* uten argumenter. Den er den som er brukt ovenfor til å skape et objekt – en instans – av denne klassen. Det er ikke helt bra, fordi det da er mulig å lage et *Student*-objekt uten reelt *studnr* og *studnavn*. Derfor bør vi lage vår egen *New()* der vi tvinger andre programmere til å oppgi et *studnr* og *studnavn*. Den kan se slik ut:

```
Public Sub New(ByVal studnr As Integer, ByVal studnavn As String)
    Me.studnr = studnr
    Me.studnavn = studnavn
End Sub
```

Legg merke til bruken av *Me* for å skille objektets eget *studnr* fra det som ble gitt som parameter. Sistnevnte er "lokal variabel" og navnet vil "overskygge" objektets egenskap fordi den er deklarert nærmere i koden.

Når vi lager vår egen *New()* vil kompilatoren ikke legge til standard *New*. Derved tvinges de som vil skape en *Student* nå til å legge inn et nr og et navn når objektet skapes:

```
Dim nystudent As Student 'deklarer at vi vil bruke et objekt av klassen Student
nystudent = New Student()'skaper objektet i RAM
nystudent.studnr = 99
nystudent.studnavn = "Knut"
nystudent = New Student(99, "Knut")
```

eller enklere ved direkte initiering:

```
Dim nystudent As New Student(99, "Knut")
```

[10] Vanligvis bør vi unngå at objekter kommuniserer med brukeren, men vi gjør det her for illustrasjonens skyld.

Den ferdige studentklassen

Slik ser klassen ut nå:

```
Option Strict On
Option Explicit On

'************************
'* Klassen Student
'************************
Public Class Student
    'Egendefinert klasse
    Public studnr As Integer
    Public studnavn As String

    Sub New(ByVal studnr As Integer, ByVal studnavn As String)
        Me.studnr = studnr
        Me.studnavn = studnavn
    End Sub

    Public Sub Clear()
        'Nullstiller alle egenskapene
        studnr = 0
        studnavn = ""
    End Sub

    Public Sub VisDeg()
        'Objektet "viser seg" i en meldingsboks
        MsgBox(Me.ToString(), MsgBoxStyle.Information, "Dette er meg")
    End Sub

    Public Overrides Function ToString() As String 'overstyrer den arvede
        'ToString er definert fra før - dette er en overstyring
        Return studnr.ToString() & ": " & studnavn
    End Function
End Class
```

Arrays (tabeller) med objekter

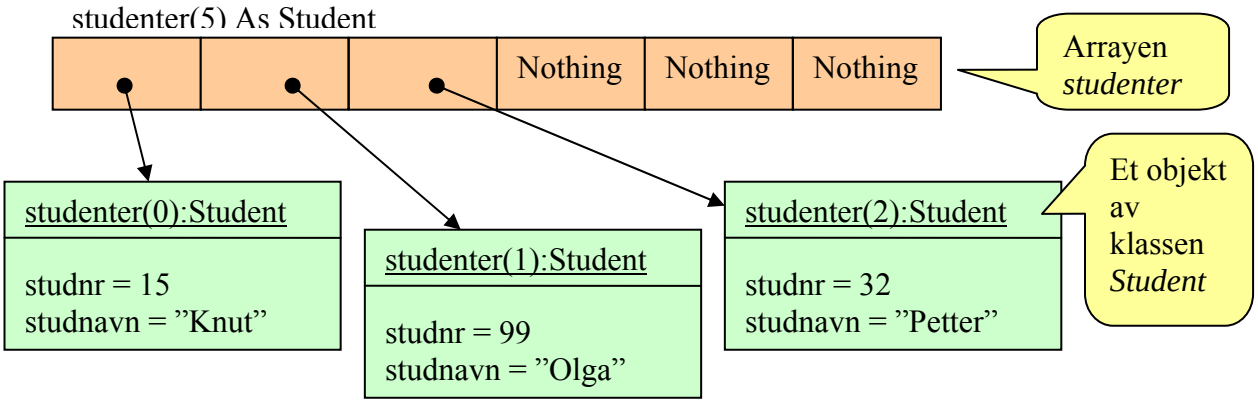
Dere har tidligere arbeidet med *arrays*. Da har dere laget arrays med enkle datatyper, f.eks.

```
    Dim høyde(15) As Integer 'vanlig - med automatisk initiering til 0
    Dim navn() As String 'uten angitt dimensjon
    Dim tall() As Integer = {5, 15, 7, 32} 'med eksplisitt initiering
```

Arrays representerer flere verdier av samme type og med samme navn, skilt med en indeks. Det er også mulig – og vanlig – å lage arrays med *objekter*. Da må alle objektene i arrayen være av samme type, dvs objekter som tilhører samme klasse. Hvis vi f.eks. vil ha en array med plass til 6 studenter, kan vi skrive:

```
        Dim studenter(5) As Student 'plass til 6 student-objekter
```

I den arrayen *studenter* som da lages, settes det av plass i hvert element (celle) til en *referanse* (også kalt *peker*) som henviser til en student. Slik kan det se ut i RAM når tre studenter er registrert (her ser vi bort fra kompilatorens variabelliste):



Nothing[11] er en referanse (peker) til ”ingenting”.

Hvis vi nå vil *henvise* til en bestemt student i arrayen, f.eks. Olga, skriver vi som vanlig *studenter(1)* og bruker det f.eks. slik for å vise navnet:

```
MsgBox(studenter(1).studnavn)
```

Denne meldingsboksen viser navnet ”Olga”. Legg spesielt merke til at vi må skrive hvilket element i arrayen vi skal ha *før* vi bruker punktum for å angi egenskapen eller prosedyren/funksjonen. Det kan bli mer enn ett punktum også, hvis vi f.eks. vil vise studentens *studnr*:

```
MsgBox(studenter(1).studnr.ToString())
```

For å *opprette* en ny student i element 3, skriver vi da

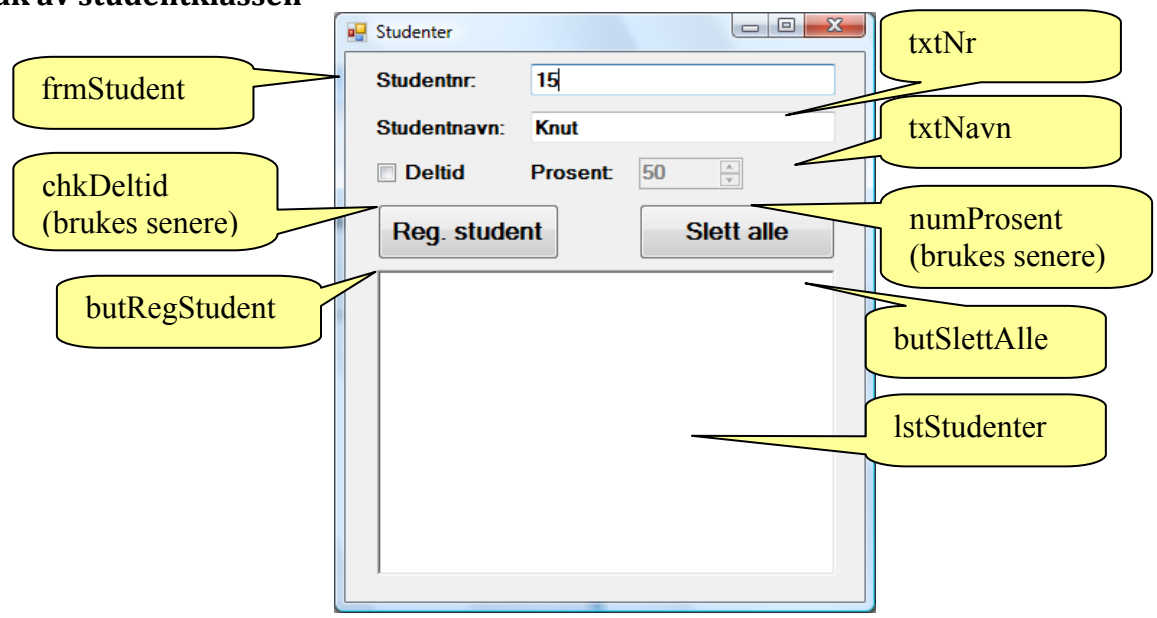
```
studenter(3) = New Student(105, "Cathrine") 'lager en ny student i studenter(3)
```

Husk å holde orden på hvor mange som er registrert til enhver tid, så vi vet hvor vi skal sette inn den neste studenten[12]. Det er vanlig å bruke en egen variabel, f.eks. *antStud* til dette.

[11] *Nothing* er et ord spesielt for VB. I databaser heter det *Null* og gjelder en verdi i tabellen som ikke er oppgitt. Siden *Nothing* ikke er en verdi i seg selv, kan man ikke bruke sammenlikningsoperatorer for den. Det er altså ikke lov til å skrive ”= *Nothing*”, du må skrive ”*Is Nothing*”.

[12] Hvis man bruker en *liste* istedenfor *array* unngår man problemet, men det kommer vi tilbake til i en senere forelesning.

Bruk av studentklassen



Vi har altså et skjema (Form) som skal bruke Student-objekter. Vi skal registrere studenter, legge dem i en array og vise dem i en listeboks. De skal også kunne slettes. Senere skal vi også kunne registrere deltidsstudenter.

Slik kan vi programmere knappen *butRegStudent* som henter studentnummer og navn fra skjemaets tekstbokser:

```
Public Class frmStudent
    Private Const MAX_STUD As Integer = 50 'maks antall studenter
    Public studenter(MAX_STUD) As Student 'kan være Student eller Deltid
    Dim antstud As Integer = 0 'antall som er registrert hittil

    Private Sub butRegStudent_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butRegStudent.Click
        'Legger en ny student inn i arrayen studenter

        'Skap ny student av riktig klasse og legg inn i arrayen
        Dim nystudent As New Student(CInt(txtNr.Text), txtNavn.Text)

        'Vis studenten i en meldingsboks
        nystudent.VisDeg() 'vis i meldingsboks

        'Legg inn studenten i arrayen studenter
        studenter(antstud) = nystudent
        antstud += 1

        'Oppdater skjemaet
        listAlle()
        Me.Clear()
    End Sub
```

Her brukes objektets egen *ToString()*. Det er objektet selv som lager den strengen som inneholder både studnr og studnavn. Meldingsboksen vil se slik ut:

butRegStudent bruker *listAlle()* som kan gjøres slik:

```
Private Sub listAlle()
    'Lister alle registrerte studenter i lstStudenter
    lstStudenter.Items.Clear()
    For i As Integer = 0 To antstud - 1
        lstStudenter.Items.Add(studenter(i).ToString())
    Next
End Sub
```

Vider bruker hovedprogrammets *clearForm()* som kan se slik ut:

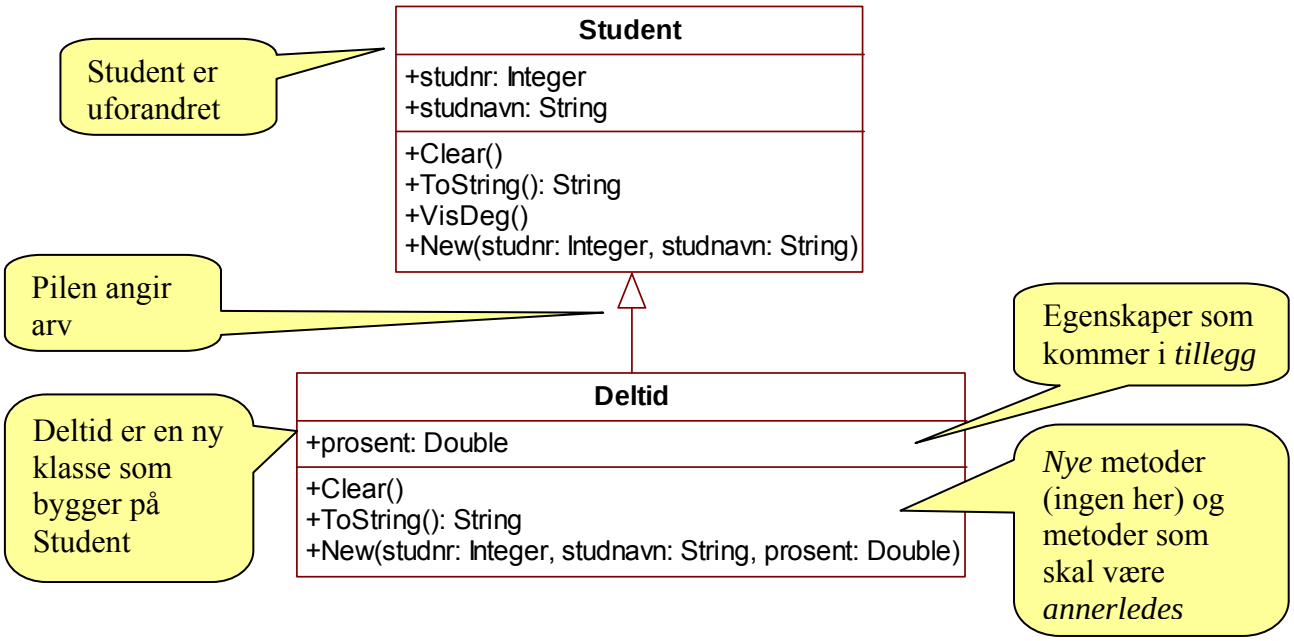
```
Private Sub Clear()
    'Sletter input i skjemaet
    txtNr.Clear() 'bruker tekstboksen ferdigdefinerte Clear()
    txtNavn.Clear() 'det samme
    chkDeltid.Checked = False
    txtNr.Focus() 'bruker tekstboksen ferdigdefinerte Focus()
End Sub
```

Og slik kan alle studentene slettes igjen fra arrayen *studenter*:

```
Private Sub butSlettAlle_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butSlettAlle.Click
    'Sletter alle studenter, listen og nullstiller antstud
    Dim i As Integer
    For i = 0 To antstud - 1
        studenter(i).Clear() 'bruker student-objektets egen Clear()
    Next
    antstud = 0
    lstStudenter.Items.Clear() 'sletter alle linjene i listeboksen
    Me.Clear() 'bruker skjemaets egen Clear()
End Sub
```

En ny klasse Deltid som bygger på klassen Student

Vi vil nå også kunne registrere deltidsstudenter. De er akkurat som vanlige studenter, men har i tillegg en egenskap *prosent* som angir hvor stor del av studieåret de tar. Det vil jo da være tungvint (og redundant) å definere klassen *Deltid* helt fra bunnen. Isteden utnytter vi det vi allerede har gjort, gjennom *arv*. Klassen *Deltid* arver da alt (og da mener vi *alt*) fra den overordnede klassen *Student*, og så endrer vi og legger til etter behov. Det er ikke mulig å fjerne noe. Dette kan sammenliknes med å si at ”en deltidsstudent er som en student, bare at i tillegg…”.



Ting å merke seg:

a. Egenskaper:
    a. Det *er* tillatt å deklarere arvede egenskaper pånytt, men det er aldri lurt. De vil "overskygge" egenskapene som ble arvet og det blir svært vanskelig å holde orden på. Derfor **gjøres det i praksis aldri**.
    b. Man kan *legge til* nye egenskaper, og det er bare det som er aktuelt. **Det gjøres ofte.**
    c. Det er **ikke mulig** å *fjerne* egenskaper som er arvet.
b. Metoder:
    a. Det er tillatt å deklarere arvede metoder pånytt med samme argumenter (*overriding*). Den nye deklarasjonen vil da "legge seg oppå" den arvede. **Dette gjøres mye**. De overordnede metodene må da merkes *Overridable*. De skal altså virke annerledes enn i den overordnede klassen. Her er det gjort for *Clear()* og *ToString()*.
    b. Det er tillatt å deklarere metoder med samme navn som de arvede, men med andre argumenter (*overloading*). **Det gjøres ofte.** Her er det gjort med *New*.
    c. Man kan legge til helt nye metoder. **Det er svært vanlig**, men det er ikke gjort her.
    d. Man kan legge til flere metoder i samme objekt, med samme navn, bare de har forskjellig *signatur*. Signaturen utgjøres av metodens *navn*, *argumenttype(r)* og *argumentrekkefølgen*. Signaturen ser altså bort fra argumentenes navn og returtype for funksjoner. Dette heter *overloading* og er **også ganske vanlig**, men det er ikke gjort her.

For ordens skyld nevner jeg at det er uaktuelt å deklarere arv uten å gjøre noen endring i klassen som arver. Da vil de jo bli helt like.

Vi må nå først gjøre en liten endring i *Student*-klassen siden *clear()* skal overstyres:

```
Public Overridable Sub Clear()
```

(Det er ikke nødvendig å gjøre det samme med *ToString()* for den er allerede merket *Overriding* i *Student*-klassen.)

Deretter kan vi skrive klassen *Deltid* slik – i en egen fil *Deltid.vb*:

```
Option Strict On
Option Explicit On

'************************
'* Klassen Deltid - arver fra Student
'************************
Public Class Deltid
    Inherits Student
    'Som andre studenter, men noe mer og/eller annerledes
    Public prosent As Double = 50 'standardverdi er 50%

    Sub New(ByVal studnr As Integer, ByVal studnavn As String, ByVal prosent As Double)
        MyBase.New(studnr, studnavn)
        Me.prosent = prosent
    End Sub

    Public Overrides Sub Clear() 'overstyrer den arvede
        MyBase.Clear() 'kaller overordnet klasse sin metode
        prosent = 0 '... og legger til dette
    End Sub

    'Public VisDeg() arves uten endringer

    Public Overrides Function ToString() As String 'overstyrer den arvede
        Return MyBase.ToString() & " - " & prosent.ToString() & "%"
    End Function

End Class
```

Merk at klassen som helhet er merket *Inherits Student*. Merk videre at henvisningen til *MyBase* i metoden *clear()* er til den klassen som *Deltid* arver fra (VB kaller den *Base Class*, ellers heter den ofte *Metaklassen*). Ved å henvise til den, blir den kjørt, slik at først tømmes verdiene for studnr og navn, deretter tømmes *prosent*. Vi utnytter alt vi kan av det som allerede er gjort! Helt tilsvarende har vi utnyttet det som allerede er gjort i *Student.ToString()* – vi bare legger litt til i strengen.

I hovedprogrammet (skjemaet) legger vi nå til en hendelsesprosedyre for *chkDeltid*:

```
    Private Sub chkDeltid_CheckedChanged(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles chkDeltid.CheckedChanged
        'Gjør numProsent aktiv eller ikke avhengig av status for chkDeltid
        numProsent.Enabled = chkDeltid.Checked
    End Sub
```

Dette gjør *numProsent* tilgjengelig når sjekkboksen er haket av, ellers ikke. Vi bruker sjekkboksen som signal om at det skal skapes et *Deltid*-objekt istedenfor et *Student*-objekt.

En meget interessant effekt av å bruke arv, er at arrayen *student* som er satt til å inneholde *Student*-objekter, også kan inneholde *Deltid*-objekter. Det er jo nemlig slik at *Deltid*-objekter er en slags *Student*-objekter de også! Når vi kaller et objekt i arrayen og ber det f.eks. om å "vise seg", vil programmet selv finne ut om det er et *Student*-objekt eller et *Deltid*-objekt, og velge den riktige metoden. Dette kalles *polymorfisme* ("mangeformig") og er en meget kraftig og viktig mekanisme i objektorientert programmering.

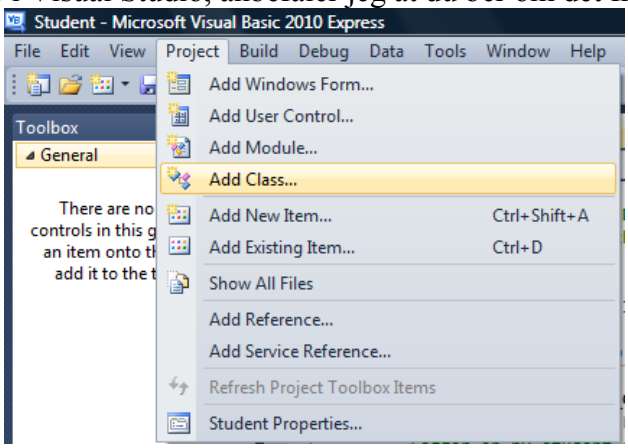
Nå kan vi også skape deltidsstudenter og bruke dem på akkurat samme måte som vanlige studenter. Vi kan altså også legge dem inn i den samme arrayen. Hovedprogrammet endrer vi slik at det også kan håndtere deltidsstudenter. Det er bare *butRegStudent_Click* som må endres, fordi vi kanskje skal skape en *Studnet* eller kanskje en *Deltid*:

```
    Private Sub butRegStudent_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butRegStudent.Click
        'Legger en ny student inn i arrayen studenter

        'Skap ny student av riktig klasse og legg inn i arrayen
        Dim nystudent As Student 'kan være heltid eller deltid (gjennom arv)
        If chkDeltid.Checked Then 'dette er en deltidsstudent
            nystudent = _
                New Deltid(CInt(txtNr.Text), txtNavn.Text, numProsent.Value)
        Else 'dette er en heltidsstudent
            nystudent = _
                New Student(CInt(txtNr.Text), txtNavn.Text)
        End If

        'Vis studenten i en meldingsboks
        nystudent.VisDeg() 'vis i meldingsboks

        'Legg inn studenten i arrayen studenter
        studenter(antstud) = nystudent
        antstud += 1

        'Oppdater skjemaet
        listAlle()
        clearForm()
    End Sub
```

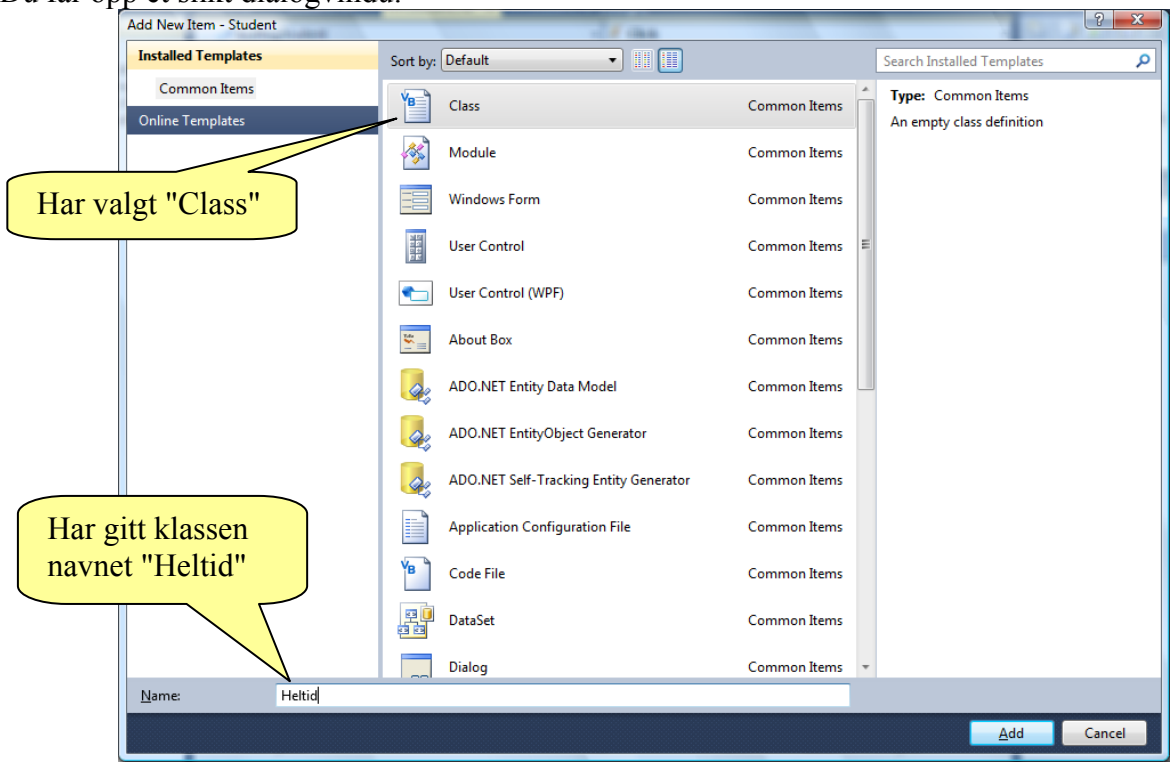
I praksis ville vi nok gjort ytterligere forbedringer, f.eks. sikret mot feil input, mot at det ble for mange for arrayen, vi ville lagret objektene til fil og lest dem ved neste oppstart osv. Det skal vi foreløpig la ligge, så nå er programmet ferdig.

Lage klasser i Visual Studio

Når du skal lage en klasse i Visual Studio, anbefaler jeg at du ber om det fra *Project*-menyen:



Du får opp et slikt dialogvindu:



Resultatet er en ny fil med navnet "Heltid.vb" som bare inneholder et par linjer:

```
Public Class Heltid

End Class
```

Filen er lagret sammen med skjemaet (altså i samme prosjekt).

Egne hendelser

Note: *Dette er ikke pensum, men vises for helhetens skyld. Eksemplet er kunstig, bare for å vise prinsippet til spesielt interesserte.*

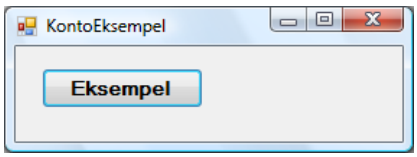
Vi kan legge til egne hendelser i en klasse *Konto*. Den skal reagere hvis saldoen blir negativ. Da må vi gjøre følgende:

1. Deklarere en *hendelse* i klassen *Konto*.
2. Programmere når hendelsen skal inntreffe (det er jo ikke mulig for brukeren å klikke på et objekt som ikke er synlig).
3. Deklarere objektet *WithEvents* i hovedprogrammet som egenskap (på klassenivå).
4. Skrive den nødvendige hendelsesprosedyren i hovedprogrammet.

Konto-klassen

```
Public Class Konto 'Kontoklassen
    Public Event Overtrekk(ByVal sender As Konto) 'deklarer en hendelse
    Public kontonr As Integer
    Public saldo As Decimal = 0

    Public Sub New(ByVal nr As Decimal)
        kontonr = nr
    End Sub

    Public Sub oppdater(ByVal beløp As Decimal)
        saldo += beløp
        If saldo < 0 Then 'kontoen er overtrukket
              RaiseEvent Overtrekk(Me) 'skaper en hendelse
        End If
    End Sub
End Class
```

Hovedprogram (skjemaet)




```
Public Class KontoEksempel 'Skjemaet
    Public WithEvents konto As Konto 'WithEvents kun tillatt for egenskaper (skjemanivå)

    Private Sub butEksempel_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butEksempel.Click

        '1. Opprett en konto
        MsgBox("Oppretter konto 2535")
        konto = New Konto(2535)

        '2. Sett inn 300
        MsgBox("Setter inn 300")
        konto.oppdater(+300)

        '3. Ta ut 500 - vil gi overtrekk
        MsgBox("Tar ut 500")
        konto.oppdater(-500)
    End Sub

    Private Sub konto_Overtrekk(ByVal sender As Konto) Handles konto.Overtrekk
        'Håndterer Overtekk-hendelse for konto
        MsgBox(sender.kontonr & " er overtukket. Saldoen er " & sender.saldo, _
               MsgBoxStyle.Exclamation, "Feil")
    End Sub
End Class
```

Legg merke til at hendelsen er deklarert øverst i koden for *Konto*-klassen og at *Konto*-objektet er deklarert som *egenskap*, altså på klassenivå (for skjemaet) *med hendelser*.

Når kontoen oppdateres sjekkes det om saldoen er negativ. I så fall skapes en hendelse, som fanges opp av hovedprogrammets hendelsesprosedyre *konto_overtrekk* (det står *Handles konto.Overtrekk*). Kontoen sender med en referanse til seg selv (*Me*). Den kan hendelsesprosedyren bruke til å hente opplysninger om kontoen, her *kontonr* og *saldo*.

Det ferdige programmet for studenter

Skjemaet

```
Option Strict On
Option Explicit On

'***********************
'* Skjemaet
'* Bruker klassene Student og Deltid
'* OBS! Her er det ingen inputkontroll!
'***********************
Public Class frmStudent
    Private Const MAX_STUD As Integer = 50 'maks antall studenter
    Public studenter(MAX_STUD) As Student 'kan være Student eller Deltid
    Dim antstud As Integer = 0 'antall som er registrert hittil

    Private Sub butRegStudent_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butRegStudent.Click
        'Legger en ny student inn i arrayen studenter

        'Skap ny student av riktig klasse og legg inn i arrayen
        Dim nystudent As Student 'kan være heltid eller deltid (gjennom arv)
        If chkDeltid.Checked Then 'dette er en deltidsstudent
            nystudent = _
                New Deltid(CInt(txtNr.Text), txtNavn.Text, numProsent.Value)
        Else 'dette er en heltidsstudent
            nystudent = _
                New Student(CInt(txtNr.Text), txtNavn.Text)
        End If

        'Vis studenten i en meldingsboks
        nystudent.VisDeg() 'vis i meldingsboks

        'Legg inn studenten i arrayen studenter
        studenter(antstud) = nystudent
        antstud += 1

        'Oppdater skjemaet
        listAlle()
        Me.Clear()
    End Sub

    Private Sub listAlle()
        'Lister alle registrerte studenter i rtfStudenter
        rtfStudenter.Items.Clear()
        For i As Integer = 0 To antstud - 1
            lstStudenter.Items.Add(studenter(i).ToString())
        Next
    End Sub

    Private Sub Clear()
        'Sletter input i skjemaet
        txtNr.Clear() 'bruker tekstboksen ferdigdefinerte Clear()
        txtNavn.Clear() 'det samme
        chkDeltid.Checked = False
        txtNr.Focus() 'bruker tekstboksen ferdigdefinerte Focus()
    End Sub
```

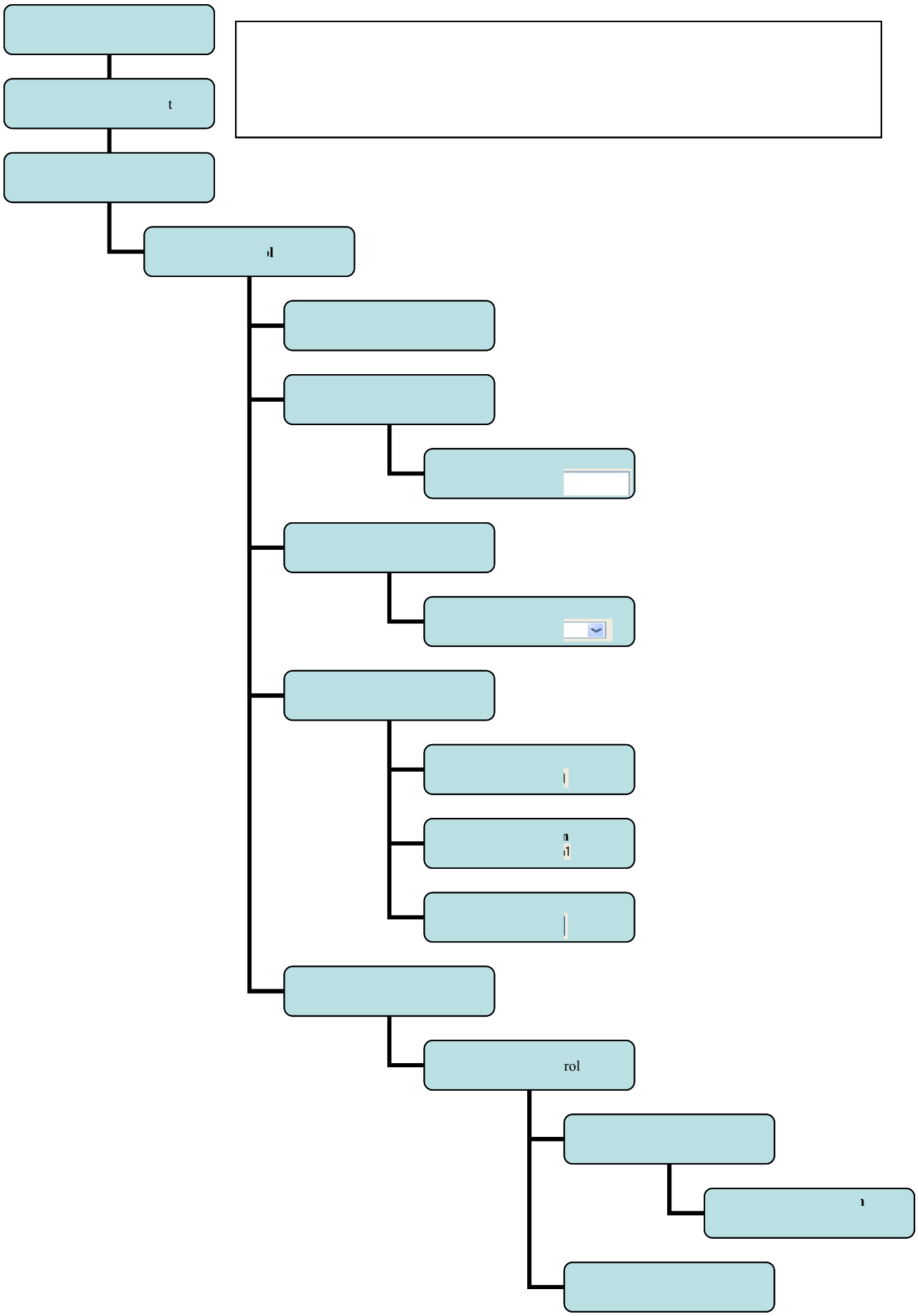
```
    Private Sub butSlettAlle_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butSlettAlle.Click
        'Sletter alle studenter, listen og nullstiller antstud
        Dim i As Integer
        For i = 0 To antstud - 1
            studenter(i).Clear() 'bruker student-objektets egen Clear()
        Next
        antstud = 0
        lstStudenter.Items.Clear() 'sletter alle linjer i listeboksen
        Me.Clear() 'bruker skjemaets egen Clear()
    End Sub

    Private Sub chkDeltid_CheckedChanged(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles chkDeltid.CheckedChanged
        'Gjør numProsent aktiv eller ikke avhengig av status for chkDeltid
        numProsent.Enabled = chkDeltid.Checked
    End Sub
End Class
```

Klassen Student

```
Option Strict On
Option Explicit On

'************************
'* Klassen Student
'************************
Public Class Student
    'Egendefinert klasse
    Public studnr As Integer
    Public studnavn As String

    Public Sub New(ByVal studnr As Integer, ByVal studnavn As String)
        Me.studnr = studnr
        Me.studnavn = studnavn
    End Sub

    Public Overridable Sub Clear()
        'Nullstiller alle egenskapene
        studnr = 0
        studnavn = ""
    End Sub

    Public Sub VisDeg()
        'Objektet "viser seg" i en meldingsboks
        MsgBox(Me.ToString(), MsgBoxStyle.Information, "Dette er meg")
    End Sub

    Public Overrides Function ToString() As String 'overstyrer den arvede
        'ToString er definert fra før - dette er en overstyring
        Return studnr.ToString() & ": " & studnavn
    End Function
End Class
```

Klassen Deltid

```
Option Strict On
Option Explicit On

'************************
'* Klassen Deltid - arver fra Student
'************************
Public Class Deltid
    Inherits Student
    'Som andre studenter, men noe mer og/eller annerledes
    Public prosent As Double = 50 'standardverdi er 50%

    Sub New(ByVal studnr As Integer, ByVal studnavn As String, ByVal prosent As Double)
        MyBase.New(studnr, studnavn)
        Me.prosent = prosent
    End Sub

    Public Overrides Sub Clear() 'overstyrer den arvede
        MyBase.Clear() 'kaller overordnet klasse sin metode
        prosent = 0 '... og legger til dette
    End Sub

    'Public VisDeg() arves uten endringer

    Public Overrides Function ToString() As String 'overstyrer den arvede
        Return MyBase.ToString() & " - " & prosent.ToString() & "%"
    End Function
End Class
```


t
rol

De objektene vi bruker oftest, er *kontroller* (controls). Det inkluderer *forms* (vinduer) og elementene inne i vinduet, f.eks. etikett, tekstfelt og kommandoknapp. Når vi lager en ny slik, bruker vi en *klasse* (class) som mal. Alle egenskapene får en standardverdi. Alle objekter har *egenskaper*, *metoder* og reagerer på *hendelser*.

Hendelser (events)

som objektet kan *reagere* på, f.eks. Click, DoubleClick, TextChanged og MouseMove.

Microsoft har bestemt hvilke hendelser deres objekter skal reagere på, og forskjellige slags objekter kan reagere på forskjellige slags hendelser.

Metoder (methods)

som objektet kan *gjøre* på anmodning, f.eks. Focus() (setter seg selv i fokus som om brukeren hadde klikket på det) og ToString() (gjør om objektet til en tekst for utskrift).

Metodene er laget ferdig av Microsoft, og forskjellige slags objekter har forskjellige metoder.

Egenskaper (properties)

som *beskriver* objektet, f.eks. Name, BackColor, BorderStyle, Size og Visible.

Microsoft har bestemt hvilke egenskaper deres objekter skal ha, og forskjellige slags objekter har forskjellige slags egenskaper, men *alle* objekter har et navn.

Egenskaper har alltid en *verdi* (men verdien kan være ”ingenting”), f.eks. kan høyden på et tekstfelt være 250. Den verdien vi gir dem i egenskapsvinduet under grensesnittdesignet, er startverdien de har når programmet starter. Noen få egenskaper er *statiske* og kan ikke endres mens programmet kjøres, f.eks. objektets navn. De fleste egenskaper er imidlertid *dynamiske* og kan endres mens programmet kjører.

Merk:
Vi kan *definere våre egne objekter* også. Da bestemmer vi selv hvilke egenskaper og metoder de skal ha og hvilke hendelser de skal reagere på.

Tema B – Filer og feilfeller

Filer er en mengde bytes lagret på et ytre lagringsmedium og som er gitt et navn. Operativsystemets *filsystem* holder orden på filene, hva de heter, hvor de fysisk er lagret, hvor store de er, om de er opptatt/ledige osv.

Filer inneholder enten *data* eller *programmer*. I MS Windows angir filens etternavn hva slags fil det er, slik at Windows skal starte det riktige programmet når vi ber om at filen skal "åpnes". Windows vil f.eks. starte Word hvis filen har etternavnet *doc*. Det er imidlertid lov å gi en fil et hvilket som helst etternavn, men det kan bli upraktisk å ikke følge navnestandarden.

Enkeltstående filer kalles ofte *"flate filer"*. Det finnes også filer som det er en sammenheng mellom og som håndteres av et eget program, og de kalles *databaser*. Vi kan f.eks. lagre alle eiere i én fil og båtene deres i en annen fil. Da må vi selv holde orden på hvem som eier den enkelte båten. Hvis vi bruker en database, vil databaseprogrammet passe på sammenhengen mellom båtene og eierne deres. **Dette notatet omhandler bare enkeltstående, "flate" datafiler.** Databaser vil bli tatt opp senere.

Tips: Ting går ofte galt når man bruker filer. Du bør derfor sette opp en feilfelle – se nedenfor om det. Det er vanskelig å gjennomteste programmet, for feilen kan oppstå først når programmet flyttes, f.eks. fordi filen ikke finnes lenger.

Noen filbehandlingsoppgaver med klassen System.IO.File

Det finnes metoder til å slette, kopiere, flytte og gi nytt navn til en fil. Da kan du bruke klassen *System.IO.File*. Noen nyttige metoder for filbehandling med *File* (du finner flere i VB hjelp):

- ✓ *copy(sourceFileName As String, destFileName As String, Optional overwrite As Boolean)* kopier filen. Hvis *overwrite* er *True* kan kopien overskrive en eksisterende, hvis den er *False* er overskriving ikke tillatt (default er *False*).
- ✓ *move(sourceFileName As String, destFileName As String)* flytter en fil. Du kan angi bare katalognavnet i *destFileName* – da vil filen beholde navnet sitt. Overskriving er ikke tillatt.
- ✓ *delete(path As String)* sletter en fil.
- ✓ *exists(path) As Boolean* sjekker om en fil finnes. Du kan også bruke en feilfelle til å håndtere feil som oppstår hvis en fil (som det skal leses fra) ikke finnes, eller andre feil ved filen.

Når bør filer åpnes/lukkes?

En fil som det skal skrives til, må naturligvis åpnes *før* lesingen/skrivingen kan skje, og den må før eller senere lukkes igjen. I mellomtiden står filen merket i filsystemet som "Read Only". Andre kan få lese fra filen, men ikke skrive til den.

Å åpne/lukke en fil, er relativt tidkrevende, både fordi noe må gjøres med filbufferet og fordi det må gjøres kall til operativsystemet (Utforsker/Explorer). Av denne grunn bør man åpne/lukke filer minst mulig. Altså bør filene åpnes i *Form_Load* og lukkes i *Form_Closed*.

På den annen side, er en åpen fil utilgjengelig – i alle fall for skriving – fra andre programmer og brukere. Hvis noe skjer, eller brukeren avbryter arbeidet, kan en fil bli stående åpen lenge. Av denne grunn bør filene åpnes senest mulig og lukkes så fort som mulig, selv om det fører til at samme fil åpnes/lukkes flere ganger. Altså bør filene åpnes rett før lesing og lukkes umiddelbart etter skriving.

Hvilken av disse to strategiene som er riktig, avhenger av situasjon. I et flerbrukermiljø, der mange bruker samme fil, er nok den siste strategien best. I et enbrukermiljø, kan den første være riktigst.

Det som står fast, er at alle filer bør lukkes før programmet avslutter, og vi bør skrive kode som gjør det.

Filen åpnes for skriving – og bindes opp – når *StreamWriter-objektet* skapes og den lukkes med metoden *Close()*.

Sekvensielle tekstfiler

Fordelen med sekvensielle filer, er at de kan ha fri lengde (begrenset av operativsystemets filsystem) som kan variere dynamisk med behovet. Man kan alltid legge til flere bytes bakerst (sålenge det er plass på det ytre lageret). Lengden på dataene vil variere, og derfor skilles dataene med et skilletegn. Da kan man lagre strenger av varierende lengde: "Knut W, Hansson" i 15 tegn mens "Are Ås" tar seks tegn. Alt skrives som tekst, og funksjonen *ToString()* – som alle objekter har – brukes. Tekstfilen kan være inndelt i linjer, med *vbNewline* (eller *vbCrLf*) bakerst på hver linje.

Ulempen med sekvensielle filer at de må skrives og leses sekvensielt forfra og utover, og det blir langsomt for store filer. Det blir også svært tungvint å legge til, slette eller endre data et sted inne i filen (det krever kopiering av filen). Sortering blir også svært tungt og vanskelig.

Til sekvensielle tekstfiler bruker man objekter[13] av klassene
1) **System.IO.StreamWriter** til output
2) **System.IO.StreamReader** til input

Vi kan enkelt si at man leser fra en tekstfil fil med *Read() As Integer* og *Readline() As String* og skriver til en fil med *Write(data)* og *WriteLine(data)*. Hvis du bruker *Read()* får du et tegn, men det returneres som et heltall – vanligvis i Unicode (men det er mulig å angi annet) – du må selv konvertere tallet til et tegn.

Du må oppgi full sti når du åpner en fil. Hvis du vil henvise til programmets sti, så heter den
```
My.Application.Info.DirectoryPath
```
Da blir programmet lett å flytte. En subkatalog "filer" vil da hete
```
My.Application.Info.DirectoryPath & "\filer"
```

Ad 1 StreamWriter

Du må først deklarere en variabel som peker til et objekt:
```
Dim utfil As System.IO.StreamWriter
```

Objektet må også skapes med *New*, f.eks.
```
utfil = New System.IO.StreamWriter(sti & filnvn, False) 'ikke append
```

Syntaksen for konstruktøren til *StreamWriter()*, dvs. den som kjøres når vi skriver *New*:
```
New System.IO.StreamWriter(path As String, append As Boolean)
```
Path skal være en fullstendig sti og filnavn (for- og etternavn). *Append* angir om filen skal tømmes hvis den finnes fra før eller om nye data skal legges til bakerst. Hvis filen ikke finnes, vil den bli laget.

[13] I tidligere versjoner av Visual Basic har man brukte prosedyrene *FileOpen()* og *FileClose()* .Det er fortsatt mulig av hensyn til kompatibilitet bakover. Disse prosedyrene vedlikeholdes imidlertid ikke lenger, og bør derfor ikke brukes. Nå skal man bruke objekter til det samme.

Noen metoder for *StreamWriter*:

- ✓ *Close()* er en prosedyre som lukker filen (returnerer ingenting).
- ✓ *Write(data)* er en prosedyre som skriver dataene til filen. Dataene kan være de aller fleste typer som f.eks. Boolean, String, Integer, objekter – i prinsippet alle som har metoden *ToString*.
- ✓ *WriteLine(data)* er en prosedyre som skriver en hel linje (avsluttes med linjeskift) til filen. Data kan være som for *Write()*.

Skrive tekstfil med skilletegn ("tegnseparert fil")

Ofte vil man lage tekstfiler der hver post ligger på én linje, og dataene i posten er adskilt med et skilletegn. Det kan f.eks. være nyttig til lagring av attributtene i objekter. Det er vanlig å bruke komma som skilletegn, og filene kalles da "kommaseparert fil" eller CSV-filer (CSV = "Comma-Separated Values"). Det passer imidlertid ikke med komma, hvis dataene i seg selv inneholder komma. Da må vi bruke et annet skilletegn, f.eks. $-tegnet. En slik fil kan f.eks. se slik ut:

```
1$0,7055475↵2$0,533424↵3$0,5795186↵4$0,2895625↵5$0,301948[EOF]
```

Her er "$" brukt som skilletegn mellom data (fordi tallene inneholder komma) og ↵ betyr linjeskift. *[EOF]* er et sluttmerke, som alle filer har, siden de lagres i blokker og da er det ellers ikke mulig å vite akkurat hvor i blokken filen tar slutt. Slik kan man skrive en slik fil (*utfil* er en åpen *StreamWriter*):

```
For i = 1 To 5
     utfil.Write(i.ToString())
     utfil.Write("$")
     utfil.WriteLine(Rnd().ToString())
Next
```

Alternativt bare én *WriteLine*:

```
For i = 1 To 5
     utfil.WriteLine(i.ToString() & "$" & Rnd().ToString())
Next
```

Vi må altså selv legge inn skilletegnene og bestemme linjeskiftene. Det er selvsagt også mulig å bruke bare *write* og skrive linjeskiftene med *vbNewLine*.

Ad 2 StreamReader

Igjen må du deklarere en variabel som peker til objektet:

```
Dim innfil As System.IO.StreamReader
```

Så må du skape objektet med *New*, f.eks.

```
New System.IO.StreamReader(path As String)
```

Også her er *Path* en fullstendig sti og filnavn (for- og etternavn). Hvis filen ikke finnes, oppstår det en kjørefeil som du fanger med en feilfelle.

Noen egenskaper/metoder for *StreamReader*:

- ✓ *EndOfStream as Boolean* er en egenskap som blir *True* når filen er lest til enden
- ✓ *Close()* er en prosedyre som lukker filen (returnerer ingenting).
- ✓ *Read() As Integer* leser tegn. Returnerer *Integer* som kan omgjøres til et Unicode tegn med *ChrW()*
- ✓ *ReadLine() As String* leser én linje (frem til og med neste linjeskift) og returnerer en streng (uten linjeskiftet bakerst).
- ✓ *ReadToEnd() As String* leser resten av filen (frem til slutten). Returnerer en streng.
- ✓ *BaseStream As Stream* viser til den filen som *StreamReader* har åpen.

Filbuffer
Før VB kan skrive eller lese en fil, må den *åpnes*. Da knyttes filen til programmet, og det lages et *buffer* i RAM (eller tilsvarende sted) slik at man kan lese og skrive større deler av filen – det er mye raskere enn å lese/skrive én og én byte. Når vi f.eks. ber om at det skal leses en byte, så sjekkes det om denne byten ligger i bufferet og i så fall leses det derfra. Hvis byten ikke ligger der, blir en større porsjon av filen lest til bufferet før den ene byten hentes derfra.

En fil som har vært åpnet, må lukkes igjen, ellers vil den bli stående som åpen i filsystemet. (*End* vil fristille filen, men det er ikke sikkert at filen fristilles hvis brukeren bare lukker vinduet.) Når filen lukkes, blir bufferet evt. "tømt". Hvis det ligger noe i bufferet som ikke er lagret, blir det skrevet til filen før filen frigis.

Fillengde
En *StreamReader* har ingen lengde, så vi må henvise til den filen som *StreamReader* peker til. Den heter ***BaseStream***:

```
MsgBox(innfil.BaseStream.Length)
```

Finne slutten på en sekvensiell tekstfil
Siden sekvensielle filer ikke har fast lengde, må vi ofte lese inntil det er slutt på filen, gjerne i en *While*-løkke. Da må vi hele tiden teste om filen er slutt, og til det bruker vi egenskapen *EndOfStream*. F.eks. kan det se slik ut:

```
Do While Not innfil.EndOfStream()
    tekst &= innfil.ReadLine() 'les en linje
Loop
```

Lese tekstfil med skilletegn ("tegnseparert fil")
Hvis man skal lese en tekstfil med skilletegn, må man lese linje for linje, og deretter splitte linjen i de enkelte data (*innfil* er her en åpen *StreamReader*):

```
Dim linje As String
Dim data() As String 'array med strenger - udimensjonert
While Not innfil.EndOfStream
    linje = innfil.ReadLine() 'les en hel linje, men ikke linjeskiftet
    data = linje.Split("$"c) 'splitt linjen i delstrenger ved tegnet "$"
    txtOutput.Text &= data(0) & ": " & data(1) & vbNewLine 'vis dataene
End While
```

Legg merke til at dataene som skilles ut i linjen (her kalt *data(0)*, *data(1)* osv.), er strenger. Hvis vi skal bruke dem som tall, må de konverteres.

Lagring av objekter
Dataene i objekter (attributtverdiene) kan lagres i en tegnseparert tekstfil som forklart ovenfor. Da må verdiene skrives ett for ett til filen, med skilletegn. Når dataene skal leses igjen, må verdiene leses linje for linje, skilles ut fra linjen med *split*. Deretter må vi skape et nytt objekt, og tilordne attributtverdiene fra de utskilte dataene – ofte med konvertering. Dette kan man overlate til hvert objekt, f.eks. med en funksjon *getVerdier* og en annen *saveVerdier*. Uansett er dette svært tungvint.

Det ville være mye enklere om vi kunne lagre objektene direkte, i en fil som lagrer *objekter* istedenfor bare *tekst*. Det er mulig, og vi skal her se på én måte å gjøre det på.

Lagring av objekter i XML-fil
VB har også sin egen måte å lagre objekter på. Da brukes en XML-fil. Det er også en flat tekstfil, men det er føyd til tagger, omtrent som i en HTML-fil. Taggene er tilpasset det objektet som skal lagres på filen. En fil der det er lagret to personer, kan da se slik omtrent slik ut (den er noe forenklet her):

```
<PersonArray>
  <personer>
    <PersonType>
      <navn>Knut</navn>
      <telefon>32117235</telefon>
    </PersonType>
    <PersonType>
      <navn>Erik</navn>
      <telefon>61616161</telefon>
    </PersonType>
  </personer>
</PersonArray>
```

Legg merke til hvordan dette likner på en HTML-fil, men har helt andre tagger.

Objektet må altså gjøres om til en tekststreng med tagger og verdier, som deretter kan skrives til tekstfilen. Det kan man jo gjøre selv, men det er svært mye enklere å bruke en XMLSerializer (*System.Xml.Serialization.XmlSerializer*). Den deklareres slik:

```
Dim xmlSkriver As System.Xml.Serialization.XmlSerializer
```

Deretter må det skapes XMLSerializer-objekter med *New*:

```
xmlSkriver = _
        New System.Xml.Serialization.XmlSerializer(GetType(PersonArray))
```

Her er *PersonArray* en klasse som vi har definert i programmet. *XMLSerializer* bruker vår definisjon til å lage de taggene som passer til objektet av denne klassen.

XMLSerializer har bare én metode som er nyttig for oss når objektene skal skrives til fil:

- ✓ *Serialize (StreamWriter-objekt, objekt som skal skrives)* som gjør om objektet til en tekststreng etter XML-syntaks og skriver strengen til filen ved hjelp av den oppgitte *StreamWriter*. Dette er en prosedyre og gir ingen returverdi.

Som det fremgår, har metoden *Serialize* to argumenter (parametre), nemlig et *StreamWriter-objekt* og et objekt.

Ad argumentet StreamWriter-objekt

Man må altså ha et *StreamWriter*-objekt, for å kunne skrive til filen. Det er en helt vanlig *StreamWriter* og den skapes og knyttes til en åpen fil på samme måte som vist ovenfor i avsnittet om tekstfiler, f.eks.

```
Dim utfil As System.IO.StreamWriter
utfil = New System.IO.StreamWriter(sti & filnvn, False) 'ikke append
```

Ad argumentet objektet som skal skrives

Spesielt for XML-filer er at en slik fil bare kan inneholde ett objekt. Hvis vi skal skrive ut flere, må vi derfor først lage en overordnet klasse som inneholder alle som vi vil lagre i en array. Denne lager vi et objekt av. I dette objektets array legger vi referanse til de objektene vi egentlig vil ha lagret. Her er et eksempel, når objektene representerer personer med navn og telefon.

1) Definer person-klassen:

```
Class PersonType
    Public navn As String
    Public telefon As Integer
End Class
```

2) Definer en klasse som har en array med personer av denne typen:

```
Class PersonArray
    Public personer(maxPerson) As PersonType
End Class
```

3) Skap en personarray i programmet og skap alle personene i arrayen:

```
Dim i As Integer
Dim person As PersonType = New PersonType()
Dim tblPerson As New PersonArray()
For i = 0 To maxPerson
    tblPerson.personer(i) = New PersonType()
Next
```

4) Gi personene verdi, f.eks. person nr 4:

```
With tblPerson.personer(4)
    .navn = "Erik"
    .telefon(0) = 61616161
End With
```

Når du har laget en *XMLSerializer* og en *StreamWriter* som er tilknyttet til en åpen fil, kan du skrive objektet til XML-filen slik:

```
    xmlSkriver.Serialize(utfil, tblPerson)
```

Lesing av objekt fra XML-fil

Når objekter skal leses fra en XML-fil, gjør man omtrent det samme som når man leser. Forskjellen er at man må ha en *StreamReader* og bruker metoden *Deserialize()* fra *XMLSerializer*. Når objektene leses med *Deserialize()*, må de dessuten konverteres til den arrayen som vi lagrer objektene i.

Først deklarerer vi en *XMLSerializer* og skaper den:

```
    Dim xmlLeser As System.Xml.Serialization.XmlSerializer
    xmlLeser = _
        New System.Xml.Serialization.XmlSerializer(GetType(PersonArray))
```

Deretter må vi deklarere og skape en *StreamReader* til å lese XML-filen:

```
    Dim innfil As System.IO.StreamReader
    innfil = New System.IO.StreamReader(sti & filnvn)
```

XMLSerializer har bare én metode som er nyttig for oss når objektene skal leses fra fil:

- ✓ *Deserialize (StreamReader-objekt) As Object* som leser en XML-fil med den oppgitte *StreamReader* og returnerer bare ett objekt.

Det objektet som *Deserialize()* returnerer, er av den generelle klassen *Object*. Det må konverteres til et objekt av den klassen vi lagret. En slik konvertering av objektet gjør vi med funksjonen *CType()*

Som det fremgår, har metoden *Deserialize* bare ett argument, nemlig en *StreamReader*.

Ad argumentet StreamReader-objekt

Man må altså ha et *StreamReader*-objekt, for å kunne lese fra filen. Det er en helt vanlig *StreamReader* og den skapes og knyttes til en åpen fil på samme måte som vist ovenfor i avsnittet om tekstfiler, f.eks.

```
    Dim innfil As System.IO.StreamReader
    innfil = New System.IO.StreamReader(sti & filnvn)
```

Ad objektet som er lest

Siden en XML-fil bare kan inneholde ett objekt, er det bare ett objekt som returneres når filen leses. Hvis vi har flere, må de derfor lagres i en array i en overordnet klasse. Nedenfor er et eksempel, gitt samme objekter som ovenfor (vi leser de samme objektene som vi skrev ut ovenfor). Den benytter

konverteringsfunksjon *CType(uttrykk, typenavn) As typenavn*[14] som konverterer uttrykket til den angitte typen – her et objekt av *PersonArray*-klassen:

```
Dim lestObjekt As Object = New Object()
lestObjekt = xmlLeser.Deserialize(innfil)
tblPerson = CType(lestObjekt, PersonArray)
```

Alternativt kan vi konvertere direkte, uten å gå veien om en egen Objekt-variabel:

```
tblPerson = CType(xmlLeser.Deserialize(innfil), PersonArray)
```

[14] *CType(expression, typename)* er den generelle varianten av konverteringsfunksjonene *CDate(expression)*, *CDbl(expression)*, *CInt(expression)*, *CLng(expression)*, *CStr(expression)* osv. Mange av disse har du sikkert brukt før.

Eksempelprogram

```
Option Strict On
Option Explicit On
Public Class frmFiler
    Inherits System.Windows.Forms.Form
    Private Sub butTekstfil_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butTekstfil.Click
        'Viser skriving og lesing til/fra en ren sekvensiell tekstfil
        Dim sti As String = My.Application.Info.DirectoryPath
        Dim filnvn As String = "\Sekvensiell.txt"
        Dim i As Integer

        Try
            'A - Skriv ren tekstfil
            '======================
            'a - Åpne filen for skriving
            Dim utfil As System.IO.StreamWriter
            utfil = New System.IO.StreamWriter(sti & filnvn, False) 'ikke append

            'b - Skriv linjeorientert tekst til filen
            utfil.Write("A"c) 'skriv ett tegn uten linjeskift (c for char)
            utfil.Write(ChrW(65)) 'skriv ett tegn til
            For i = 1 To 5
                utfil.Write(i.ToString()) 'skriv et tall uten å ta linjeskift
                utfil.Write(": ") 'skriv en tekst uten linjeskift
                utfil.WriteLine("Tilfeldig: " & Rnd().ToString()) 'skriv linje
            Next

            'd - Lukk filen
            utfil.Close()

            'B - Les ren tekstfil
            '====================
            'a - Åpne filen for lesing
            Dim innfil As System.IO.StreamReader
            innfil = New System.IO.StreamReader(sti & filnvn)

            'b - Les fra filen
            txtOutput.Text = ChrW(innfil.Read()) ' les ett tegn
            txtOutput.Text &= ChrW(innfil.Read()) ' les ett tegn
            Do While Not innfil.EndOfStream()
                txtOutput.Text &= innfil.ReadLine() 'les en linje
                txtOutput.Text &= vbNewLine
            Loop
            txtOutput.Text &= ("Fillengde: " _
                    & innfil.BaseStream.Length.ToString())

            'c - Lukk filen
            innfil.Close()

        Catch ex As Exception
            MsgBox(ex.Message())
        End Try
    End Sub
```

```
Private Sub butMedSkilletegn_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butMedSkilletegn.Click
        'Skriver/leser kommaseparert fil linje for linje
        Dim sti As String = My.Application.Info.DirectoryPath
        Dim filnvn As String
        Dim i As Integer

        'Housekeeping
        txtOutput.Clear()

        Try
            'A - Skriv tegnseparert fil
            '==========================
            'a - Åpne filen som en Stream
            Dim utfil As System.IO.StreamWriter
            filnvn = "\MedSkilletegn.txt"
            utfil = New System.IO.StreamWriter(sti & filnvn, False)
            'b - Skriv til filen
            For i = 1 To 5
                utfil.Write(i.ToString())
                utfil.Write("$")
                utfil.WriteLine(Rnd().ToString())
                'alternativt:
                'utfil.WriteLine(i.ToString() & "$" & Rnd().ToString())
            Next
            'c - Lukk filen
            utfil.Close()

            'B - Les fra tegnseparert fil
            '============================
            'a - Åpne filen som en Stream
            Dim innfil As System.IO.StreamReader
            filnvn = "\MedSkilletegn.txt"
            innfil = New System.IO.StreamReader(sti & filnvn)
            'b - Les tegnseprarert fil
            Dim linje As String
            Dim data() As String 'array med strenger - udimensjonert
            While Not innfil.EndOfStream
                linje = innfil.ReadLine() 'les en linje, ikke linjeskiftet
                data = linje.Split("$"c) 'splitt linjen i delstrenger ved "$"
                txtOutput.Text &= data(0) & ": " & data(1) & vbNewLine 'vis
            End While
            'c - Lukk filen
            innfil.Close()
            'C - Les hele filen som ren tekst og vis i vinduet
            '================================================
            'a - Overskrift
            txtOutput.Text &= vbNewLine & _
                "INNHOLDET AV MEDSKILLETEGN.TXT-FILEN" & vbNewLine
            'b - Åpne filen som Stream
            filnvn = "\MedSkilletegn.txt"
            innfil = New System.IO.StreamReader(sti & filnvn)
            'c - Les hele filen på en gang og vis i vinduet
            txtOutput.Text &= innfil.ReadToEnd()
            'd - Lukk filen
            innfil.Close()

        Catch ex As Exception
            MsgBox(ex.Message())
        End Try
    End Sub
```

```
'#####################################
'# Bruker XML-fil til å lagre objekter
'# Demo - ikke pensum
'#####################################
Class PersonType
    Public navn As String
    Public telefon As Integer

    Public Overrides Function Tostring() As String
        Dim tekst As String
        tekst = LSet(navn, 10)
        tekst &= " Tlf: " & LSet(telefon.ToString(), 8)
        Return tekst
    End Function
End Class

Class PersonArray
    Public personer(4) As PersonType
End Class

Private Sub butObjektfil_Click(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) Handles butObjektXML.Click
    Dim sti As String = My.Application.Info.DirectoryPath
    Dim filnvn As String = ""
    Dim i As Integer
    Dim tblPerson As New PersonArray()

    'Housekeeping
    txtOutput.Clear()
    For i = 0 To 4
        tblPerson.personer(i) = New PersonType()
    Next

    'Gi noen personobjekter verdi
    tblPerson.personer(0) = New PersonType
    tblPerson.personer(0).navn = "Knut"
    tblPerson.personer(0).telefon = 32117235

    'Blir du lei av å skrive samme navn etter hverandre mange ganger,
    'kan du bruke with:
    tblPerson.personer(4) = New PersonType
    With tblPerson.personer(4)
        .navn = "Erik"
        .telefon = 61616161
    End With

    Dim xmlSkriver As System.Xml.Serialization.XmlSerializer
    Dim utfil As System.IO.StreamWriter
    Dim xmlLeser As System.Xml.Serialization.XmlSerializer
    Dim innfil As System.IO.StreamReader

    Try
        'A - Skriv objektene til en XML-fil
        '==================================
        'a - Lag en XML-skriver
        xmlSkriver = _
        New System.Xml.Serialization.XmlSerializer(GetType(PersonArray))
        'b - Åpne filen som en Stream
        filnvn = "\Personer.xml"
        utfil = New System.IO.StreamWriter(sti & filnvn, False)
        'c - Skriv XML-kode til filen
        xmlSkriver.Serialize(utfil, tblPerson)
        'd - Lukk filen
        utfil.Close()
```

```
            'B - Les objektene fra XML-fil
            '============================
            'a - Slett alle objektene i RAM for sikkerhets skyld
            For i = 0 To 4
                tblPerson.personer = Nothing
            Next
            'b - Lag en XML-leser
            xmlLeser = New _ System.Xml.Serialization.XmlSerializer _
                 (GetType(PersonArray))
            'c - Åpne filen som en Stream
            filnvn = "\Personer.xml"
            innfil = New System.IO.StreamReader(sti & filnvn)
            'd - Les objektene og gjør dem om til en objektarray
            tblPerson = CType(xmlLeser.Deserialize(innfil), PersonArray)

            'e - Lukk filen
            innfil.Close()

            'C - Vis objektene i vinduet
            '==========================
            'a - Overskrift
            txtOutput.Text = _
                "POSTENE i RAM etter lesing fra PERSONER.XML-FILEN:" _
                 & vbNewLine
            'b - Vis alle objektene
            For i = 0 To 4
                txtOutput.Text &= "Index " & LSet(i.ToString(), 3)
                txtOutput.Text &= tblPerson.personer(i).Tostring & vbNewLine
            Next

            'D - Les XML-filen som ren tekst og vis i vinduet
            '===============================================
            'a - Overskrift
            txtOutput.Text &= vbNewLine & "INNHOLDET AV PERSONER.XML-FILEN" & _
                vbNewLine
            'b - Åpne filen som Stream
            filnvn = "\Personer.xml"
            innfil = New System.IO.StreamReader(sti & filnvn)
            'c - Les hele filen på en gang og vis i vinduet
            txtOutput.Text &= innfil.ReadToEnd()
            'd - Lukke filen
            innfil.Close()
        Catch ex As Exception
            MsgBox(ex.Message())
        End Try
    End Sub
End Class
```

Feilfeller

Når programmet kompileres, så kontrolleres *syntaksen*, dvs. (a) de ord som er brukt og (b) måten de er satt sammen på. Når programmet kjører, er det under kontroll av *Run Time System*. Kritiske feil som oppdages da, fører til at brukeren får en feilmelding og programmet stopper. Det er både irriterende og kan være katastrofalt for brukeren. Det kalles *kjørefeil* (eng. *Run Time Error*). Det må vi altså unngå!

Vi kan gjøre mye når vi skriver programmet, f.eks. ved å sjekke at en input er et tall før vi prøver å konvertere det (*IsNumeric* eller *TryParse*), sjekke at en streng ikke er tom osv. Likevel er det ikke alltid mulig å kontrollere alt på forhånd. Vi kan f.eks. sjekke at en fil finnes før vi prøver å åpne

den, men vi risikerer likevel at en annen bruker har slettet den før vi rekker å åpne. Da oppstår en kjørefeil.

Fange kjørefeil

For å fange opp slike kjørefeil, slik at brukeren ikke får stopp, setter du opp en såkalt *feilfelle* (eng. *error trap*) med *Try/Catch/Finally*.

Syntaks:

```
Try
  .. prøv noe
Catch Ex as Exception
  .. håndter eventuelle kjørefeil
Finally
  .. gjør noe uansett feil eller ikke feil
End Try
```

Finally er frivillig.

Forklaring:

Når programmet kjøres, skjer følgende:

1) *Try*: Maskinen forsøker å gjøre det som er beskrevet under ”Try”.
   Hvis det går bra, hopper maskinen til ”Finally” (eller til ”End Try” hvis ”Finally” ikke finnes).
   Hvis det oppstår en kjørefeil, lager *Run Time System* et Exception-objekt og hopper til ”Catch”
2) *Catch*: Hvis programmet kommer til ”Catch” har det oppstått en kjørefeil. Da gjøres det som er beskrevet under ”Catch”. Man kan utnytte Exception-objektet til å finne ut hva som skjedde. Du kan kalle objektet hva du vil, men VB foreslår ”Ex”.
3) *Finally*: Uansett om programmet kom innom ”Catch” eller ikke, så utføres til slutt det som står beskrevet under ”Finally” (hvis det står der).

Eksempel (programmet ”Feilfelle”):

```
Private Sub butCDbl_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butCDbl.Click
    Dim tall As Double = 0
    Dim melding As String = ""
    lblResultat.Text = ""
    Try '1: Prøv noe som lett kan gi feil
        tall = CDbl(txtInput.Text)
        lblResultat.Text = tall.ToString()
        melding = "Alt vel"
    Catch ex As Exception '2: Fang evt. feilen
        melding = "Feil i CDbl: " & vbNewLine & ex.Message
    Finally '3: Gjør noe UANSETT - vil bli utført FØR evt. Exit Sub
        MsgBox(melding, MsgBoxStyle.Information, "Resultat")
    End Try
End Sub
```

Husk at du – som programmerer – ikke kan være sikker på at tallet faktisk ble konvertert. Det må du ta hensyn til i den videre programmeringen. Programmet fortsetter i alle fall uten ”bråstopp” for kjørefeil, og brukeren som kjører programmet får en feilmelding (på engelsk).

Pass også på at variable som er deklarert inne i en feilfelle, bare finnes inne i feilfellen.

Mer avansert: Kaste feil selv

Du kan skape dine egne kjørefeil når du oppdager en feil i en prosedyre. Da avbrytes utførelsen av prosedyren med en gang. Da må du selv skape et exception-objekt og ”kaste” feilen. Slik kan det se ut:

```
Private Sub butKastFeil_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butKastFeil.Click
    'Lager ("kaster") en feil
    'Prøv å gjøre noe (som kan gi feil)
    Try
        If IsNumeric(txtInput.Text) Then 'skal være numerisk
            MsgBox("Tallet var " & txtInput.Text, MsgBoxStyle.Information)
        Else 'txtInput.Text var et tall
            Throw New Exception("Feil: Input var ikke et tall") 'kast feil
        End If
    Catch ex As Exception
        MsgBox(ex.Message, MsgBoxStyle.Exclamation)
    End Try
End Sub
```

butKastfeil_Click "kaster" eventuelt en feil som lages "i farta" med *New*. Den "fanges" av *Catch* som gir feilmeldingen til brukeren.

Man kan også gjerne deklarere feilen først, og tilordne den ved behov:

```
Dim feil As Exception
feil = New Exception("Feil: Input var ikke et tall")
Throw feil
```

Legg merke til at *Ex.Message* henter exception-objektets melding. Denne egenskapen kan bare leses. Derfor får du *ikke* lov til å skrive

```
Ex.Message = "en tekst"
```

Isteden må du gi feilen en meldingstekst når du skaper den:

```
Ex = New Exception("en tekst")
```

Hvis man vil vite mer om feilen, kan man også skrive ut feilens *StackTrace*:

```
MsgBox("Feil i Kast en feil: " & vbNewLine & ex.Message & ex.StackTrace)
```

Dette er nok ikke opplysninger som brukeren er interessert i, men fint for feilfinning under testing.

Feil i objekter

Når det oppstår feil i et objekt som du selv har laget, bør du ikke la objektet selv skrive ut feilmeldingen i en meldingsboks, men heller returnere feilen til hovedprogrammet som tar seg av brukeren. Da blir klassen du lager letter å bruke i andre programmer.

Du lager da et feilobjekt, slik det er vist ovenfor, og kaster feilen. Hovedprogrammet må da fange feilen som oppstår, og selv håndtere den.

Her er det deklarert en klasse *Student* der *New* kan kaste en feil:

```
Public Class Student
    Private navn As String
    Public Sub New(ByVal navn As String)
        'Konstruktør
        If navn Is Nothing Or navn = "" Then 'ikke akseptabelt navn
            Throw New Exception("Navnet er tomt eller ingenting")
        Else 'navnet er tilsynelatende OK
            Me.navn = navn
        End If
    End Sub
End Class
```

Hovedprogrammet har en knapp som forsøker å skape en slik student, og det må fange feilen:

```
    Private Sub butStudent_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles butStudent.Click
        'Skaper en student og håndterer feil
        Try
            Dim stud As New Student(txtInput.Text)
            MsgBox("Studenten ble skapt og fikk navnet " & txtInput.Text)
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub
```

Tema C1 – Prosedyrer og funksjoner

Tema C2 er litt "snadder" – se nedenfor.

Prosedyrer og funksjoner er *navngitte blokker av kode*. Andre deler av programmet kan *kalle på dem* og kan da oppgi verdier – kalt *argumenter* – som prosedyren/funksjonen trenger for å kunne gjøre det den skal. Forskjellen på prosedyre og funksjon, er at sistnevnte tilordner en verdi (*funksjonsverdien*) som den *returnerer*[15].

Hensikt

Ved å skrive slike navngitte kodeblokker, oppnår vi mange fordeler, bl.a.

- ✓ Programmet blir delt opp i mindre deler. Det er en fordel fordi
  - Det gir bedre oversikt – en fordel både ved programmeringen, feilfinning, testing og vedlikehold. F.eks. bruker mange en regel om at ingen programdel skal være lenger enn en A4-side på utskriften (noen tillater inntil én A4-side med deklarasjoner og inntil én A4-side med algoritme).
  - Det er enklere å dele programmerings- og vedlikeholdsarbeid på flere personer.
- ✓ En gruppe kodelinjer skrives bare ett sted, selv om de brukes mange steder i programmet. En tommelfingerregel kan være at ”man skriver aldri samme kodeblokk to ganger”. Selvsagt kan man skrive samme setning flere steder i programmet, men man bør unngå å gjenta en lengre blokk med kode. Fordelen er bl.a. at
  - Vedlikehold og feilretting medfører retting av koden bare ett sted
  - Programmet virker likt, uansett hvordan det kom til denne kodeblokken. Altså f.eks. at hver gang brukeren ber om lagring av data, får han det samme grensesnittet. Det gjør at brukeren kjenner seg igjen og lærer raskere. Det kalles ”helhetlighet” (wholeness).

Sideeffekter

Metoder kan ha *sideeffekter*. Det innebærer at de gjør noe mer enn bare å ta imot verdier (argumenter) og returnere én eller flere verdier. F.eks. kan de hente data fra brukeren, med *InputBox* eller et dialogvindu. De kan hente data fra ytre enheter, f.eks. filer eller instrumenter, og de kan hente data fra konstanter og variable som er deklarert utenfor metoden, men synlige inne i dem, inkludert egenskaper for kontroller på eget eller andre skjemaer. De kan også gi data til brukere (meldingsboks, andre skjemaer), lagre til fil, sende data/signaler til ytre enheter og endre verdien på variabel utenfor metoden, inkludert egenskaper for kontroller på eget eller andre skjemaer.

Slike sideeffekter kan skape vanskeligheter, hvis de ikke tydelig vises i navnet. Eksempler på at sideeffekten er tydelig fra utsiden – og følgelig allikevel OK – er *lagreTilFil(data,filnavn)*, *visMelding(feilmelding)* og *åpneFil(filnavn)*. Hvis sideeffekten ikke er tydelig i navnet og metoden behandles som en ”sort boks” – det vil si at vi ikke ser på koden inne i dem – vil sideeffekter være usynlige og kan være overraskende. Det kan gjøre det vanskelig å finne feil. Derfor bør man være varsom med slike sideeffekter.

Metodene kan også kalle på andre metoder for å få ”hjelp til å få jobben gjort”. Det er effektivt, men hvis de påkalte metodene har sideeffekter, kan feil bli svært vanskelige å finne.

Unngå uansett at et objekt i en klasse som dere selv har definert, kommuniserer med brukeren som sideeffekt. *All* kommunikasjon med brukeren bør skje gjennom skjemaobjektet ("hovedprogrammet"). Hvis et annet objekt må melde en feil, bør det *kaste* feilen og la skjemaet

[15] Begrepene er her svært uklare. Microsoft skiller f.eks. mellom *SubRoutines* (Sub) og *Functions*, og kaller dem samlet for *Procedures*. Java v/Sun kaller alt for *Functions* – de som ikke gir returverdi, sies å returnere *void* (tomhet). Andre kaller alt for *Procedure* uansett, og mener at funksjoner bare er en variant (en subtype) av prosedyrer. For klasser og objekter brukes ofte samlebegrepet *metoder* om prosedyrer og funksjoner. For å forenkle teksten, bruker jeg *metode* på denne måten.

fortelle brukeren om det (eller ordne opp selv). Da blir klassen din mer gjenbrukbar og meldingene til brukeren helhetlige.

Parametre og argumenter

Ofte må metoden ha oppgitt noen verdier for å kunne gjøre det den skal. Disse verdiene kalles *parametre* eller *argumenter*[16]. Opprinnelig var det forskjell på de to begrepene, nemlig at

- ✓ *parameter* er en verdi som metoden må ha – det må være en variabel. Vi deklarerer den inne i parentesene når metoden deklareres: *åpneFil(ByVal filnavn As String)*
- ✓ *argument* er den verdien som metoden faktisk får overført når metoden kalles – det kan være både en variabel og en konstant. Vi oppgir dem inne i parentesen når metoden brukes: *åpneFil("\navnefil.dat")*.

I dag brukes begrepene om hverandre som synonymer (og det gjør også jeg, men Microsoft opprettholder skillet). For skille mellom dem, sier vi da *formelt parameter/argument* når vi mener verdien metoden må ha, og *aktuelt parameter/argument* når vi mener den verdien den faktisk får i et gitt tilfelle. (Nedenfor vil jeg prøve å presisere *formelt parameter* og *aktuelt argument*.)

Når vi som eksempel ser på metoden *InputBox*, så er den deklarert slik av Microsoft:

```
Public Function InputBox( _
    ByVal Prompt As String, _
    Optional ByVal Title As String = "", _
    Optional ByVal DefaultResponse As String = "", _
    Optional ByVal Xpos As Integer = -1, _
    Optional ByVal YPos As Integer = -1 _
    ) As String
```

De formelle parametrene er de variablene som er listet opp inne i parentesen: *Prompt*, *Title*, *DefaultResponse*, *Xpos* og *Ypos*. For hver av dem er det angitt en datatype, slik at kompilatoren kan kontrollere at vi bruker riktig datatypen når vi kaller på den. Den siste datatypen *String* angir at den returverdien vi får er en streng. De fire siste parametrene er merket *Optional*, hvilket betyr at vi ikke er nødt til å oppgi dem i kallet. Da vil de få den angitte standardverdien. Et formelt parameter som er merket *Optional* men uten standardverdi, vil bli initialisert på vanlig måte (tall blir satt til 0, strenger til en tom streng og Boolske til False).

Her en et eksempel på kall på *InputBox*:

```
Dim Navn As String
Navn = InputBox("Hva heter du", "Heisann!")
```

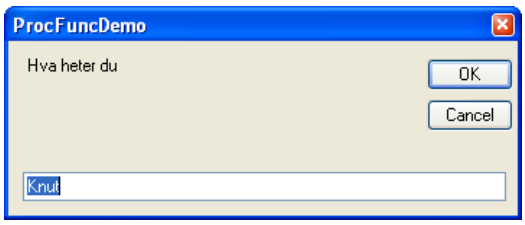
I kallet har vi oppgitt to strenger som aktuelle argumenter. Den første tilsvarer *Prompt* og er obligatorisk. Den andre er *Title* og er frivillig. *InputBox* har ytterligere tre parametre, men de er frivillige og vi har valgt å ikke oppgi noe aktuelt argument for dem. Da brukes standardverdien for dem. Verdien av de aktuelle argumentene settes inn i de formelle parametrene, slik at verdien av *Prompt* blir ”Hva heter du?” og verdien av *Title* blir ”Heisann!”.

Vi kan også ”hoppe over” formelle parametre som vi ikke vil gi verdi, f.eks.

```
Navn = InputBox("Hva heter du", , "Knut")
```

De to kommaene angir at vi ikke ønsker å oppgi noen verdi for *Title*, men vi *har* oppgitt det tredje parameteret *DefaultResponse*. Inputboksen vil se slik ut:

[16] Du kan lese en god forklaring på dette på Wikipedia http://en.wikipedia.org/wiki/Parameter_%28computer_science%29



ByVal og ByRef

Metoder kan få de aktuelle argumentene på to måter:

- ✓ Som en *referanse* der de får vite hvor den aktuelle verdien er lagret i RAM. Da kan metoden endre den der den er lagret – argumentet blir på en måte "globalt" i forhold til metoden. Når den da endres, vil endringen også gjelde utenfor metoden.
- ✓ Som en *verdi* der de bare får vite verdien, men ikke hvor den er lagret. Da kan metoden ikke endre argumentet der det er lagret, for det er ukjent. Argumentet blir en "lokal variabel" i forhold til metoden. Den kan gjerne endres – den er jo en variabel – men det får ingen virkning utenfor metoden.

Her er et eksempel. Prosedyren som bytter om to strenger har formelle parametre med *ByRef*:

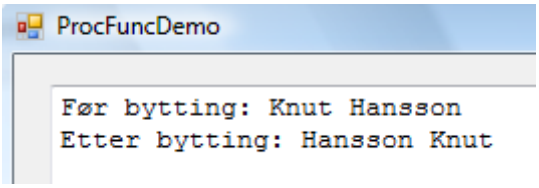
```
Public Sub byttTekst(ByRef a As String, ByRef b As String)
    'Bytter om to strenger
    'Parametrene må være ByRef for at det kallende program
    'skal "merke" effekten
    Dim tmp As String
    tmp = a : a = b : b = tmp
End Sub
```

Lokalt i denne prosedyren, kalles strengene hhv *a* og *b*.

Den brukes i denne hendelsesprosedyren, som skal bytte to tekster:

```
Private Sub butDemo1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butDemo1.Click
    'Deklarerer og initierer to variable
    Dim første As String = "Knut"
    Dim siste As String = "Hansson"
    txtPrint.Text = "Før bytting: " & første & " " & siste & vbCrLf
    'Kaller prosedyren
    byttTekst(første, siste)
    'Ser på resultatet
    txtPrint.Text &= "Etter bytting: " & første & " " & siste
End Sub
```

I hendelsesprosedyren kalles strengene hhv *første* og *siste*. I kallet overføres adressene til *første* og *siste* til prosedyren *byttTekst* der de får navnene *a* og *b* – men det er *samme minneadresse*. Når prosedyren da bytter *a* og *b* med standardalgoritme, bytter den verdien i RAM, som i den kallende modul heter *første* og *siste*. Ombyttingen får derfor effekt utenfor *byttTekst* også. Utskriftene fra hendelsesprosedyren blir slik:



Det er også mulig å oppgi argumenter med navnet på parameteret. Da kan vi angi argumentene i vår egen rekkefølge, f.eks.

```
Navn = InputBox(DefaultResponse:="Knut", Prompt:="Hva heter du")
```

Her er altså argumentene i feil rekkefølge, og flere (frivillige) argumenter mangler. Det gjør ingenting, fordi de er navngitt så *InputBox* kan finne dem igjen. Det er ikke så vanlig å navngi argumenter i programmering, men det gjøres alltid i HTML, f.eks.:

```
<a href="http://www.hibu.no" target="_blank">
```

Signatur

Som kjent er en fil ikke identifisert bare ved sitt filnavn. Filens sti må også være med og utgjør en del av filens fulle identifikasjon. Tilsvarende er en metode ikke identifisert bare ved navnet. Den fulle identifikasjonen – som kalles metodens *signatur* – inkluderer mer.

Dette utgjør til sammen metodens signatur[17]:

- ✓ Navn
- ✓ Formelle parametres rekkefølge og datatype

Dette er altså *ikke* med i signaturen:

- ✓ Synlighet (*scope*), som *Public*, *Protected*, *Friend*, *Private*
- ✓ Metodens levetid: *Shared*
- ✓ Formelle parametres navn
- ✓ Aktuelt arguments overføringsmetode: *ByRef* og *ByVal*
- ✓ Aktuelt arguments valgfrihet: *Optional*
- ✓ Om metoden returnerer en verdi (det skilles altså ikke mellom funksjon og prosedyre som ellers er like)
- ✓ Datatypen for evt. returverdi
- ✓ Evt. hendelser som prosedyren håndterer: *Handles*

To metoder kan bare ha samme signatur, hvis de ikke er synlige for hverandre, eller hvis den ene skal erstatte (overstyre) den andre ved arv. I siste tilfelle må den overordnede være merket *Overridable* og den nye må merkes *Overrides*. Det kan være mange grunner til at to metoder ikke er synlige for hverandre, f.eks. kan de være deklarert i hver sin klasse, på hvert sitt skjema, eller i hvert sitt navnerom (*NameSpace*). Poenget er at kompilatoren må vite hvilken av dem som skal brukes ved et kall.

Anta at vi har skrevet denne deklarasjonen:

```
Private Function GjørNoe(ByVal x As Double) As Short 'utganspunkt
```

Signaturen for denne er *GjørNoe(Double)*.

Disse har da samme signatur:

```
Public Function GjørNoe(ByVal x As Double) As Short 'Public
Private Function GjørNoe(Optional ByVal x As Double = 7) As Short 'Optional
Private Function GjørNoe(ByRef x As Double) As Short 'ByRef
Private Sub GjørNoe(ByVal x As Double) 'Sub
Private Function GjørNoe(ByVal y As Double) As Short 'Parameternavn
Private Function GjørNoe(ByVal x As Double) As Long 'Returtype
Private Shared Function GjørNoe(ByVal x As Double) As Long 'Shared
```

Disse er også like, fordi de to datatypene *Integer* og *Int32* er synonymer:

```
Private Shared Function GjørNoe(ByVal x As Integer) As Long
Private Shared Function GjørNoe(ByVal x As Int32) As Long
```

Disse har en annen signatur:

```
Private Function GjørNoeAnnet(ByVal x As Double) As Short 'Navn
Private Function GjørNoe(ByVal x As Long) As Short 'Parametertype
Private Function GjørNoe(ByVal x As Long, ByVal y As Short) As Short 'Antall
```

[17] Her har jeg unnlatt å ta med konverteringsoperatorer, som det gjelder litt spesielle regler for

Disse er også forskjellige, på grunn av parametrenes rekkefølge:

```
Private Function GjørNoe(ByVal x As Long, ByVal y As Short) As Short
Private Function GjørNoe(ByVal y As Short, ByVal x As Long) As Short
```

De to signaturene er henholdsvis *GjørNoe(Long, Short)* og *GjørNoe(Short, Long)*.

Analogi: Metoden som en datakiosk

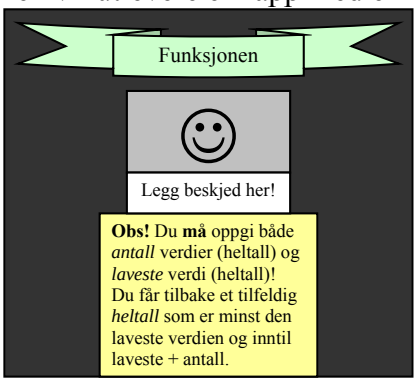
Jeg liker å tenke på metoder som en kiosk. Kiosken er svartmalt (sort boks = "black box") så vi ikke kan se hva som skjer innenfor. Den er merket med navn, og har en luke, der du kan legge en lapp med de argumentene du vil ha brukt. Hvis argumentet er frivillig (*Optional*) kan du la være å oppgi det, ellers må du enten oppgi argumentene i riktig rekkefølge eller med navn, og de må være av riktig type. Noen argumenter vil kiosken ha *adressen* til (*ByRef*), andre vil den ha *verdien* til (*ByVal*).

Hvis alt går vel, utfører kiosken det som er angitt i navnet. Du kan oppleve – når du ser etter – at de argumentene du oppgav adressen til, er blitt forandret. Hvis det er en funksjon, vil du få en lapp i retur ut av luken, med en verdi (funksjonsverdien) på, og den vil være av den typen som er angitt.

Nedenfor ser vi en slik kiosk. Den er deklarert slik i VB:

```
    Private Function Random( _
        ByVal antall As Integer, _
        ByVal laveste As Integer _
        ) As Integer
    End Function
```

De to argumentene som vi må skrive på lappen som vi legger i luken, er angitt inne i parentesen, etter navnet på metoden. Det fremgår at kiosken vil ha *verdier* (ikke *adresser*) og at begge må være *heltall*. Videre fremgår det at kiosken vil utlevere en lapp med en heltallsverdi på.



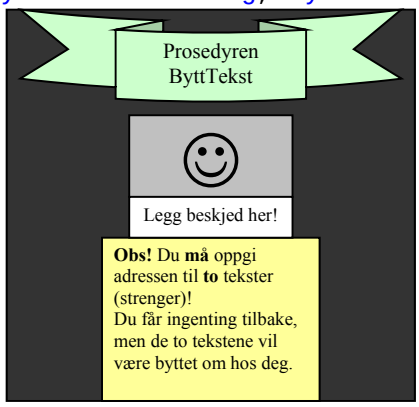
Hvis vi da skriver (6, 1) på en lapp og legger den inn i luken til *Random*, få vi tilbake en lapp med en verdi fra 1 til 6. Det kan se slik ut:

```
        Dim terning As Integer = Random(6, 1)
```

terning får en verdi i intervallet [1..6]

Her er en annen kiosk. Denne gang er det kiosken ByttTekst som vil ha to argumenter, og begge skal være adressen til en streng. Den er deklarert slik:

```
Private Sub ByttTekst(ByRef a As String, ByRef b As String)
```

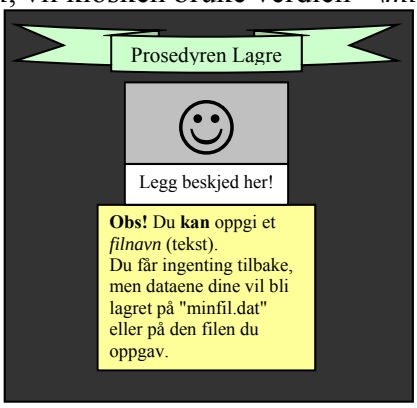


Denne kiosken vil ha *adressen* til to tekster, og gir ingenting tilbake. Siden kiosken får vite hvor vi har lagret tekstene våre, kan den likevel bytte dem om. Programmet og bruken er gjengitt som eksempel ovenfor.

I kiosken nedenfor – som har en sideeffekt – er det et frivillig (*Optional*) argument:

```
Private Sub Lagre(Optional ByVal fil As String = "\minfil.dat")
```

Hvis du ikke oppgir argumentet *fil*, vil kiosken bruke verdien *"\minfil.dat"*.



Det er full anledning til å ha strukturerte data som argumenter, parametre og funksjonsverdier, f.eks. objekter og arrays. Denne funksjonen håndterer arrays:

```
Private Function array(ByVal noe() As Integer) As Integer()
    ... gjør ett eller annet med arrayen noe
    Return noe
End Function
```

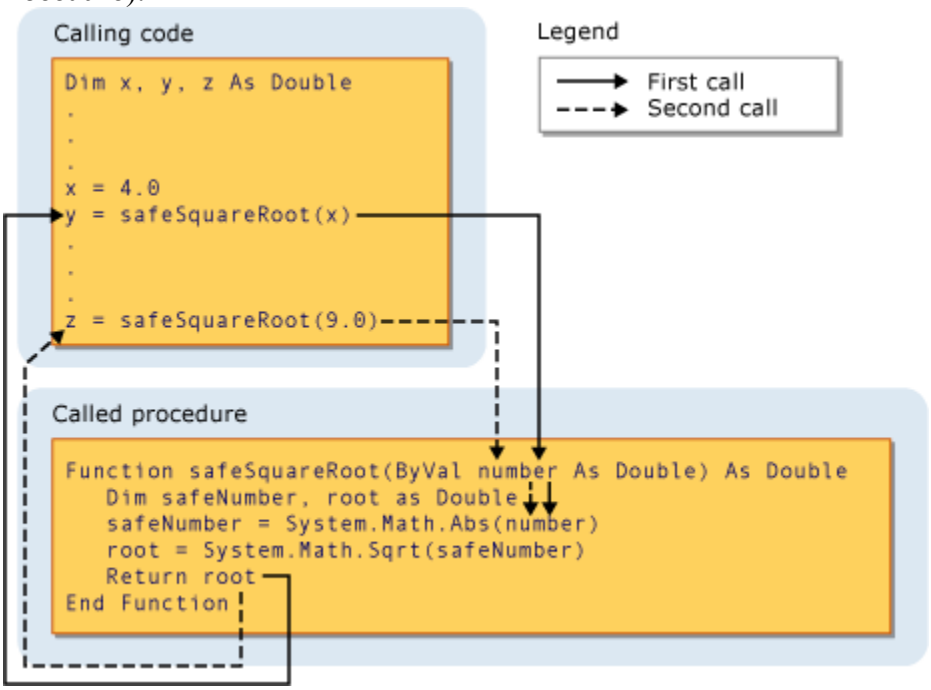
Som du ser er syntaksen for arrayene uten størrelse. Da gjelder dette for alle arrayer uansett størrelse. Parameteret er her *ByVal*. Det må derfor "skrives en lapp" med alle verdiene i arrayen på som argument. Tilsvarende må "kiosken" skrive en lapp med alle array-verdiene på for å gi en retur.

Dette vil være tidkrevende. De fleste ville derfor vanligvis deklarert metoden slik:

```
Private Sub array2(ByRef noe() As Integer)
```

Argumentet er da én *adresse til arrayen*, istedenfor alle verdiene i den. Da kan kiosken bruke samme minneområde som deg selv, og manipulere med verdiene der. Slik slipper maskinen mye arbeid med overføring av verdiene i arrayen, men på den annen side innfører vi en sideeffekt.

VB hjelp har denne tegningen av hva som foregår nå en funksjon kalles (de bruker her sitt samlebegrep *Procedure*):



Hendelsesprosedyrer

I programmer som skal reagere på hendelser, må vi programmere hva som skal gjøres når denne hendelsen inntreffer. Det gjør vi i en *hendelsesprosedyre*. Det er en helt vanlig prosedyre[18], men det stilles noen ekstra krav til den. For det første kan hendelsen skape noen verdier, som "følger med" hendelsen og må tas vare på som parametre. For det andre må vi angi hvilke hendelser prosedyren skal håndtere:

```
Private Sub Åpne( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs _
        ) Handles butÅpne.Click, mnyÅpne.Click, knappÅpne.Click
```

Denne prosedyren *Åpne* håndterer tre, forskjellige hendelser. Alle de tre hendelser sender de samme to aktuelle argumentene, nemlig et *System.Object* og et *System.EventArgs*. Disse argumentene må prosedyren ta imot med formelle parametre.

Hvis to hendelser sender forskjellige aktuelle argumenter, kan vi ikke enkelt håndtere dem med samme hendelsesprosedyre. Derfor gir dette feilmelding:

```
Private Sub Lese( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs _
        ) Handles butÅpne.Click, knappÅpne.MouseMove
```

Feilmelding:

Method 'Private Sub Lese(sender As Object, e As System.EventArgs)' cannot handle Event 'Public Event MouseMove(sender As Object, e As System.Windows.Forms.MouseEventArgs)' because they do not have the same signature.

Vi ender med å måtte lage to hendelsesprosedyrer med forskjellige formelle parametre, men istedenfor å skrive koden dobbelt, kan vi jo la dem kalle på en og samme prosedyre. (Siden de vil ha forskjellige formelle parametertyper, kan de likevel hete det samme – jfr. ovenfor om signaturer.)

[18] Det kan *ikke* være en funksjon, for den har ikke noe sted å returnere funksjonsverdien. Den kan jo ikke returnere en verdi til et klikk e.l.

Tema C2 – Snadder: Gjør Windowsprogrammet mer "profesjonelt"

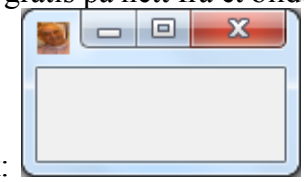
"Profesjonelle" Windowsprogrammer har elementer som vi ikke har sett mye på hittil. Her skal vi se på noen av dem.

Hjelp

Alle programmer med respekt for seg selv, tilbyr hjelp – gjerne med trykk på F1-tasten. Hvordan det gjøres, skal vi komme tilbake til i egne forelesninger senere. Det samme gjelder ToolTips som viser et lite vindu med litt tekst når brukeren holder musen stille over et ikon.

Ikon for applikasjonen

Vinduet og den kompilerte versjonen, har et ikon med default omtrent slik:  Dette kan du endre selv. Du endrer ikonet for hvert *skjema* ved å sette egenskapen *Icon*. Du endrer ikonet for den *kompilerte (.exe) versjonen* ved å velge *Icon* under *Project/Properties/Application*. Du må jo ha et ikon å velge, og det finner du mange av på Internett, eller du kan lage ikonet der. Slik ser f.eks. jeg ut som ikon laget gratis på nett fra et bilde: 

Skjemaet blir slik: 

Splash Screen

Mange programmer viser et eget bilde/vindu mens programmet starter. Det heter Splash Screen, og det lager du enkelt ved å gå inn i *Project/Add Windows Form*, der du bl.a. finner *Splash Screen*. Det henter opplysninger fra *Project Properties*, men det er enkelt å tilpasse – enten ved å skrive inn det du vil ha, eller endre egenskapene i *Project Properties*. Tilpass det så det blir som du vil ha det. Sett også egenskapen *Splash Screen* i *Project/Properties*, så er den saken biff!

Innloggingsskjema

VB tilbyr også et standardisert innloggingsskjema. Det henter du inne gjennom *Project/Add Windows Form* og velger *Login Form*. Du må selv legge til kode for å kontrollere brukernavn/passord og åpne det riktige skjemaet hvis alt er OK. Du må også sette innloggingsskjemaet som åpningsskjema i *Project Properties* og sjekk ut samme sted at applikasjonen *ikke* skal lukkes "When startup form closes" (for da vil applikasjonen avslutte når innloggingsskjemaet lukkes).

Legg merke til trikset Microsoft har brukt i innloggingsskjemaet: Når du trykker ALT+P skal markøren hoppe til labelen merket "Password", men den har ikke brukeren tilgang til (det er jo en etikett) og da hopper markøren til den neste kontrollen i tab-rekkefølgen, dvs. tekstfeltet for passord. Fikst!

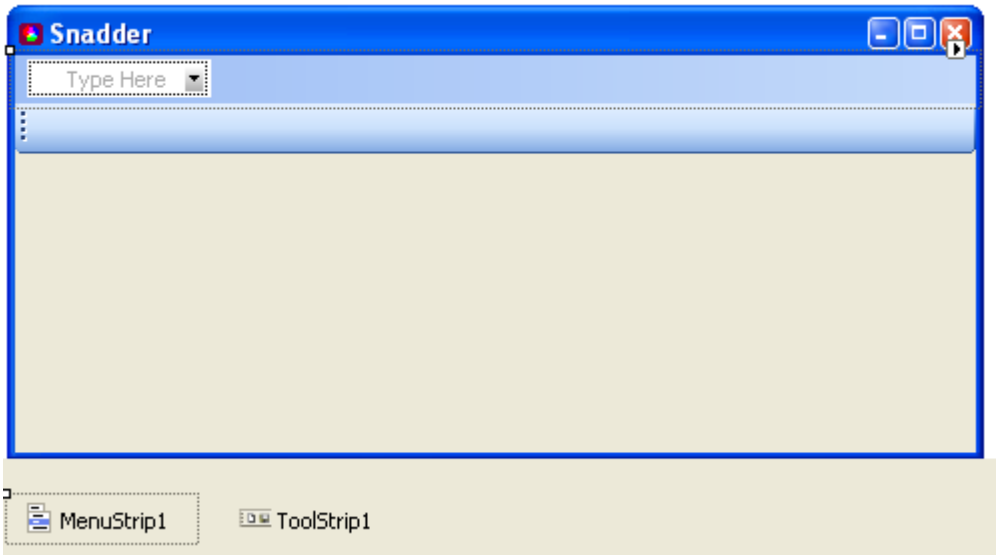
About Box ("Om-skjema")

Det er vanlig å kunne velge "Om.." på en meny og få vite litt om programmet. VB kan også lage det for deg. Bruk *Project/Add Windows Form* og velg *About Box*. Du må selv skrive koden som viser denne boksen, og der bør du ikke bruke *Show* men *ShowDialog*. Da regnes om-skjemaet som et underordnet skjema på samme måte som f.eks. *MsgBox*.

Du kan gi egenskaper til prosjektet, som vil komme med i *About Box*, gjennom *Project Properties* (*Assembly Information*).

Menyer og verktøylinje

Det er vanlig at vinduer har menyer og en verktøylinje øverst. Dem kan du lage selv. Du finner dem i *Toolbox* under *Menus & Toolbars*. Bruk gjerne *MenuStrip* og *ToolStrip*. Slik ser det ut i programmeringsmiljøet når begge er lagt til:



Her er det klikket på *MenuStrip1* (som du kan endre navn på) og da er det bare å begynne å skrive der det står "Type Here". Tilsvarende gjelder for verktøylinjen.

En av menyene bør være "Hjelp" og der finner man gjerne "Om…" som viser Om-boksen.

Hvis du vil at menyen "Hjelp" skal være tilgjengelig med snarveien ALT+H, skriver du "&Hjelp". Da vil "H" bli understreket[19] og brukeren kan bruke ALT+H fra tastaturet istedenfor å klikke menyen. (Du kan gjøre det samme på kommandoknapper, og i etiketter[20].)

Hvert menyvalg har egenskaper, som alle andre kontroller. Der kan du endre det litt håpløse navnet som VB foreslår. Bruk gjerne prefikset "mny" eller "mnu" f.eks. "mnyOm".

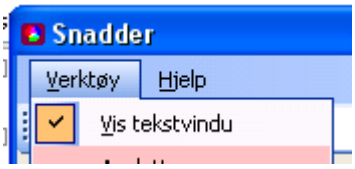
Du må programmere hendelsene som inntreffer når brukeren velger en meny. Dobbelt-klikk på menyen for å få frem koden.

Ikonene på verktøylinjen får du frem på samme måte, men her er det mer å velge mellom: Knapp, komboboks osv. Egenskapen *Image* angir hvilket bilde som evt. skal vises, og du må selvsagt programmere hver og en for å få noe gjort.

Det er vanlig å si at alt som det finnes verktøy for på verktøylinjen, skal det også finnes meny for, men ikke omvendt. Verktøylinjen oppfattes altså ofte som et *utdrag av menyene*. Om det skal være menyvalg for det som det finnes vanlige knapper for, er en smaksak.

[19] Noen brukere ser ikke understrekingen i menyer (men de synes på kommandoknapper). Da må de endre oppsettet i Windows. Du finner det under *Display* i *Control Panel* – fanen *Appearance* og knappen *Effects*. Der er det en sjekkboks for "*Hide underlined letters until I press the Alt keyK*". Den må ikke være krysset. Uansett kommer understrekingen frem når brukeren trykker ned ALT-tasten.

[20] Siden brukeren ikke kan aktivere en etikett, vil bruk av snarveien føre til at *neste* kontroll i tabulatorrekkefølgen fokuseres.



Hvis du vil at menyen skal sjekkes, setter du egenskapen *Checked* til *True*. Når brukeren velger en slik meny, skal den sjekkes eller avsjekkes og noe skal sikkert også gjøres. Denne koden endrer sjekkingen og endrer synligheten av et tekstvindu i takt:

```
mnyVisTekst.Checked = Not mnyVisTekst.Checked
rtfTekst.Visible = mnyVisTekst.Checked
```

Husk at hvis brukeren har flere måter å gjøre det samme på, kan du lage bare én prosedyre som håndterer alle hendelsene. Denne håndterer f.eks. tre hendelser (fet skrift):

```
Private Sub Avslutt(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles butAvslutt.Click, mnyAvslutt.Click, knappAvslutt.Click
    End
End Sub
```

Lagre fil dialog

Når brukeren ber om å få noe lagret, enten ved å velge på en meny, klikke på verktøylinjen eller på en knapp, bør du vise standard Windows *SaveFileDialog*. Du henter den fra *Tools* og bruker den slik (jeg har gitt den navnet *dlgLagreFil*):

```
'Setter noen egenskaper (det finnes flere):
dlgLagreFil.DefaultExt = "rtf"
dlgLagreFil.Filter = _
    "Tekstfiler|*.rtf;*.txt;|Alle filer (*.*)|*.*"
dlgLagreFil.InitialDirectory = "D:\UserData"
dlgLagreFil.Title = "Lagre teksten i Snadder Demo"
dlgLagreFil.CreatePrompt = True 'default
dlgLagreFil.OverwritePrompt = True
'Viser dialogen
'Sjekker om og evt. hva brukeren valgte, og evt. lagre teksten:
If dlgLagreFil.ShowDialog() = vbCancel Then 'klikket Cancel
    MsgBox("Ingenting lagret", MsgBoxStyle.Information, "Lagring")
Else 'har valgt fil - lagre teksten
    rtfTekst.SaveFile(dlgLagreFil.FileName)
End If
```

Dette skjer:

1) Default etternavn på filen vil bli ”rtf” (men brukeren kan velge noe annet)
2) Brukeren kan velge å vise ”Tekstfiler” og får vist rtf- og txt-filer, eller ”Alle filer” (syntaksen er en iterasjon av (1) hva som skal stå i vinduet, deretter (2) hvilken filtereffekt det skal ha atskilt med |)
3) Dialogen viser først katalogen D:\UserData – hvis den finnes (eller brukes applikasjonens katalog)
4) Tittelen i dialogvinduet blir satt til ”Lagre teksten i Snadder Demo”
5) Hvis brukeren skriver et filnavn som ikke finnes, vil han få spørsmål om filen skal lages
6) Hvis brukeren velger en fil som finnes, vil han bli spurt om den skal overskrives
7) Dialogen vises som dialogvindu, underordnet hovedskjemaet som *Child Form*
8) Hvis brukeren klikker Cancel vises en melding om det. Dialogboksen holder ellers selv orden på om det er oppgitt noe filnavn, om den kan overskrives osv.
9) Hvis brukeren velger/skriver inn et filnavn, vil egenskapen *Filename* inneholde filnavnet med full sti – her brukes filnavnet til å lagre *rtfTekst*. OBS! Dette kan gå galt – bruk feilfelle!

Alle egenskapene kan også settes i egenskapsvinduet i design modus.

Åpne fil dialog

OpenFileDialog likner svært på *SaveFileDialog*, men noen egenskaper er litt annerledes:

```
'Setter egenskaper for dialogvinduet
dlgÅpneFil.ShowReadOnly = True
dlgÅpneFil.DefaultExt = "rtf"
dlgÅpneFil.Filter = _
    "Tekstfiler|*.rtf;*.txt;|Alle filer (*.*)|*.*"
dlgÅpneFil.InitialDirectory = "D:\UserData"
dlgÅpneFil.Title = "Lese tekst inn i Snadder Demo"
dlgÅpneFil.FileName = ""
'Viser dialogen
If dlgÅpneFil.ShowDialog() = vbCancel Then 'trykket Cancel
'Sjekker om og evt. hva brukeren valgte, og evt. lagre teksten:
If dlgÅpneFil.FileName = "" Then 'valgte ingen fil/klikket Cancel
    MsgBox("Ingenting lest", MsgBoxStyle.Information, "Lesing")
Else 'har valgt fil - lagre teksten
    rtfTekst.LoadFile(dlgÅpneFil.FileName)
End If
```

Dette skjer:

1) Dialogen vil vise en sjekkboks for å åpne kun for lesing
2) Default etternavn på filen vil bli ”rtf”
3) Brukeren kan velge å vise ”Tekstfiler” og får vist rtf- og txt-filer, eller ”Alle filer”
4) Dialogen viser først katalogen D:\UserData – hvis den finnes (hvis ikke vises applikasjonens katalog)
5) Tittelen i dialogvinduet blir satt til ”Lese tekst inn i Snadder Demo”
6) Filnavnet som blir foreslått settes til tom streng. (Dette er viktig for å unngå at *Cancel* likevel gir lesing, for da vil filnavnet *ikke* være en tom streng – *Cancel* tømmer dessverre ikke strengen. Hvis ikke du selv har satt filnavnet, vil likevel dialogboksen foreslå noe, nemlig navnet på dialogen.)
7) Dialogen vises som dialogvindu, underordnet hovedskjemaet
8) Hvis brukeren klikker *Cancel* gis en melding om det – dialogboksen holder ellers selv orden på om filnavn er oppgitt, filen finnes osv.
9) Hvis brukeren velger/skriver inn et filnavn, vil egenskapen *Filename* inneholde filnavnet med full sti – da leses filen fra det oppgitte filnavnet. OBS! Dette kan gå galt – bruk feilfelle!

Alle egenskapene kan også settes i egenskapsvinduet i designmodus.

Fargedialog

Hvis brukeren skal velge en farge, bør du vise standard *ColorDialog*. Det kan gjøres ved å legge inn en *ColorDialog* fra *Tools*, som vi har gjort ovenfor. Da kan også egenskapene settes i designvinduet. Her viser jeg isteden hvordan det kan gjøres uten å bruke en kontroll. Da skaper jeg isteden et objekt, og setter objektets egenskaper i programmet:

```
'Skap et ColorDialog-objekt
Dim dlgVelgFarge As New ColorDialog()
'Sett defaultfarge lik den som er i bruk
dlgVelgFarge.Color = rtfTekst.BackColor
'Åpne dialogen og sjekk resultatet
If (dlgVelgFarge.ShowDialog() = Windows.Forms.DialogResult.OK) Then
    'Har valgt, endre bakgrunnsfargen i rtfTekst:
    rtfTekst.BackColor = dlgVelgFarge.Color
End If
```

Her kan vi ikke se om brukeren har svart noe ved å sjekke en streng slik vi gjør med *OpenFileDialog*, *InputBox* osv. Isteden returnerer *ColorDialog* et *DialogResult*-objekt som angir hvilken knapp brukeren klikket og som vi kan sjekke på.

Vindusstil

Ikke alle vinduer i Windows er like. Du kan endre utseendet med form-egenskapen *FormBorderStyle*:

```
Me.FormBorderStyle = Windows.Forms.FormBorderStyle.Sizable 'default
```

Merk at mange av de andre er beregnet for spesielle formål, og derfor kan *ControlBox* forsvinne fra dem, og noen vil også forsvinne fra oppgavelinjen og ALT+TAB. *Fixed* innebærer at brukeren ikke kan endre størrelsen ved å dra i kantene. Andre vinduer enn standard, egner seg vanligvis best for dialogvinduer som hører inn under et annet skjema, som vises med *ShowDialog* (istedenfor *Show*).

Andre eksempler:

```
'TabOrder satt bevisst i design men kan også endres dynamisk
'Noen egenskaper for buttons
butAvslutt.TabStop = False
butAvslutt.Cursor = Cursors.Hand
butLagre.TabIndex = 0
'Noen egenskaper for menyer
mnyAvslutt.Text = "&Avslutt"
mnyAvslutt.ShortcutKeyDisplayString = "ALT+A"
mnyVisTekst.ShortcutKeys = Keys.F12
mnyVisTekst.ShowShortcutKeys = True 'default
'Egenskaper for knapper
cboFarge.SelectedIndex = 0 'viser første tekst = "Velg..."
'Noen egenskaper for selve skjemaet
'Me.AcceptButton = butLagre 'Enter tilsvarer klikk på butLagre
Me.CancelButton = butAvslutt 'når bruker trykker ESC
Me.HelpButton = True 'vises ikke når min/maks knappene er på
Me.MaximizeBox = False   'default
Me.MinimizeBox = True   'default
Me.ControlBox = True 'default
Me.WindowState = FormWindowState.Normal 'default
Me.FormBorderStyle = Windows.Forms.FormBorderStyle.Sizable 'default
Me.ShowInTaskbar = True
```

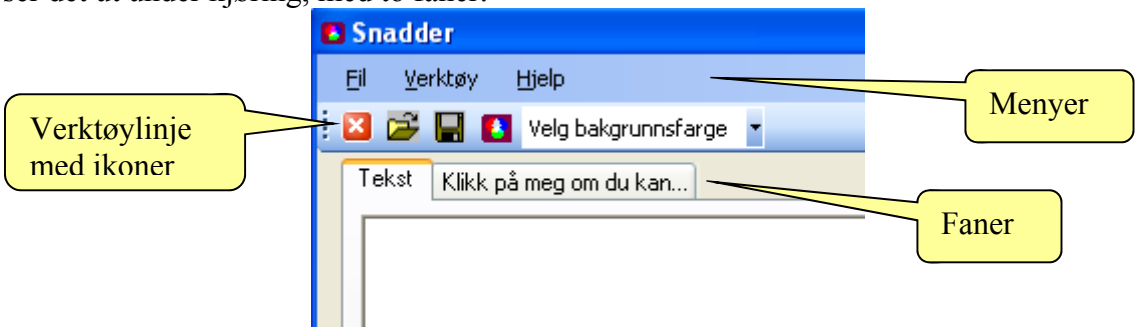
Her gjøres følgende:

1) En kommandoknapp tas ut av tabulatorrekkefølgen
2) En kommandoknapp får en annerledes kursor. Når musen peker på knappen, endres utseende på kursoren.
3) En annen kommandoknapp settes først i tabulatorrekkefølgen.
4) En meny får ny tekst – med snarvei ALT+A
5) En annen meny får satt snarvei til F12-tasten
6) Menyen skal vise brukeren hva om snarveien for denne menyen
7) En komboboks på verktøylinjen skal vise den første av de lovlige tekstene
8) En knapp får snarveien Enter. (Her er den ”kommentert vekk” fordi det gir problemer når brukeren trykker Enter-tasten inne i teksten han skriver inn.)
9) Gir en kommandoknapp snarveien Esc.
10) Viser et spørsmålstegn øverst til høyre, men den kommer bare frem når min- og maks-ikonene er fjernet. (Det vil passe for dialogbokser der brukeren skal fylle ut data. Vi ser mer på dette når vi lager interaktiv hjelp senere i kurset.)
11) Maksimeringsikonet øverst til høyre skjules
12) Minimeringsikonet øverst til høyre vises
13) Kontrollboksen – dvs. de ikonene som står øverst til høyre i vinduet, vises. Når de ikke vises, kan brukeren hverken maksimere, minimere eller lukke vinduet med disse ikonene.
14) Angir hvordan vinduet skal vises når det åpnes (normal størrelse, minimert eller maksimert)
15) Angir hva slags kant vinduet skal ha. (Velger du en *Fixed* kant, kan brukeren ikke endre størrelse på vinduet ved å dra i hjørner/kanter.)
16) Vinduet skal vises på oppgavelinjen.

Alle disse egenskapene kan også endres under design.

Fanesamlinger (tab collections)

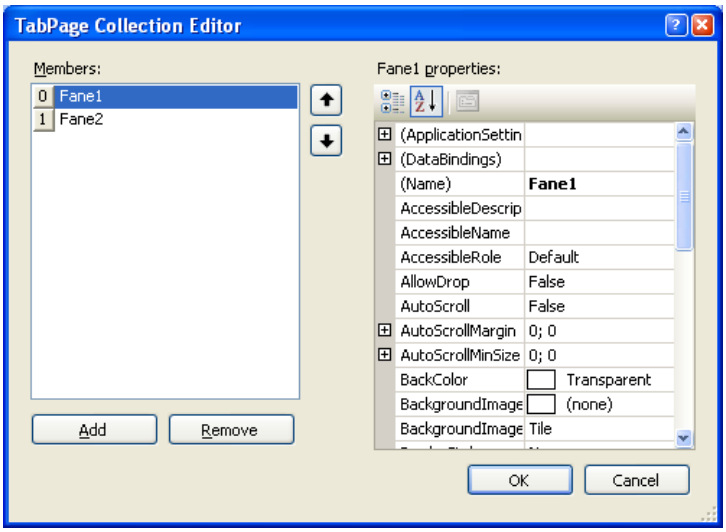
Fra *Tools* kan du lage faner. Faner er en *samling* (*collection*) med fanesider (*Tab Pages*). Du lager først fanesamlingen, deretter kan du legge kontroller inn i dem ved å dra dem inn på riktig faneside. Slik ser det ut under kjøring, med to faner:



For å få satt egenskaper på hele fanesamlingen, klikker du på en av fanene. Egenskapsvinduet viser da egenskapene for *hele samlingen*.

For å sette egenskaper for *hver enkelt faneside*, må du velge egenskapen *TabPages* og klikke på det lille ikonet til høyre for teksten *(Collection)*.

Da fremkommer dette vinduet, som bør være rimelig selvforklarende, men som nok trenger litt eksperimentering før alt virker helt som du vil☺:



(Jeg har klikket sorteringsknappen for å få egenskapene alfabetisk, slik jeg er vant til. Du kan se at jeg har gitt fanesiden et nytt navn.)

Du henviser til fanesider og kontroller på fanesiden direkte med navn, uten å bry deg om at kontrollen ligger på en faneside som inngår i en fanesamling. Selvom f.eks. *rtfTekst* ligger inne på *Fane1* i *Fanesamling*, kan du altså allikevel helt enkelt skrive

```
rtfTekst.Clear()
```

MsgBoxStyle i meldingsbokser

Ikoner

```
MsgBoxStyle.Critical
MsgBoxStyle.Exclamation
MsgBoxStyle.Information
MsgBoxStyle.Question
```

Knapper

```
MsgBoxStyle.AbortRetryIgnore
MsgBoxStyle.OkCancel
MsgBoxStyle.OkOnly 'Default
MsgBoxStyle.RetryCancel
MsgBoxStyle.YesNo
MsgBoxStyle.YesNoCancel
```

Annet

```
MsgBoxStyle.ApplicationModal 'Applikasjonen venter til meldingen er besvart
MsgBoxStyle.DefaultButton1 'Første knapp fra venstre er default
MsgBoxStyle.DefaultButton2 'Andre knapp fra venstre er default
MsgBoxStyle.DefaultButton3 'Tredje knapp fra venstre er default
MsgBoxStyle.MsgBoxSetForeground 'Meldingen blir aktivt vindu uansett
MsgBoxStyle.SystemModal 'Ingen applikasjoner fortsetter før meldingen er besvart
```

Default

Default MsgBoxStyle er

```
MsgBoxStyle.OkOnly Or MsgBoxStyle.DefaultButton1 Or MsgBoxStyle.ApplicationModal
```

Feil

Det blir feil å bruke to konstanter som overlapper hverandre. F.eks. kan man ikke samtidig be om både *Information* og *Critical*-ikon.

Hva klikket brukeren?

Når vi viser en meldingsboks, kan brukeren klikke på en eller flere knapper (satt opp med en knappekombinasjon som forklart ovenfor). Vi kan være interessert i å finne ut hva brukeren faktisk klikket på. Det returnerer *MsgBox* som en funksjonsverdi så du kan sjekke resultatet:

```
Dim svar As MsgBoxResult 'kan også deklareres som en Integer
svar = MsgBox("Vil du virkelig avslutte?", MsgBoxStyle.YesNo, "Avslutter")
If svar = vbYes Then End
```

Det er en konstant for hver knapp, som du kan sammenlikne med:

```
vbOK
vbCancel
vbAbort
vbRetry
vbIgnore
vbYes
vbNo
```

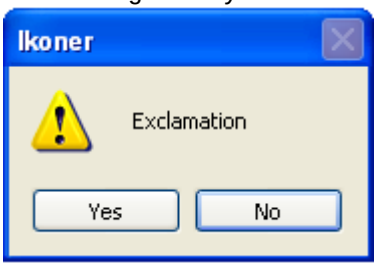
Konstanter

Istedenfor *MsgBoxStyle.OkOnly* osv., kan man bruke konstanter, f.eks. *vbOkOnly*, men det går jeg ikke inn på her.

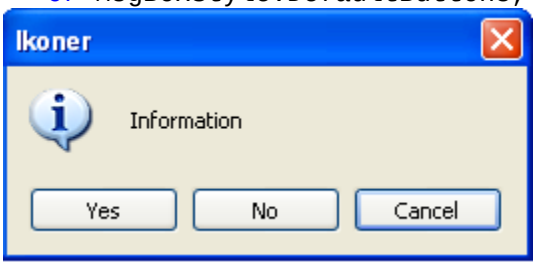
Eksempler

```
resultat = MsgBox("Critical", _
   MsgBoxStyle.Critical Or MsgBoxStyle.OkOnly, "Ikoner")
```

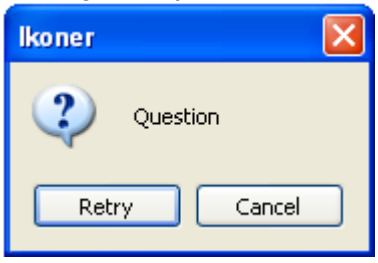

```
resultat = MsgBox("Exclamation", _
   MsgBoxStyle.Exclamation Or MsgBoxStyle.YesNo _
   Or MsgBoxStyle.DefaultButton2, "Ikoner")
```




```
resultat = MsgBox("Information", _
   MsgBoxStyle.Information Or MsgBoxStyle.YesNoCancel _
   Or MsgBoxStyle.DefaultButton3, "Ikoner")
```




```
resultat = MsgBox("Question", _
   MsgBoxStyle.Question Or MsgBoxStyle.RetryCancel, "Ikoner")
```



Caveat (= pass på!)

Det demo-programmet jeg har laget for dette, har *ikke* godt grensesnitt. Det er for overlesset. Jeg har selvsagt gjort det for å få vist mest mulig. Dere skal senere ha grafisk brukergrensesnitt (GUI) som eget fag.

Tema D – Rekursive funksjoner og rekursiv programmering
Rekursjon på godt og vondt – en utfordring!

Generelt

Hva må et programspråk minimum ha
En stor matematiker, Alan Turing, er kjent bl.a. for å legge grunnlaget for elektroniske datamaskiner. Han definerte i 1937 (før man faktisk hadde slike maskiner) en teoretisk datamaskin "a-maskin" som nå kalles "Turings datamaskin". Turing brukte denne teoretiske maskinen som grunnlag for å bevise teoremer om datamaskiner generelt.[21]

Et av hans resultater gjaldt hva som må finnes i et programmeringsspråk for at det skal være "komplett". Forenklet kan vi si at programmet *på visse betingelser* må kunne *hoppe til et hvilket som helst sted i minnet* og utføre det som står der, og det må kunne *endre verdien av en hvilken som helst celle*.

Programmeringsspråket må altså gjøre det mulig å programmere
1. *endring av verdier*
2. *beslutninger*
3. *hoppe til et annet sted i programmet*

Her står det ingen ting om iterasjoner (gjentakelser), men det kan enkelt programmeres med de tre enkle mulighetene ovenfor. Her viser jeg f.eks. tre tall i en listeboks med bare de enkle elementene:

```
    Private Sub butIterasjonGoto_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles butIterasjonGoto.Click
        'Viser iterasjon uten for-next men goto og seleksjon
        Dim i As Integer = 1
gjenta:  'label for goto
        lstOutput.Items.Add(i)
        i += 1 'endrer verdien av celle i
        If i < 4 Then GoTo gjenta 'valg og evt. hopp
    End Sub
```

I praksis blir dette litt tungvint. De fleste språk inneholder derfor *enten* iterasjoner (For/Next, Do/Loop, Do While osv.) *og/eller* rekursjon (der man tillater at en prosedyre eller funksjon kaller seg selv). Da unngår man også å bruke *goto* med etikett, og det er en fordel fordi det blir vanskeligere å rote til programmet ("spagettiprogrammer"). Slik programmering kalles ofte "goto-less programming". Noen språk har kun iterasjoner, andre kun rekursjon men svært mange har begge deler, slik som Visual Basic.[22]

[21] Forenklet kan maskinen hans beskrives som en enhet med en uendelig lang tape som kan flyttes frem og tilbake i en enhet. Tapen har plasser for verdier og er maskinens minne. Ett, eneste symbol på tapen er til enhver tid inne i maskinen, og alle plassene på tapen kan representere enten en verdi eller en handling. Enheten kan hoppe til et hvilket som helst sted på tapen, og kan endre symbolet som ligger der. Det vil altså enten endre en verdi eller maskinens senere handlinger. I dagens maskiner snakker vi isteden om registre og celler i RAM, og enheten som kan lese/endre symbolene er CPU. CPU henter innholdet i en og en celle fra RAM og kan utføre det som ligger der eller endre verdien og legge den tilbake. Dette er altså helt analogt med Turings maskin, bare at våre datamaskiner ikke har uendelig minne. Teoriene til Turing gjelder imidlertid også for våre endelige maskiner.

[22] I tillegg har programmeringsspråkene masse annet, som skal gjøre det enklere for programmereren, men som altså teoretisk sett ikke er strengt nødvendig.

Slik ville vi kanskje gjort det i VB som bl.a. har iterasjonen For/Next:

```
Private Sub butIterasjonFor_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butIterasjonFor.Click
    'Viser utskrift av tre tall med iterasjon for-next
    For i As Integer = 1 To 3 Step 1
        lstOutput.Items.Add(i)
    Next
End Sub
```

Da slipper vi

1. å bruke *goto* med en etikett
2. å øke variabelen *i* da den ligger implisitt i verbet *Next* og verdien av *Step*
3. å skrive testen som skal gi uthopp da den ligger implisitt i *To*

De fleste vil nok oppleve dette som enklere.

Rekursjon

Jeg minner (igjen) om at en funksjon er en regel som til hvert element i en mengde A tilordner én (og bare en) verdi i en mengde B. Mengden A kalles *definisjonsmengden*, og mengden B er *løsningsmengden*. Regelen kan være delt opp i flere delregler som til sammen definerer funksjonen. **Når A og B er samme mengde, har vi rekursjon.** Vi har brukt rekursjon i rekkeutviklinger, der vi hadde en *enkel* regel (f.eks. $X_1 = 15$) og en *rekursiv* regel (f.eks. $X_i = X_i + 3$). Man må ha minst én av begge, ellers er det ikke rekursjon (den rekursive regelen mangler) eller evig rekursjon (den enkle regelen mangler).

Et eksempel på evig rekursjon – ment som en fleip – er betydningen av GNU: "GNU's Not Unix" der GNU er brukt rekursivt i definisjonen av GNU[23].

Her gjør jeg det samme som ovenfor rekursivt:

```
Private Sub butRekursjon_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butRekursjon.Click
    'Viser utskrift av tre tall med rekursjon
    skrivTall(1) 'starter rekursjonen
End Sub

Private Sub skrivTall(ByVal n As Integer)
    If n >= 4 Then Return 'enkel regel
    lstOutput.Items.Add(n)
    skrivTall(n + 1) 'rekursjon
End Sub
```

Hoppet *goto* er her erstattet av et prosedyrekall til seg selv (*skrivTall* kaller opp *skrivTall*). Dette innebærer også et hopp til et annet sted i programmet, men her huskes returadressen så *skrivTall* kan returnere en verdi til stedet den ble kalt fra. Den enkle regelen gir uthopp fra rekursjonen når n >= 4.

Rekursjon kan alltid erstattes av iterasjon, men det kan være svært vanskelig å finne riktig iterasjon og vanskelig å programmere, hvis problemet er klart rekursivt[24]. Disse problemene er betydelig enklere å skrive rekursivt. Rekursjon gir også ofte kort, "elegant" kode. En ulempe er at de bygger opp en *stack* med returadresser, og genererer nye variabelsett for hver rekursjon. Det krever stackplass og minneplass, og begge er begrenset. Rekursive programmer kan derfor plutselig gi kjørefeil fordi stack og/eller minne er fullt.

[23] GNU er navnet på en familie av lisenser for fri programvare som forvaltes av Free Software Foundation, fsf.org.
[24] Dessuten finnes det altså språk som ikke har iterasjon – bare rekursjon, men det er ikke mange igjen.

Rekursjon dreier seg om å "se problemet løst" og så etterpå jobbe baklengs fra løsningen. Her er et eksempel på en rekursiv arbeidsform (ikke et program) der vi gjør det enklere for oss selv:

> **Oppgave: Opprinnelig pris**
> **Frakt for et vareparti utgjør kr 10 000. Toll legges på med 20 % av pris med frakt, og kjøpmannen legger til slutt på 100 % avanse. Varepartiet ble solgt for kr 240 000. Hva var opprinnelig kjøpspris for varepartiet?**

Noen løser dette med likning:

$$\left[(k+10000)+(k+10000)\cdot 20\%\right]+\left[(k+10000)+(k+10000)\cdot 20\%\right]\cdot 100\% = 240000$$

$$\Leftrightarrow (k+10000)\cdot 120\%\cdot 200\% = 240000 \Leftrightarrow (k+10000)\cdot 240\% = 240000$$

$$\Leftrightarrow 2{,}4\cdot k + 24000 = 240000 \Leftrightarrow 2{,}4\cdot k = 240000 - 24000 \Leftrightarrow 2{,}4\cdot k = 216000 \Leftrightarrow k = 216000/2{,}4$$

$$\Leftrightarrow k = 90000$$

Alternativt kan vi sette opp et skjema og "tenke oss oppgaven løst". Fyll selv ut og se om du får 90 000! Løsning bakerst i notatet.

Utsalgspris		240 000
Avanse 100% av pris fortollet	- Halvparten av utsalgsprisen	
Pris fortollet		
Toll 20% av pris på bryggen	- Toll 20%/120% = 1/6-del	
Pris på bryggen	Pris fortollet - toll	
Frakt Kr 10 000	Trekker fra 10 000	
Innkjøpspris		

Krav til rekursjon

Som nevnt ovenfor, stiller vi generelt følgende krav til rekursjon:

1. Det er minst én elementær regel
2. Det er minst én rekursiv regel, der ett element er definert på grunnlag av et annet element i samme mengde.

Når vi skriver programkode for rekursjon, har vi alltid et "sluttkriterium" som tilsvarer den elementære regelen, og minst ett rekursivt kall. Programmets struktur blir svært lik rekursjonens regler. Vanskeligheten er som oftest å finne de reglene som gjelder, ikke å skrive koden.

Eksempler

Eksempel 1 - Fakultet

Funksjonen **fakultet** kan defineres slik:

1. $0! = 1$
2. $n! = (n-1)!\cdot n$

0!	1!	2!	3!	4!
$=1$	$=(1-1)!\cdot 1$	$=(2-1)!\cdot 2$	$=(3-1)!\cdot 3$	$=(4-1)!\cdot 4$
	$=0!\cdot 1$	$=1!\cdot 2$	$=2!\cdot 3$	$=3!\cdot 4$
	$=1\cdot 1$	$=1\cdot 2$	$=2\cdot 3$	$=6\cdot 4$
	$=1$	$=2$	$=6$	$=24$

Her er den første regelen elementær, dvs den er definert uten henvisning til andre elementer i mengden, men regel 2 er rekursiv. De to reglene tilsammen definerer funksjonen.

For å programmere fakultet etter reglene ovenfor, følger vi reglene meget nøye, slik:

```
Private Function Fakultet(ByVal X As Long) As Long
    txtOutput.Text &= "Beregner " & X & "!" & vbCrLf
    If X = 0 Then
        Return 1 'regel 1
    Else
        Return Fakultet(X - 1) * X 'regel 2
    End If
End Function
```

På siste side i dette kapittelet er det en tegning som illustrerer hva som foregår.

Eksempel 2 – Enkel rekke

Eksempel med en **enkel** rekke:

1. $X_0=5$
2. $X_n=X_{n-1}+3$ for n>0

X_0	X_1	X_2	X_3	X_4
=5	$=X_{1-1}+3$ $=X_0+3$ $=5+3$ $=8$	$=X_{2-1}+3$ $=X_1+3$ $=8+3$ $=11$	$=X_{3-1}+3$ $=X_2+3$ $=11+3$ $=14$	$=X_{4-1}+3$ $=X_3+3$ $=11+3$ $=14$

Igjen ser vi én elementær og én rekursiv regel. Koden kan bli slik:

```
Private Function Enkel(ByVal n As Long) As Long
    If n = 0 Then
        Return 5 'regel 1
    Else
        Return Enkel(n - 1) + 3 'regel 2
    End If
End Function
```

Eksempel 3 – Fibonacci

Et annet eksempel med to elementære regler og en rekursiv, som gir de såkalte **Fibonacci-tallene**:

1. $X_0=0$
2. $X_1=1$
3. $X_n= X_{n-2}+ X_{n-1}$ for n>1

X_0	X_1	X_2	X_3	X_4
=0	=1	$=X_{2-2}+X_{2-1}$ $=X_0+X_1$ $=0+1$ $=1$	$=X_{3-2}+X_{3-1}$ $=X_1+X_2$ $=1+1$ $=2$	$=X_{4-2}+X_{4-1}$ $=X_2+X_3$ $=1+2$ $=3$

På samme måte som ovenfor, blir det en if-setning for hver regel, slik:

```
Private Function Fibonacci(ByVal n As Long) As Long
    If n = 0 Then
        Return 0 'regel 1
    ElseIf n = 1 Then
        Return 1 'regel 2
    Else
        Return Fibonacci(n - 1) + Fibonacci(n - 2) 'regel 3
    End If
End Function
```

Det vil gi *meget* fin øvelse om du forsøker å *skrivebordsteste* denne f.eks. med tall nr 4, *før* du prøvekjører den. Se ikke bort fra at du kan bli litt overrasket!

Eksempel 4 – Kompleks rekke

Et eksempel på en rekke med flere elementære regler og **flere** rekursive:

1. $X_0=4$
2. $X_1=7$
3. $X_n=X_{n-2}+4$ når n er partall > 0
4. $X_n=X_{n-2}+7$ når n er oddetall > 1

X_0	X_1	X_2	X_3	X_4
=4	=7	Partall : $=X_{n-2}+4$ $=X_{2-2}+4$ $=X_0+4$ $=4+4$ $=8$	Oddetall: $=X_{n-2}+7$ $=X_{3-2}+7$ $=X_1+7$ $=7+7$ $=14$	Partall : $=X_{n-2}+4$ $=X_{4-2}+4$ $=X_2+4$ $=8+4$ $=12$

Dette eksempelet er mer komplisert rekker med to elementære og to rekursive regler. Men igjen er det bare å programmere rett frem etter reglene:

```
Private Function Kompleks(ByVal n As Long) As Long
    If n = 0 Then
        Return 4 'regel 1
    ElseIf n = 1 Then
        Return 7 'regel 2
    ElseIf n Mod 2 = 0 Then 'partall
        Return Kompleks(n - 2) + 4 'regel 3
    Else 'oddetall
        Return Kompleks(n - 2) + 7 'regel 4
    End If
End Function
```

Eksempel 5 – Søking i en sortert array med binærsøk (halveringsmetoden)

Gitt en array med sorterte heltall (under prøvekjøring fylles den med partall så vi vet hva som finnes der). Vi skal sjekke om et bestemt tall finnes i arrayen, og det kan vi rekursivt gjøre slik:

Funksjonen *F(array, søk, fra, til)*, der *array* er den arrayen det skal søkes i, *søk* er det tallet det søkes etter, *fraindeks* er den laveste og *tilindeks* den høyeste indeksen vi søker blant.

1. Hvis det bare er ett element i *array* (da er *fraindeks* = *tilindeks*):
    a. Funnet: Hvis *søk* = *array(fraindeks)*: *F = True*
    b. ellers: *F = False*
2. Hvis det bare er to elementer i *array* (da er *tilindeks – fraindeks* < 2):
    a. Funnet: Hvis *søk = array(fraindeks)* eller *søk = array(tilindeks)*: *F = True*
    b. ellers: *F = False*
3. Hvis det er mer enn to elementer i *array* (da er *tilindeks – fraindeks* ≥ 2):
   Beregn *midtindeks* = (*tilindeks* + *fraindeks*) \ 2
    a. Søk i øvre halvdel:
    Hvis *søk* > *array*(*midtindeks*): *F = F(array, søk, midtindeks, tilindeks)*
    b. ellers søk i nedre halvdel: *F = F(array, søk, fraindeks, midtindeks)*

Funksjonen har fire argumenter: *Tall()* er den arrayen vi leter i, *fra* og *til* angir søkeintervallets grenser, mens *søk* angir hvilket tall vi leter etter. Det er regel 3 som er rekursiv, de to første er enkle.

Funksjonen programmeres rekursivt, slik:

```
Private Function finnes(ByRef Tall() As Integer, _
        ByRef søk As Integer, ByVal fraindeks As Integer, _
        ByVal tilindeks As Integer) As Boolean
    Dim midtindeks As Integer
    txtOutput.Text &= "Leter fra indeks " & fraindeks & _
        " til indeks" & tilindeks & vbCrLf
    If tilindeks - fraindeks < 2 Then 'maks to elementer - regel 1 og 2
        Return Tall(fraindeks) = søk Or Tall(tilindeks) = søk
    Else 'mer enn to igjen - regel 3
        midtindeks = (fraindeks + tilindeks) \ 2
        If søk > Tall(midtindeks) Then 'let i øvre halvdel
            Return finnes(Tall, søk, midtindeks + 1, tilindeks) '3a
        Else 'let i nedre halvdel
            Return finnes(Tall, søk, fraindeks, midtindeks) '3b
        End If
    End If
End Function
```

Det tar litt tid å bygge opp en stor array med heltall, men ellers er funksjonen forbausende rask. Det har jo med å gjøre at vi hele tiden halverer søkeområdet. Hvis vi f.eks. starter med 10 millioner elementer, reduseres søkeområdet til 2 elementer etter bare 24 rekursive kall, mens f.eks. 1 000 milliarder elementer, kan søkes med bare 40 rekursive kall.En full søk fra start til slutt i en løkke ville ta 1000 milliarder iterasjoner. Da ville det være bedre å prøve f.eks. hver milliard til vi kom for langt, deretter baklengs 100 millioner av gangen, oppover igjen 10 millioner av gangen osv.

PS: Det kan jo være greit å vite at Microsoft allerede har implementert dette for en én-dimensjonal og sortert null-basert[25] array. Du kan søke etter verdien *søk* i en arrary *Tall()* ved å skrive bare

```
Array.BinarySearch(Tall, søk)
```

Hvis verdien *søk* finnes, returnerer funksjonen indeksen der verdien søk ble funnet (finnes det flere får du én av dem – ikke nødvendigvis den første). Finnes den ikke, returneres et negativt tall. Denne er også marginalt raskere enn den jeg har vist ovenfor – jeg har målt det med et *StopWatch*-objekt.

Eksempel 6 – Snu en streng bak/fram

En funksjon som snur en streng bak/fram kan beskrives slik med ord: ”En snudd streng er lik siste tegn i strengen & resten av strengen snudd”. Mer formelt kan vi si at gitt en tegnarray (en streng) s med indekser fra 0..n, skrives s[0..n] der lengden av strengen er antall tegn (kan være 0), så kan vi snu den bak/fram med følgende funksjon:

1. Lengde = 0: snustreng("") = ""
2. Lengde = 1: snustreng(s[0]) = s[0]
3. Lengde > 1: snustreng(s[0..n]) = s[n] & snustreng[0..n-1]

[25] Null-basert (zero-based) betyr at laveste indeks er 0.

```
Private Function snustreng(ByVal s As String) As String
    'Strenger er indeksert fra 0 til lengden -1
    Dim lengde As Integer = s.Length 'strengens lengde
    If lengde = 0 Then 'regel 1, enkel
        Return ""
    ElseIf lengde = 1 Then 'regel 2, enkel
        Return s(0)
    Else 'regel 3, rekursjon
        Dim sistetegn As String = s(lengde - 1) 'det siste tegnet
        Dim forpart As String = s.Substring(0, lengde - 1) 'resten
        Return sistetegn & snustreng(forpart)
    End If
End Function
```

PS: Hvis et ord/setning blir lik enten den skrives bakfra eller forfra, kalles den et Palindrom (av gr. *palin* = «tilbake» eller «igjen» og *dromos* = «bane» eller «vei»). Da ser man bort fra stor/liten bokstav og "glemmer" mellomrommene. Her er noen eksempler (kilde Wikipedia):

- ✓ *Otto*
- ✓ *reker*
- ✓ *regninger*
- ✓ *Agnes i senga*
- ✓ *Den ene tar apparatene ned.*
- ✓ *Du har bra hud.*
- ✓ *Rolf Are vurderer om Arons ni drag i gardinsnora morer edru Vera Flor.*

Ved å kalle funksjonen snustreng, kan vi lett sjekke om setning = snrustreng(setning), men da må vi først fjerne alle mellomrom og gjøre alt til store eller alt til små bokstaver, og det er ikke *så* enkelt.

Eksempel 7 - Rekursive prosedyre

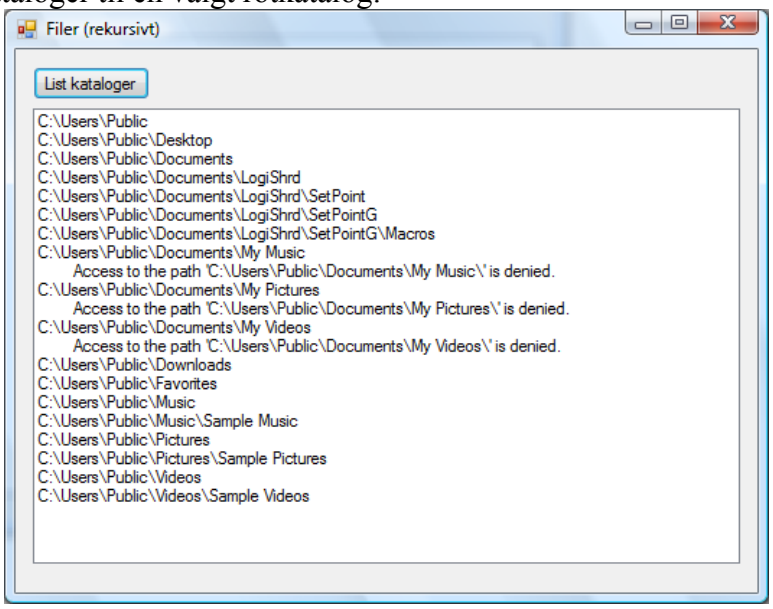
Man kan også bruke rekursive *prosedyrer*. Her er et enkelt, men oppkonstruert eksempel. Vi vil skrive ut en tekst *baklengs* på skjemaet, en bokstav pr linje. (Da kan vi ikke bruke regler for en rekursiv *funksjon*.) Vi kan skrive prosedyren slik:

```
Private Sub Baklengs(ByVal Tekst As String)
    If Tekst <> "" Then
        Baklengs(Tekst.Substring(1)) 'alt unntatt første tegn
        txtOutput.Text &= Tekst(0) 'skriv første tegn
    End If
End Sub
```

Denne kan gi litt å tenke på, for hvis vi bytter om på de to setningene inne i if-setningen, vil teksten bli skrevet ut *forlengs*!

Eksempel 8 – Liste kataloger

Her listes alle subkataloger til en valgt rotkatalog:




```
Sub finnDir(ByVal dirnavn As String)
    'Finner alle subkataloger dirnavn" og nedover, rekursivt
    listOut.Items.Add(dirnavn) 'legg denne til i listeboksen
    listOut.Refresh() 'så vi kan se hva som skjer - tar tid!
    Try 'GetDirectories kan gi feil hvis man ikke har tilgang
        'hent alle subkataloger til "dirnavn" og gå igjennom dem
        Dim kataloger As String() = System.IO.Directory.GetDirectories(dirnavn)
        For Each dir As String In kataloger
            finnDir(dir) 'rekursivt kall
        Next
    Catch ex As Exception
        listOut.Items.Add("        " & ex.Message) 'feilmelding til lstOut
    End Try
End Sub
```

finnDir er rekursiv og startes med kallet *finnDir(katalognavn)*. Den får da oppgitt et katalognavn som da regnes for "lokal rot". Den legges til i listeboksen. Deretter hentes alle subkataloger til denne "lokale rotkatalogen" med *Directory.GetDirectories*. De gås igjennom én for én og for hver av dem kalles *finnDir* igjen, denne gangen med en av disse katalogene som ny "lokal rotkatalog" osv.

Eksempel 9 – Rekursiv klasse

Det er ikke noe i veien for å ha klasser som inneholder objekter av samme klasse. F.eks. kan en venn ha en liste med venner:

```
Public Class Venn
    Public navn As String 'denne vennens navn
    Public venner As New List(Of Venn) 'liste med vennens venner (rekursjon!)
    Public Sub New(ByVal navn As String)
        'Konstruktør
        Me.navn = navn
    End Sub
    Public Sub addVenn(ByVal nyVenn As Venn)
        'Legg til en ny venn
        venner.Add(nyVenn)
    End Sub
    Public Overrides Function ToString() As String
        'Returnerer vennens navn og alle venner (med venners venner)
        Dim tekst As String
        tekst = Me.navn & vbNewLine
        For Each minVenn As Venn In venner
            tekst &= "Venn av " & Me.navn & ": " & minVenn.ToString()
        Next
        Return tekst
    End Function
End Class
```

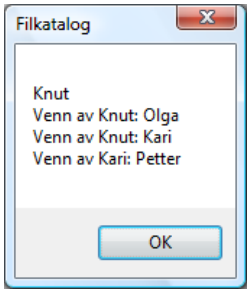
Denne klassen er rekursiv på to måter

1. Hver venn har en liste med sine venner som igjen har en liste med sine venner osv.
2. ToString kaller vennenes ToString som igjen kaller sine venners ToString osv.

Slik kan klassen testes:

```
    Private Sub butVenn_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles butVenn.Click
        'Prøver klassen Venn
        '1. Skaper noen venner
        Dim minVenn As New Venn("Knut") 'en av mine venner
        Dim venn1 As New Venn("Olga")
        Dim venn2 As New Venn("Kari")
        Dim venn3 As New Venn("Petter")
        '2. Legg inn venner
        minVenn.addVenn(venn1) 'Olga er venn av meg
        minVenn.addVenn(venn2) 'Kari er venn av meg
        venn2.addVenn(venn3) 'Kåre er venn av Kari
        '3. List alle i en meldingsboks
        MsgBox(minVenn.ToString)
    End Sub
```

Slik ser meldingsboksen ut:

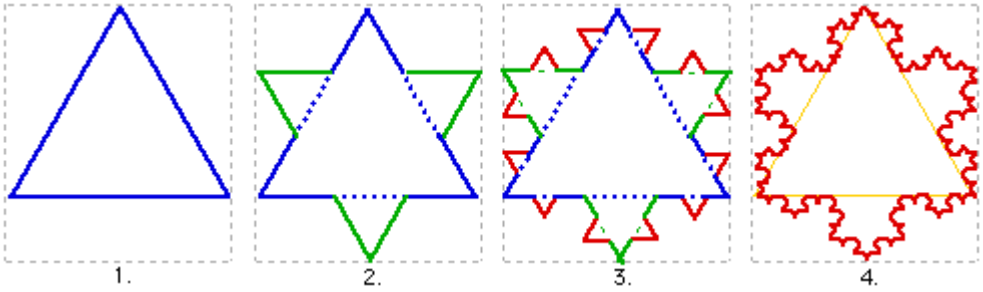


Legg merke til at vennen Knut har skrevet ut seg selv og alle sine venner, og en av dem – nemlig Kari – har skrevet ut sine venner også.

Ekstra: Rekursiv grafikk

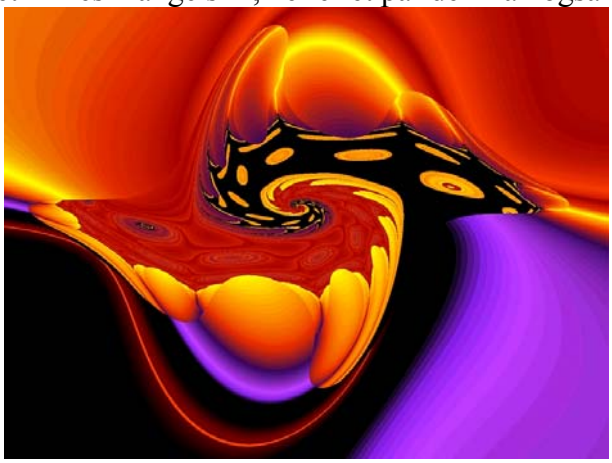
Rekursjon gjør det mulig å tegne grafikk som det ville ha vært svært vanskelig å tegne uten bruk av rekursjon. En type av slike er *fraktaler* som er tegninger som er tegnet rekursivt i det uendelige (de har altså ingen enkel regel). Omtale av fraktaler finner du f.eks. hos Wikipedia: http://en.wikipedia.org/wiki/Fractals.

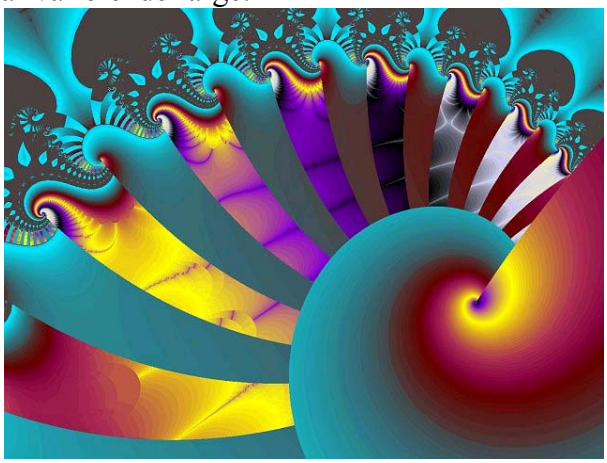
Et eksempel: Koch's Snowflake.
Figuren er en trekant, der hver linje har en trekant midt på, tegnet med linjer lik en tredjedel av linjen. Rekursjonen fremkommer ved at hver linje du tegner skal ha en trekant midt på som også består av linjer med en trekant midt på osv. Her er den tegnet iterativt:



(En flott animasjon på http://www.shodor.org/interactivate/activities/KochSnowflake/?version=1.5.0_04&browser=MSIE&vendor=Sun_Microsystems_Inc og mange andre steder).

Det finnes mange slik, her er et par der man også har varierende farge:





Løsning på oppgaven "opprinnelig pris":

Utsalgspris		240 000
Avanse 100% av pris fortollet	- Halvparten av utsalgsprisen	-120 000
Pris fortollet		=120 000
Toll 20% av pris på bryggen	- Toll 20%/120% = 1/6-del	-20 000
Pris på bryggen	Fortollet / 120%	=100 000
Frakt Kr 10 000	Trekker fra 10 000	-10 000
Innkjøpspris		=90 000

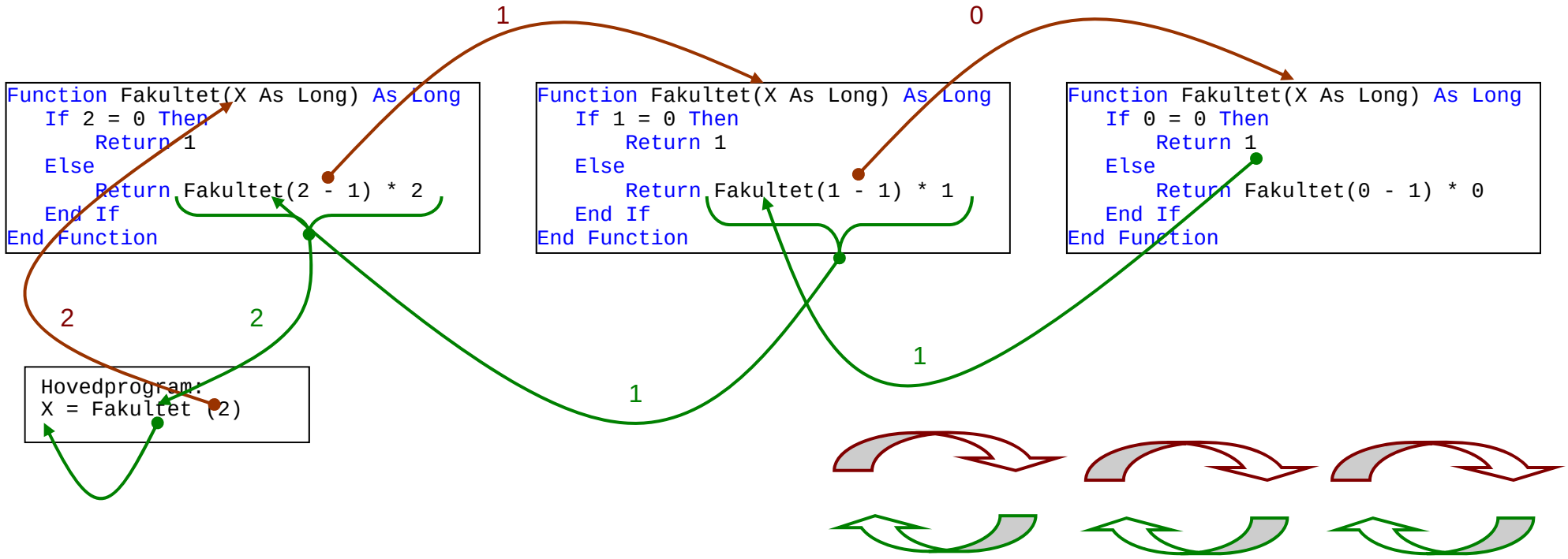
Regnet "riktig vei" – slik kjøpmannen sikkert gjorde det:

Innkjøpspris	90 000
Frakt kr 10 000	10 000
Pris på bryggen	100 000
Toll 20% av pris på bryggen	20 000
Pris fortollet	120 000
Avanse 100% av pris fortollet	120 000
Utsalgspris	240 000

Fakultet – figur

```
Private Function Fakultet(ByVal X As Long) As Long
    txtOutput.Text &= "Beregner " & X & "!" & vbCrLf
    If X = 0 Then
        Return 1 'regel 1
    Else
        Return Fakultet(X - 1) * X 'regel 2
    End If
End Function
```

Når Fakultet blir kalt opp, erstattes det formelle parameteret X med verdien av det aktuelle argumentet. Internt i funksjonen, brukes verdien av den X som ble overført. I tegningen er de rødbrune pilene (og tallene) overføring av det aktuelle argumentet. De grønne pilene (og tallene) representerer returverdien (= funksjonsverdien).



Tema E1 – Strengmanipulering (tegn og tekst, Char og String)

Det er også litt om hvordan man får hjelp i programmeringsmiljøet – se nedenfor.

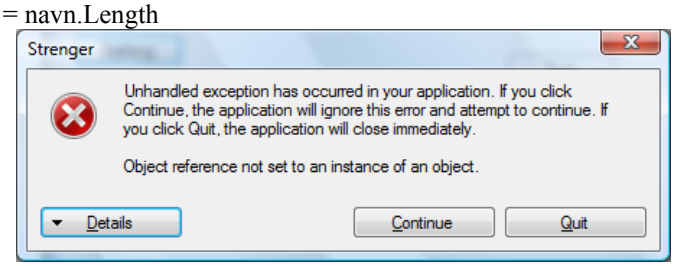
VB kaller tekster for String, og på norsk sier vi ofte "streng". En streng som har fått verdi, består av en sekvens med ingen (en "tom streng"), ett eller flere tegn.

Strenger er *objekter*. Inntil objekter har fått plass i RAM er de *Nothing*. Du kan sjekke om en streng er *Nothing* med uttrykket *Is Nothing* (heller enn = *Nothing*[26]):

```
If navn Is Nothing Then MsgBox("Strengen eksisterer ikke!")
```

Hvis man benytter operasjoner på en streng som er *Nothing*, vil det gi kjørefeil, f.eks.

```
Dim navn As String
Dim lengde As Integer = navn.Length
```



Tegn (Char)

Et tegn består av en bokstav eller et annet skrivbart tegn (det vil si at de synes på en printer) samt kontrolltegn som brukes til styring av printeren. VB bruker egentlig Unicode (to bytes), men faktisk lagres tekster med vanlige tegn etter ANSI med én byte pr tegn.
Eksempel på skrivbare tegn: a, b, A, Å, @, #, _
Eksempel på kontrolltegn: backspace, tab, linefeed

Du deklarerer tegn på vanlig måte:

```
Dim tegn As Char = "a"c 'tegnet "a"
Const a As Char = Chr(97) 'tegnet "a"
```

Tegnkonstanter angis med en streng med ett tegn i, etterfulgt av bokstaven *c* (for *character*), slik du ser det ovenfor. Du kan også bruke funksjonene *Chr()* som omgjør en tegnkode etter ANSI til et tegn og evt. *ChrW()* som omgjør en tegnkode etter Unicode til et tegn, deklarert slik av Microsoft:

```
Public Function Chr(ByVal CharCode As Integer) As Char
Public Function ChrW(ByVal CharCode As Integer) As Char
```

Forskjellen på Unicode og ANSI, er at Unicode inkluderer alle tegn på alle språk og er uavhengig av Windows setting. ANSI er avhengig av omgivelsesspråket og er *"Locale Aware"* (der vi i Norge har f.eks. æ har andre språk et annet tegn, så en tekst kan tolkes annerledes på en dansk maskin).

Man finner tegnkoden med *Asc()* eller *AscW()* og datatypen er *Integer*. Forskjellen er den samme som på *Chr()* og *ChrW()*:

```
Dim tegnkode As Integer
tegnkode = Asc("a") 'gir tallet 97
tegnkode = AscW("a"c) 'gir også 97
tegnkode = Asc("abcdef") 'gir også 97
tegnkode = Asc("") 'gir feil
```

[26] I mange sammenhenger i VB vil en streng som er *Nothing* regnes som en tom streng (med verdi ""). Derfor er det faktisk *tillatt*, men risikabelt, å sjekke om en streng er *Nothing* med likhet:

```
If navn = Nothing Then MsgBox("Strengen eksisterer ikke!") 'tillatt men risikabelt!
```

Her blir *Asc("a")* og *AscW("a")* like, fordi "a" er plassert på samme sted i de to tegntabellene. Legg merke til at funksjonene kan ta både et tegn og en streng som argument og strengen kan være mer enn ett tegn (da gir den tegnkoden for første tegn i streng).

Noen ferdigdefinerte tegnkonstanter

Navn	Tegnkode	Alternativ	Betydning
vbCr	13	Chr(13)	Tilbake til starten av linjen (*carriage return*), men bare hvis vbLf kommer rett etterpå.
vbLf	10	Chr(10)	Ny linje (*linefeed*), men bare hvis den kommer rett etter vbCr.
vbNullChar	0	Chr(0)	Tegn med tegnkode 0 skriver ingenting. OBS! Dette er ikke det samme som en tom streng "".
vbTab	9	Chr(9)	Tabulator (*tab*) flytter til neste tabulatorstopp.
vbBack	8	Chr(8)	Ett tegn til venstre (*back*). Virker ikke i tekstfelt.

Strenger (String)

Å deklarere strenger bør nå være velkjent:

Objekter kan ha fått et navn (med *Dim*) uten å ha fått plass i RAM. Det får de først med *New*. Strenger er litt spesielle objekter, for de kan også få plass i RAM med en tilordning.

```
Dim etternavn As String = "Hansson"
Const feilmeld As String = "Noe har skjedd!"
Dim etternavn As String = New String("Hansson")
Dim likhetstegn As String = New String("=", 15)
```

Det er mange forskjellige argumenter som kan brukes i *New String()*, her vises de to vanligste. Den første er vel selvforklarende, den andre lager en streng med 15 likhetstegn. Når VB ikke er så nøye på om du skriver *New*, så skyldes det at strenger brukes så mye. Microsoft har skrevet kompilatoren slik at den legger det til selv.

Et slikt strengobjekt har to egenskaper, nemlig *Length* og *Chars*. *Length* er en *Integer* og angir hvor mange tegn strengen består av. *Chars* er en array med tegn (altså *Char()*), slik at vi kan få tegnene i en streng ved å skrive f.eks.

```
tegn = etternavn.Chars(1)
```

…som gir tegnet med indeks 1 (indeksene starter på 0 som vanlig i arrays), dvs. "a" i "Hansson".

Siden *String* har *Chars* som standardegenskap[27] (*Default Property*), kan vi også bare skrive

```
tegn = etternavn(1)
```

så vil kompilatoren anta at vi mener *etternavn.Chars(1)*.

Begge egenskapene (*Length* og *Chars*) er variabel som kun kan leses (*ReadOnly*). Derfor er det *ikke* tillatt å skrive

```
etternavn.Length = 15 'prøver å utvide strengen
etternavn.Chars(2) = Chr(98) 'prøver å tilordne et tegn verdien 'b'
```

Hvordan kan da VB gi en streng ny verdi (med ny lengde)? Microsoft har løst dette ved at hver gang en strengvariabel gis ny verdi, så opprettes det et nytt strengobjekt med den nye verdien og lengden. Den gamle slettes[28].

```
etternavn = "Hansson"
etternavn = "Pettersen"
```

[27] Standardegenskaper var vanlig i tidligere versjoner av VB. De er nå sjeldne, og brukes bare når de krever et argument, slik som her hvor *Chars()* må ha et heltall som argument. De gjør det mulig å skrive koden uten å angi hvilken egenskap det er snakk om – kompilatoren antar da at det er standardegenskapen vi mener – her *Chars()*.

[28] Man sier da at strenger er *immutable* dvs. de er uforanderlige.

Den første linjen skaper et strengobjekt med navnet *etternavn* som gis verdien ”Hansson”. Den andre linjen sletter det første strengobjektet og skaper et *nytt* strengobjekt med samme navn og verdien ”Pettersen”. Dette tar litt tid, men ellers merker vi ikke så mye til det. Variabelen *etternavn* viser hele tiden til det strengobjektet som har den aktuelle verdien og lengden.

Strengkonstanter

En strengkonstant uten navn, omsluttes av anførselstegn:

```
MsgBox("Heisann", , "Melding")
```

Det finnes også noen ferdig definerte strengkonstanter, f.eks.:

Konstant	Verdi
vbCrLf ≈ **vbNewLine**	Vognretur og ny linje, altså vbCr & vbLf eller Chr(13) & Chr(10). vbCrLf har alltid denne verdien, mens vbNewLine er plattformavhengig og kan ha verdien Chr(13)+Chr(10) eller bare Chr(10) (passer for Unix/Linux og Mac OS X).
vbNullString	En streng (*String*) med ett tegn, nemlig Chr(0) eller vbNullChar. Det er noe *annet* enn en tom streng – denne har ett tegn. Det kreves i visse prosedyrekall utenfor VB, f.eks. når de er skrevet i et språk som ikke tillater tomme strenger.

Anførselstegn inne i strenger

Siden faste strenger omsluttes av anførselstegn, får vi et spesielt problem hvis vi vil ha anførselstegn inn som en del av strengen, f.eks. fordi vi vil ha skrevet ut følgende:

”Heisann!”, sa hun

Her er det to anførselstegn i strengen. Vi legger inn anførselstegn i strengen enten ved å skrive anførselstegnet to ganger, eller ved å bruke funksjonen *Chr(34)* – se nedenfor under konvertering.

```
Dim tekst As String
tekst = """Heisann!"", sa hun"
tekst = Chr(34) & "Heisann!" & Chr(34) & ", sa hun"
tekst = Asc("""") 'tegnkoden blir 34 = anførselstegn
```

Konkateneringsoperator

Konkatenere (lat. *con* = sammen + *catenere* = binde).

Den vanligste operatoren for strenger er konkatineringsoperatoren & som setter de to strengene sammen til én (det *er* lov til å bruke + isteden, men det ser jeg bort fra her).

```
melding = "Jeg mener at " & mening 'melding og mening er strenger
```

Man kan også se &= som en operator, nemlig en sammenslåing av & og tilordning. Disse er like (*melding* og *mening* er strenger):

```
melding = melding & mening
melding &= mening
```

Sammenlikningsoperatorer

Alle de vanlige sammenlikningsoperatorene (>, <, =) kan brukes for strenger. Sammenlikningen skjer tegn for tegn og igjen er det forskjell avhengig av *Option Compare*. Med *Option Compare Binary* er ”absolutt” større enn ”Absolutt” fordi ANSI-koden for ”a” er større (97) enn koden for ”A” (65). Med *Option Compare Text*, er de to strengene like.

Hvis du vil sammenlikne to strenger *uten* å ta hensyn til store/små bokstaver[29] til tross for at du har satt *Option Compare Binary*, kan du konvertere begge til bare små eller begge til bare store før sammenlikningen – se under strengfunksjoner.

[29] Kalles versaler og minuskler i grafisk fagspråk.

En annen, nyttig sammenlikningsoperator er *like*. Den sammenligner strenger med et mønster, og gir True hvis mønsteret finnes i strengen. Mønsteret skrives som en streng, der visse tegn er såkalte "jokertegn" (*"wild card characters"*). Det er tre slike jokertegn:

Jokertegn	**Betydning**
?	Akkurat ett, hvilket som helst, tegn
*	Ingen, ett eller flere tegn
#	Akkurat ett siffer

I tillegg kan man liste opp et tegn i hakeparenteser, og det betyr da at på en bestemt plass i strengen skal det være ett av disse tegnene: *[abc]*, *[a-f]*, *[a-kp-å]*. Den siste – merk at den er uten komma – innebærer at tegnet skal være enten a-k eller p-å, altså ikke l-o og det kunne vi også skrevet med "ikke" som er et utropstegn: *[!l-o]*.

Eksempler: `Dim navn As String = "Knut W. Hansson"`

`MsgBox(navn Like "*ut*")`	Et antall tegn etterfulgt av "ut" etterfulgt av et antall tegn	True
`MsgBox(navn Like "K*")`	Begynner med 'K' etterfulgt av et antall tegn	True
`MsgBox(navn Like "K[m-o]?t*")`	Begynner med K, deretter 'm', 'n' eller 'o', deretter akkurat ett tegn, så en 't' og deretter et antall tegn	True
`MsgBox(navn Like "*[G,H]*")`	Et antall tegn etterfulgt av enten 'G' eller 'H' og deretter et antall tegn = Finnes enten 'G' eller 'H' i *navn*?	True
`MsgBox(navn Like "*[g,h]*")`	Et antall tegn etterfulgt av enten 'g' eller 'h' og deretter et antall tegn = Finnes enten 'g' eller 'h' i *navn*?	True med *Option Compare Text* (da er 'h' = 'H') og False med *Option Compare Binary* (da er 'h' <> 'H').

Strengfunksjoner

1) Konvertering fra andre datatyper til streng

De fleste datatyper og alle objekter har metoden *ToString()* som konverterer til en streng, f.eks.

```
Dim tall As Double = 15.2
Dim tallstreng As String = tall.ToString()
MsgBox(tallstreng) 'gir teksten "15,2" (merk komma)
```

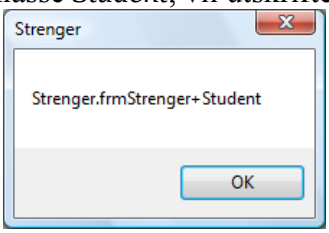
Hvis vi lager egne klasser, må vi selv definere hva vi mener med *ToString()* ellers får vi en streng vi ikke blir fornøyd med. Her er det gjort i klassen *student* – merk at *ToString()* er definert for alle klasser, arvet fra klassen *Object*. Derfor må den merkes *Overrides*:

```
Public Class Student
    Public studnr As Integer
    Public studnavn As String
    Public Overrides Function ToString() As String
        Return studnr.ToString() & " " & studnavn
    End Function
End Class
```

Slik vil det se ut når et objekt skapes og skrives i tekskontrollen:

```
Dim stud As Student = New Student()
stud.studnr = 12
stud.studnavn = "Knut"
txtUt.Text &= stud.ToString()
```

Hvis vi ikke definerer *ToString()* for en klasse *Student*, vil utskriften av et blitt slik:



Videre har vi konverteringsfunksjonene *Str()* og *Cstr()* – den første er internasjonal, den andre er *Local Aware*:

```
Dim tall As Double = 15.2
txtUt.Text &= tall.ToString() 'gir teksten "15,2" (merk komma)
txtUt.Text &= CStr(tall) 'gir teksten "15,2" (merk komma)
txtUt.Text &= Str(tall) 'gir teksten " 15.2" (merk mellomrom og punktum)
```

2) Konvertering fra streng til andre datatyper

Hvis du søker i VB hjelp etter *type conversion functions* vil du finne de fleste funksjoner som konverterer, herunder *CBool()*, *CByte()*, *CDbl()*, *CDec()*, *CInt()*, *CLng()* og *CSng()*. Alle disse kan ta en streng som argument, mens funksjonen *CDate()* som konverterer til en dato, ikke godtar det. Alle disse kan kaste feil – bruk feilfellen *try-catch*.

En spesiell komverteringsfunksjon er *Val()*. Den har den fordelen at den ikke feiler – isteden leser den strengen til den finner noe som den ikke gjenkjenner som et tall og gir opp der. Hvis teksten er helt ugjenkjennelig, returnerer *Val()* tallet 0. Ulempen er at *Val()* krever punktum som desimalskilletegn.

```
MsgBox(Val("15.32 og ikke ett øre mer!")) 'gir verdien 15,32
MsgBox(Val("Femten")) 'gir verdien 0
```

De fleste standard datatypene har også metoden *Parse()* og *TryParse* som konverterer fra streng, f.eks.:

```
Dim tall As Double
Dim OK As Boolean
tall = Double.Parse("15,32") 'Parse returnerer tallet
OK = Double.TryParse("15,32", tall) 'TryParse returnerer True/False
```

Parse() kan gi feil, hvis den ikke klarer jobben, og den bør derfor "beskyttes" med *IsNumeric()* eller *Try-Catch*. *TryParse()* returnerer verdien gjennom det andre parameteret (som er deklarert *ByRef*) og gir funksjonsverdien *True*. Hvis *TryParse()* ikke klarer å konvertere til tall, blir det andre parameteret satt til 0 og funksjonsverdien blir *False* uten at det gir feil.

3) Gjøre om på strengen

Det finnes et meget stort antall funksjoner, både selvstendige funksjoner, og funksjoner knyttet til strengobjekter og til String-klassen. Det vil føre alt for langt å ta med alle her. Jeg har valgt ut noen som kan være særlig nyttige.

a) Finne en delstreng/substreng

Hvis vi leter etter en bestemt substreng, kan vi bruke funksjonen

```
Public Function IndexOf (value As String, startIndex As Integer, _
        count As Integer) As Integer
```

Funksjonsverdien angir hvilket tegnnummer en substreng *value* begynner på. Vi leter fra *startIndex* og *count* tegn utover. Her kan *count* utelates (da letes det til og med siste tegn) og da kan også *startIndex* utelates (da letes det fra starten). Hvis det er flere forekomster, angir funksjonen bare første forekomst, og hvis *value* ikke finnes, returneres -1.

```
        Dim funnet As Integer
        tekst = "Hoppsasa"
        funnet = tekst.IndexOf("sa") ' gir indeks 4
        funnet = tekst.IndexOf("sa", 6) ' gir indeks 6
        funnet = tekst.IndexOf("hei") 'gir indeks -1 (ikke funnet)
        funnet = tekst.IndexOf("sa", 0, 5) ' gir -1 (ikke funnet)
```

b) Velge et utdrag av strenger = delstrenger/substrenger

```
Public Function Substring (startIndex As Integer, length As Integer) As String
```

Funksjonen returnerer en del av strengen fra *startIndex* og *length* tegn utover (*length* kan utelates, da returneres resten av strengen):

```
        Dim tekst As String = "Hoppsasa"
        Dim delstr As String
        delstr = tekst.Substring(4) 'returnerer "sasa"
        delstr = tekst.Substring(4, 2) ' returnerer ""sa"
```

c) Snu strengen

Denne funksjonen snur strengen så tegnene kommer i motsatt rekkefølge:

```
Public Function StrReverse(ByVal Expression As String) As String
        delstr = StrReverse("Knut") 'gir "tunK"
```

d) Fjerne/legge til tegn i hver ende

For å **fjerne** bestemte tegn i hver ende av strengen, kan vi bruke:

```
Public Function Trim (ParamArray trimChars As Char()) As String
Public Function TrimEnd (ParamArray trimChars As Char()) As String
Public Function TrimStart (ParamArray trimChars As Char()) As String
```

Funksjonen fjerner alle tegn lik *trimChars* i begge ender (*Trim*), bare bakerst (*TrimEnd*) eller bare forrest (*TrimStart*). Du kan utelate *trimChars* og da fjernes mellomrom.

```
        tekst = "..Knut...."
        delstr = tekst.TrimEnd("."c) 'gir "..Knut"
        delstr = tekst.TrimStart("."c) 'gir "Knut...."
        delstr = tekst.Trim("."c) 'gir "Knut"
```

Det er enkelt å **legge til** tegn i hver ende:

```
Public Function PadLeft (totalWidth As Integer, paddingChar As Char) As String
Public Function PadRight(totalWidth As Integer, paddingChar As Char) As String
```

Her angir *totalWidth* hvor mange tegn det skal være i strengen etter operasjonen, og *paddingChar* det tegnet som skal legges til (kan utelates og da legges det til mellomrom). *PadLeft* legger til tegn foran og *PadRight* bakerst:

```
Dim tekst As String = "Hurra"
tekst = tekst.PadRight(10, "."c) 'gir "Hurra....."
```

Å legge til tegn inne i strengen, gjøres med

```
Public Function Insert (startIndex As Integer, value As String) As String
```

Strengen *value* settes inn fra og med *startIndeks*, f.eks.

```
tekst = "Knopp"
delstr = tekst.Insert(2, "app") 'gir "Knappopp"
```

Å legge til en streng bak eller foran en annen, gjør vi selvsagt med operatoren &.

e) Fjerne tegn inne i strengen

Vi fjerner tegn inne i en streng, med

```
Public Function Remove (startIndex As Integer, count As Integer) As String
```

F.eks. kan vi fjerne ”app” i ”Knappopp” slik:

```
tekst = "Knappopp"
tekst = tekst.Remove(2, 3) 'gir "Knopp"
```

f) Erstatte visse tegn i strengen med andre

```
Public Function Replace (oldValue As String, newValue As String) As String
```

Her skal f.eks. alle p-ene i ”Knapp” erstattes av ”t”:

```
tekst = "Knapp"
tekst = tekst.Replace("p"c, "t"c) 'gir "Knatt"
```

4) Lage en streng

Du kan lage en streng som består av mange like tegn, ved å bruke *New*, f.eks.

```
tekst = New String("$"c, 5) 'gir "$$$$$"
tekst = New String(Chr(32), 4) 'gir "    " (en streng med fire mellomrom)
```

Note: 32 er tegnkoden for mellomrom.

5) Formattere en streng

Formattering vil si å gi strengen en bestemt form. De enkleste er *ToLower* og *ToUpper* som gjør om alle tegnene til enten store eller små bokstaver:

```
Dim navn As String = "Knut"
navn = tekst.ToUpper() 'gir KNUT
navn = tekst.ToLower() 'gir "knut"
```

En meget fleksibel formattering gjør du med funksjonen

```
Function Format(ByVal Expression As Object, _
       Optional ByVal Style As String = "") As String
```

Du oppgir som *Expression* det som skal formatteres – det kan være i prinsippet en hvilken som helst datatype. Deretter oppgir du hvordan du vil ha det formattert med argumentet *Style*. Du står meget fritt til å lage dine egne formater, men generelt anbefaler jeg å bruke de standardformatene som VB tilbyr, f.eks. ”Currency”, ”Percent”, ”Short Date” og ”Short Time”:

```
tekst = "3,14159"
tekst = Format(tekst, "Currency") 'gir "kr 3,14"
tekst = Format(tekst, "Percent") 'gir "314,16%"
Dim tid As Date = #1/31/2007 1:15:59 PM#
teskt = Format(#1/31/2007 1:15:59 PM#, "Short Date") 'gir "31.01.2007"
tekst = Format(#1/31/2007 1:15:59 PM#, "Long Time") 'gir 13.15.59
```

Ved å lage formatstrengen selv, får du svært høy fleksibilitet. Formatstrengen brukes som en ”mal”, der enkelte tegn har spesiell betydning. Noen av de viktigste for tall er:

Tegn	Betydning
0	Plass til et siffer, sett evt. inn ”0”
#	Plass til et siffer ved behov.
.	Plass til desimaltegn
,	Plass til tusenskiller
”<streng>”	Sett inn strengen

```
Dim tall As Double = 15.3869
tekst = Format(tall, "Ditt tall: ##0.0##") 'gir "Ditt tall: 15,387"
```

6) Dele en streng i mange delstrenger og slå sammen mange delstrenger

Særlig når vi leser fra en fil med data skilt med et skilletegn (f.eks. en kommaseparert tekstfil), er det aktuelt å dele en streng i mange delstrenger. Strengen skal deles ved et gitt skilletegn (som ikke må være med i noen av delstrengene).

```
Public Function Split (ParamArray separator As Char()) As String()
```

Legg merke til at parameteret er tegn, og det kan være flere (i en tegn-array). Legg også merke til at returverdien er en streng-array. Den kan deklareres uten at vi angir hvor mange strengelementer det skal være plass til – for det vet vi ofte ikke. Her simulerer vi at *strLest* er lest fra en slik fil:

```
Dim strTabell() As String
Dim strLest As String = "12,Knut,14,Erik,19,Petter"
strTabell = strLest.Split(","c) 'gir seks strenger i strTabell
Dim antall As Integer = strTabell.Length 'blir 6
```

Den ”motsatte” funksjonen er *Join*. Den slår sammen delstrenger til en streng, og legger skilletegn mellom hver delstreng:

```
Public Shared Function Join (separator As String, value As String()) As String
```

Her er det altså anledning til å oppgi en streng som skilletegn (altså kan det være flere tegn) og det andre argumentet er en streng-array med de elementene som skal settes sammen. Denne vil bygge opp igjen den *strLest* som vi splittet ovenfor:

```
strLest = String.Join(",", strTabell) 'gir "12,Knut,14,Erik,19,Petter"
```

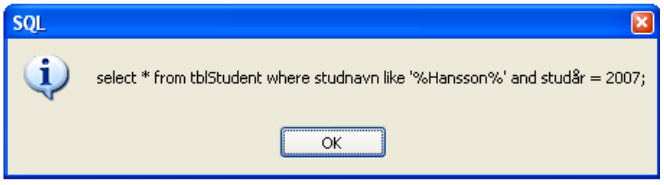
7) Blande fast tekst med variabel

Det er ofte bruk for å legge tekster fra variabel inn i en fast streng. Anta f.eks. at brukeren har et felt *txtNavn* der han kan skrive inn hele eller deler av et studentnavn. Brukeren har skrevet inn teksten ”Hansson”. Videre har brukeren mulighet til å velge et studieår, som vi har lagt i variabelen *år*. Verdien av den er her 2007.

Vi skal søke i en database – i tabellen *tblStudent* – etter studenter med det oppgitte navnet som er studenter i det oppgitte året. I SQL skal strenger omsluttes av enkle anførselstegn (”enkeltfnutter”) og det krever litt ekstra. I tilfelle brukeren skrev bare en del av navnet, skal vi søke med *like* og legge til jokertegnet "%" som i SQL angir ingen, ett eller flere tegn:

```
Dim strSql As String
Dim år As Integer
……
strSql = "select * from tblStudent where studnavn like '%" _
    & txtNavn.Text & "%' and studår = " & år.ToString() & ";"
……
MsgBox(strSql, MsgBoxStyle.Information, "SQL")
```

Den meldingsboksen som fremkommer, ser slik ut:

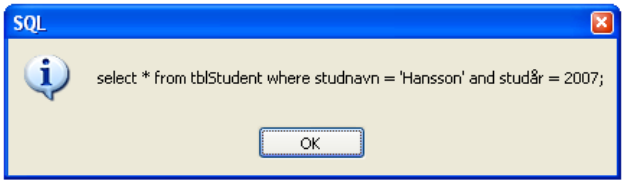


Dette er en fin måte å sjekke hva som faktisk senes til databasen. Ofte får du en feil fra databasen eller galt resultat og den kan du kanskje finne slik.

Vi kan f.eks. fjerne prosenttegnene og bytte ut *like* med likhetstegn ved å skrive:

```
strSql = strSql.Replace("%", "") 'fjerner alle "%"
strSql = strSql.Replace("like", "=") 'erstatter like med =
MsgBox(strSql, MsgBoxStyle.Information, "SQL")
```

Da ser meldingsboksen slik ut:



Tillegg: Hvilken funksjon bør jeg bruke?

I mange situasjoner finnes det flere måter å skrive koden på. Her er noen eksempler på konverteringer (linjene er nummerert så de kan henvises til i teksten under):

```
1  Dim tall As Double = 15.3
2  Dim tekst As String = "23,7"
3  tall = Double.Parse(tekst)
4  tall = CDbl(tekst)
5  tekst = tall.ToString()
6  tekst = CStr(tall)
```

I linje 3 bruker jeg en metode som hører til *klassen* Double. Det er derfor jeg må oppgi *klasse*navnet. I linje 4 bruker jeg en *ferdig, frittstående funksjon* i VB. Disse virker likt, og gir samme resultat. Det viser seg riktignok at *Parse()* er raskere enn *Cdbl()*, noe som kanskje kommer av at *CDbl()* nødvendigsvis er programmert mer generelt, siden den kan ta argumenter av mange forskjellige typer, mens *Parse()* kun kan ta streng-argumenter. Forskjellen er likevel bare at det tar ca 400 millisekunder mot 600 millisekunder å konvertere en million ganger. Det skal mye til at dette betyr noe i en applikasjon.

I linje 5 bruker jeg en metode som hører til *objekter* i klassen Double (*tall* er et slikt objekt). I linje 6 bruker jeg en f*erdig, frittstående VB-funksjon* fra den "gamle" VB 6.0. Disse gir samme resultat og er omtrent like raske (ca 600 mot 630 millisekunder på en million konverteringer).

Hvilken av disse man bør bruke, er da en ren smaksak. Det kan være greit å ha sett begge variantene, fordi begge dukker opp i andre språk og da har du kanskje ikke noe valg. Jeg har i alle fall ingen bestemte preferanser.
Noen ganger kan vi velge mellom en metode eller en operator. Disse er f.eks. like:

```
tekst = String.Concat("Knut", "Hansson")
tekst = "Knut" & "Hansson"
```

Her vil jeg personlig klart foretrekke operatoren, fordi jeg synes den er enklere å forstå. Hva synes du f.eks. selv om disse to (*inn1* og *inn2* er strengvariable)?

```
tall = Double.Parse(String.Concat(inn1, inn2))
tall = CDbl(inn1 & inn2)
```

Tema E2 – Bruk av VB Hjelp

Når du slår opp i VB Hjelp, kan du få gode forklaringer og en grundig syntaks. Her er en liten brukerveiledning.

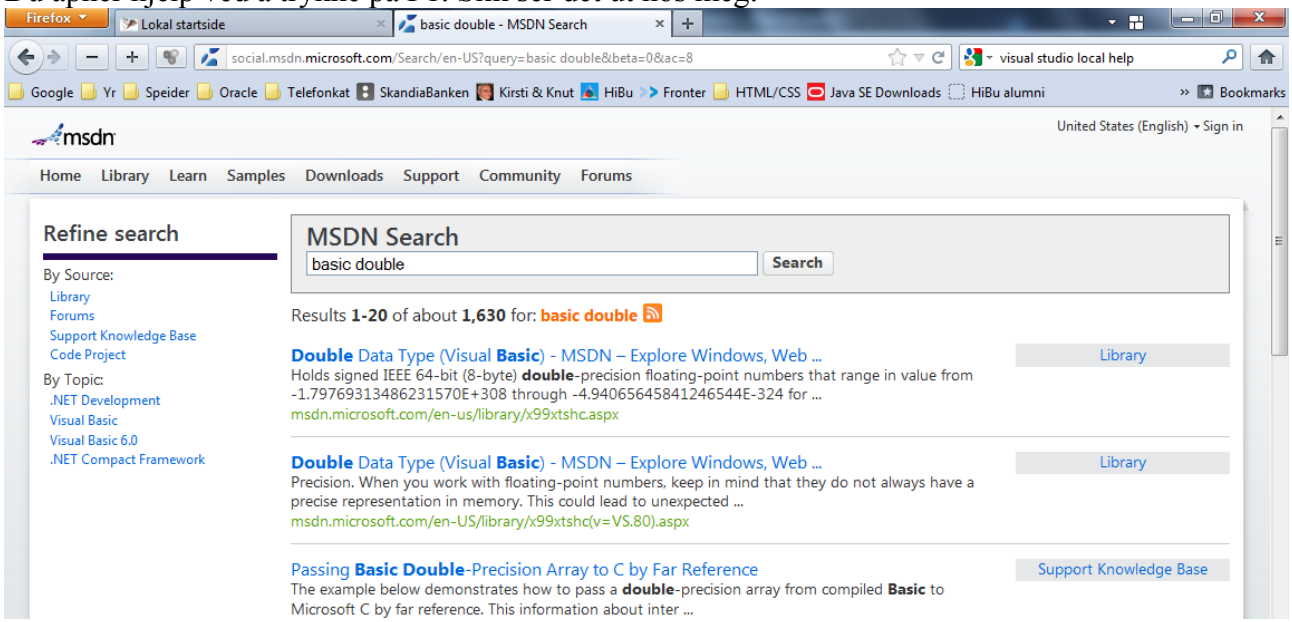
Visual Basic 2010 Express er en liten del av Visual Studio .NET miljøet. Visual Studio inneholder flere programmeringsspråk:

- ✓ Visual Basic (som vi bruker her)
- ✓ Visual C++ (et annet objektorientert språk)
- ✓ Visual C# (”C sharp”[30] – Microsofts variant av C++. Vises i et senere kurs.)
- ✓ Visual F# ("F sharp" – først og fremst et funksjonsorientert språk, for kunstig intelligens)
- ✓ ASP (Brukes når applikasjonen er delt på klient og tjener via Internett. Pensum i et senere kurs.)

Alle kompilerer til samme mellomkode, og programmeringsmiljøene likner svært på hverandre. De tre første finnes i gratis *Express* versjon.

Visual Basic 2010 Express er altså et *utdrag* av Visual Studio. Det kan skape litt forvirring, fordi enkelt ting bare gjelder for andre språk. Pass derfor på at du bare bruker hjelp som er ment for Visual Basic 2010.

Du åpner hjelp ved å trykke på F1. Slik ser det ut hos meg:

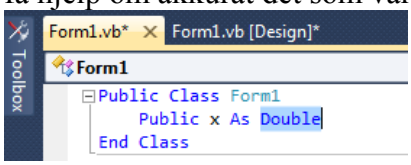


Som det fremgår i adressefeltet, så er denne hentet fra Internett. Det er mulig (hvis du er offline) å se lokal hjelp – klikk menyen *Help*, velg *Manage Help Settings* og sett hjelpen til lokal hjelp. Den ser omtrent likedan ut, men vedlikeholdes ikke slik som Internetthjelpen gjør.

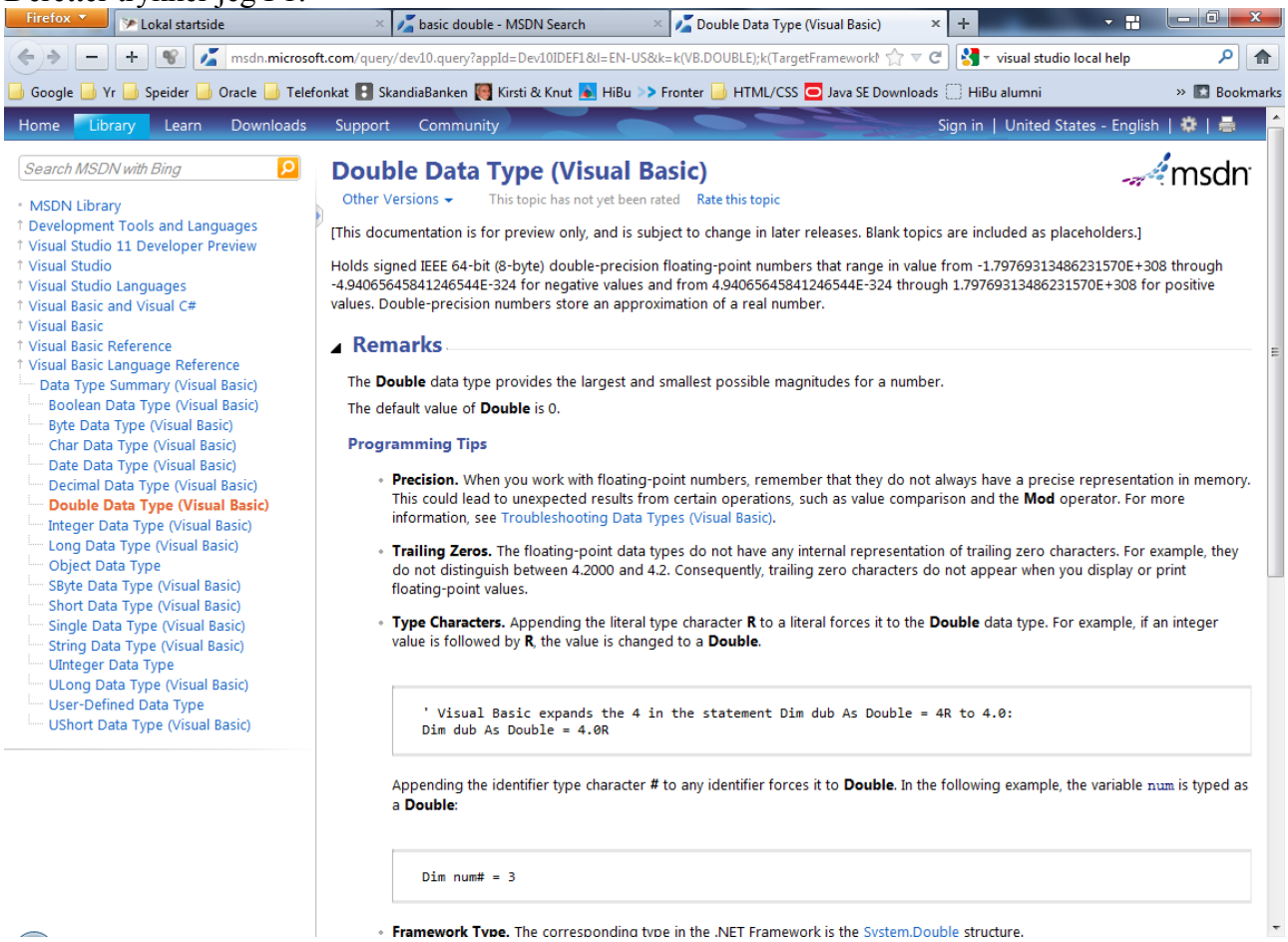
Hjelp viser hjelp fra hele .NET-miljøet. Av til kan det lønne seg å søke etter "Basic double" heller enn bare "double".

[30] C# er egentlig en toneart i musikk – på engelsk sier man ”C sharp” men på norsk vil den hete ”Ciss”.

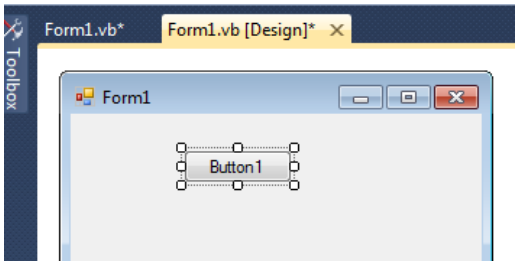
Hvis du lurer på et ord du har skrevet inn i koden, kan du merke ordet før du trykker F1. Da vil du få hjelp om akkurat det som var merket. Her merker jeg ordet *Double*:



Deretter trykker jeg F1:



Tilsvarende kan du merke en kontroll før du trykker F1.



Ofte finner jeg mye nyttig nederst under *See Also*. For *Double* står det f.eks.

See Also

Tasks

Troubleshooting Data Types (Visual Basic)

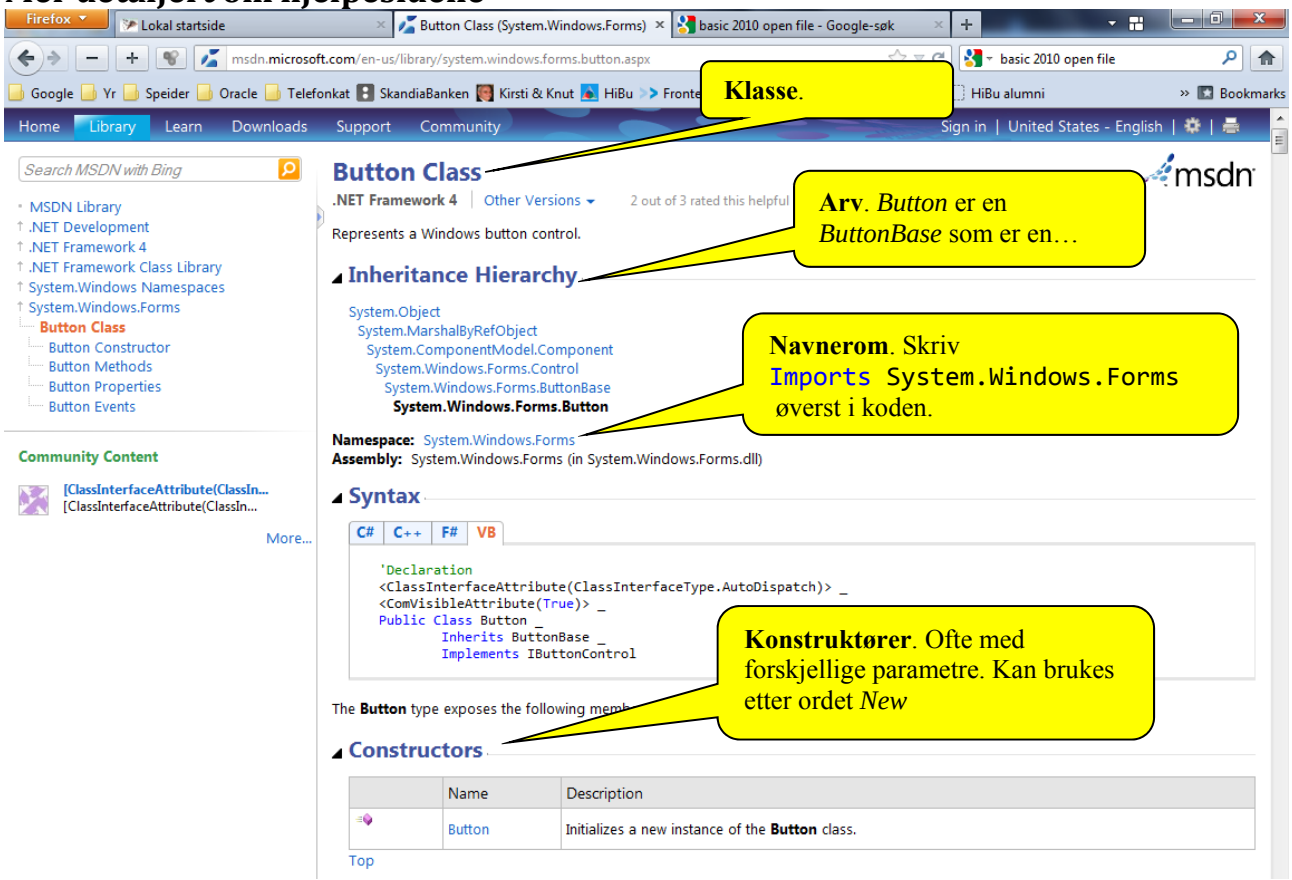
Reference

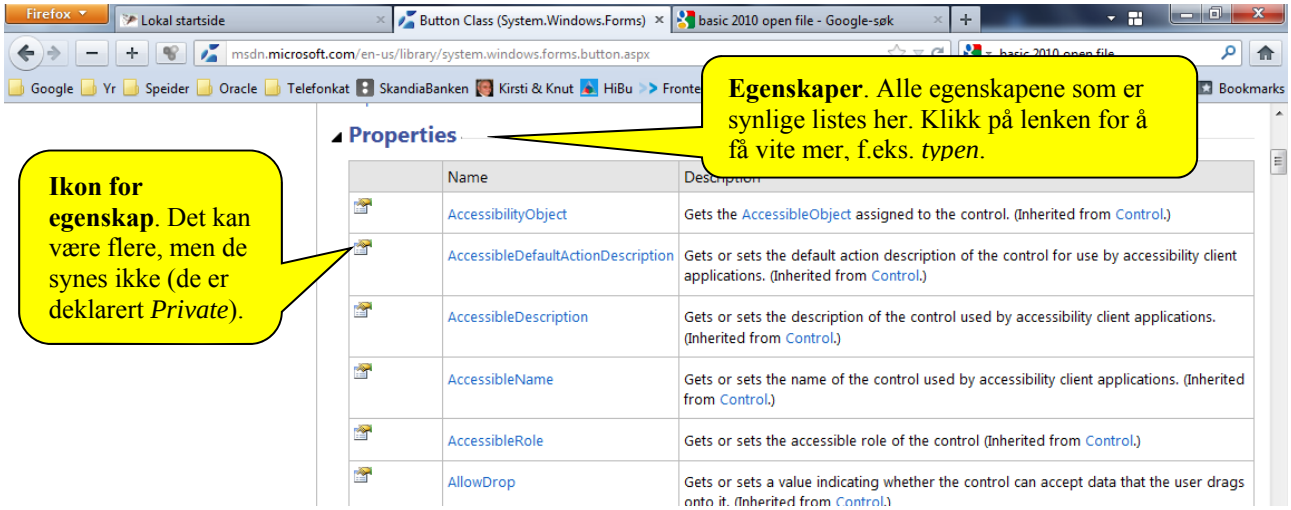
Data Type Summary (Visual Basic)
System.Double
Decimal Data Type (Visual Basic)
Single Data Type (Visual Basic)
Type Conversion Functions (Visual Basic)
Conversion Summary (Visual Basic)

Concepts

Efficient Use of Data Types (Visual Basic)
Type Characters (Visual Basic)

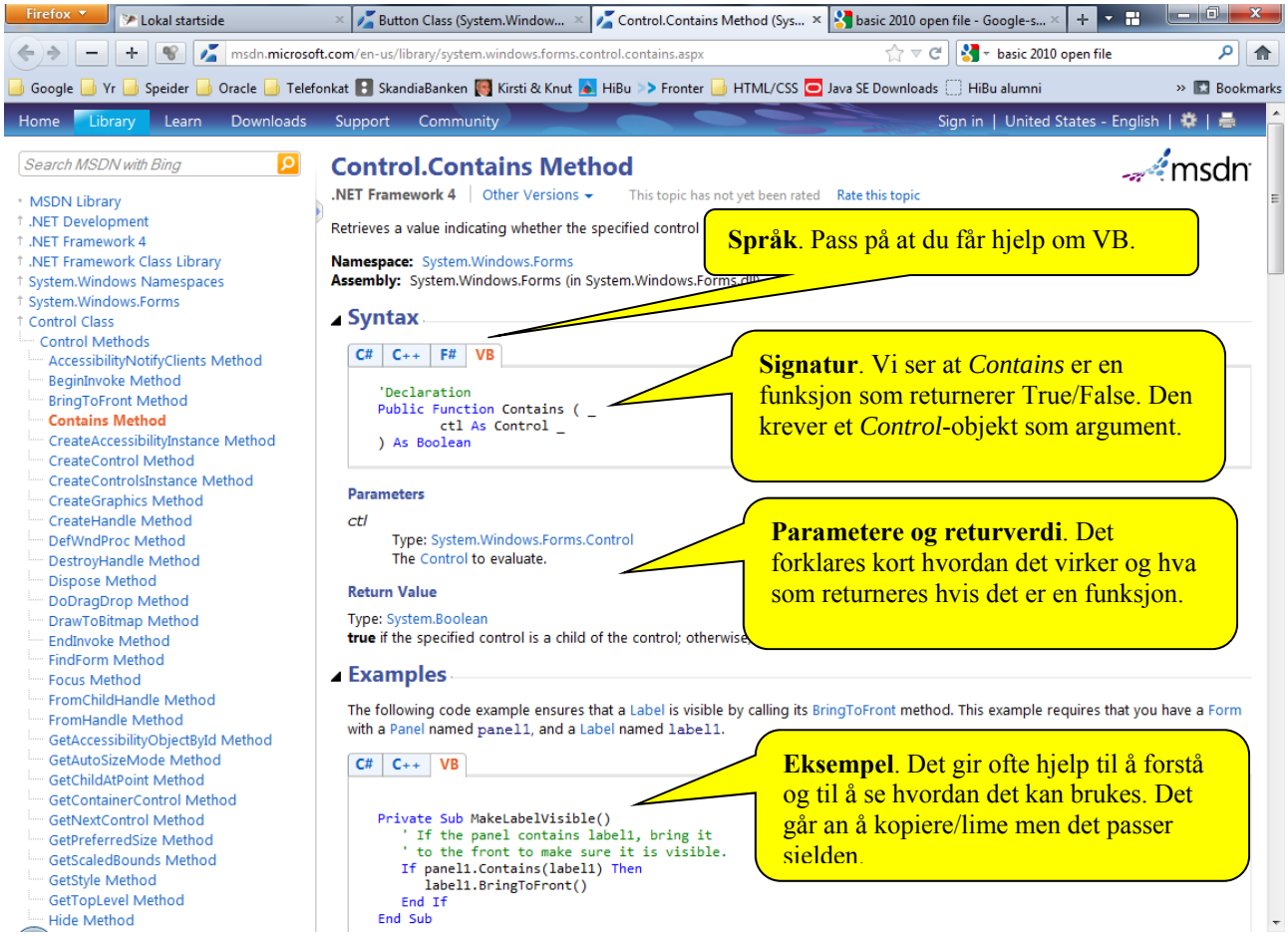
Mer detaljert om hjelpesidene

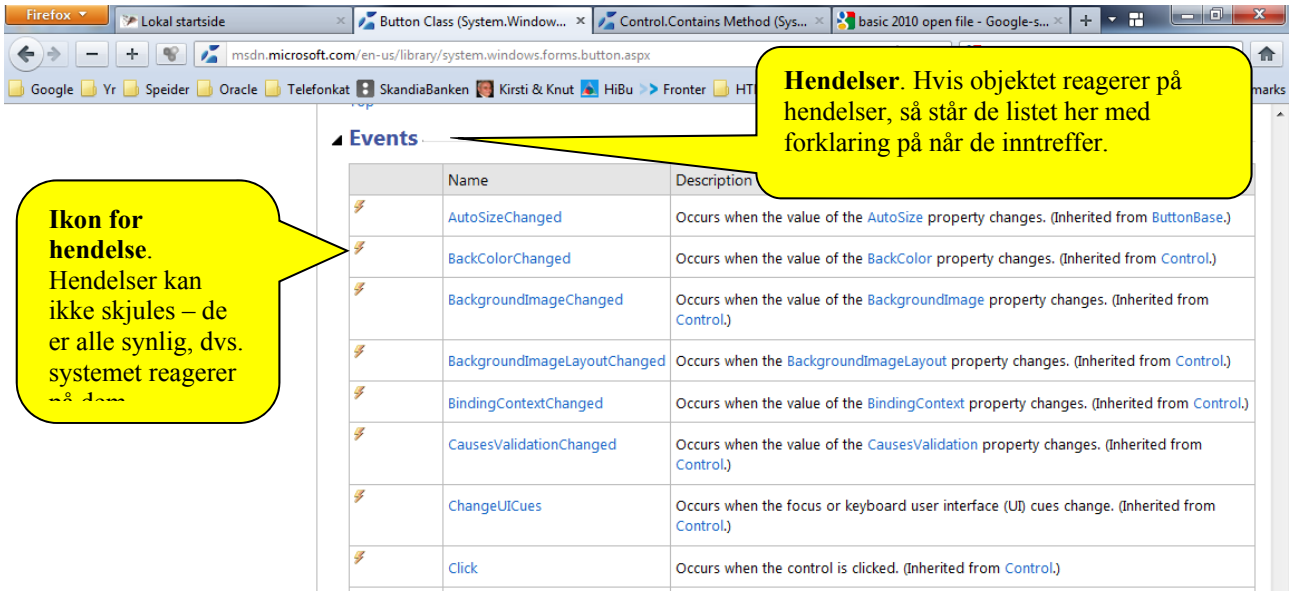




Det er også forskjell på metoder som kan brukes av objekter, og metoder som bare kan brukes av klassen selv. De siste er merket *Shared* med en stor **S**. De må brukes sammen med *klassenavnet*.

Her har jeg klikket metoden *Contains* for å få vite mer om den:





Konklusjon

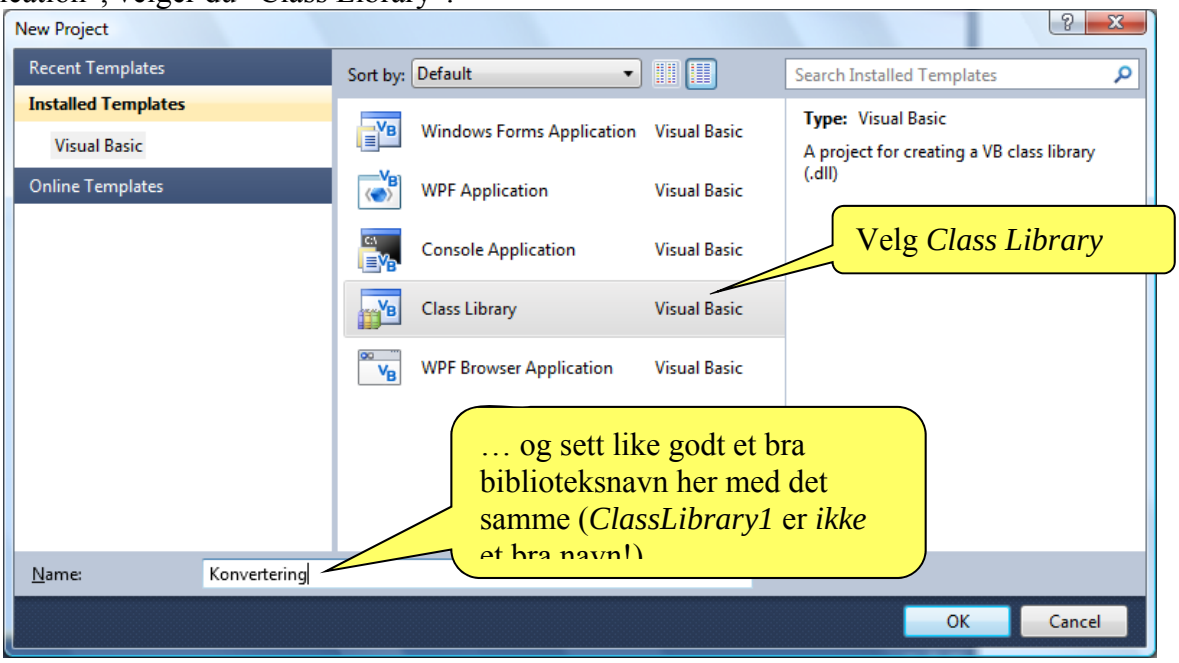
Det er veldig mye bra hjelp å få i VB hjelp, men det er så mye at det kan være forvirrende. Hvis du bruker den aktivt, vil du etter hvert erfare hvor du bør slå opp, og det er en svært nyttig del av programmeringskunnskapene.

Jeg anbefaler derfor at du bruker hjelp mye.

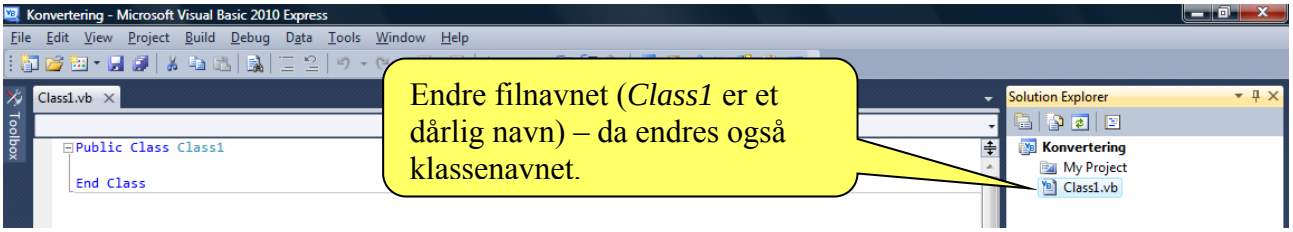
Tema F1 – Klasse- og programbibliotek (dll-filer)

En meget vanlig, og nyttig måte å øke gjenbruk på, er med klasse- og programbibliotek (dll-filer). Vi skal se på hvordan slike lages – og brukes – med et eksempel. Vi tenker oss en bedrift som skiver alle navn på en bestemt form (”Hansen, Per Erik”), og det skjer i så mange programmer at man vil lage et generelt bibliotek for dette. Det kan være flere fornavn (minst ett) og flere etternavn (minst ett).

For å lage et bibliotek, starter du et nytt prosjekt, men – istedenfor å velge det vanlige ”Windows Application”, velger du ”Class Library”.



Du får et prosjekt *uten* skjema, og det er jo rimelig for et programbibliotek. Isteden får du kodevindu med en klasse:



Her bør du endre filnavnet og klassenavnet med det samme. Det kan være flere klasser i et programbibliotek – du legger til flere med menyen *Project|Add Class*. Du bør lagre alt med det samme, så mister du ikke for mye arbeid hvis programmet henger e.l.

Så skriver du koden for klassen, altså medlemmene (attributter, prosedyrer og funksjoner) på vanlig måte. Det vil ikke bli noen hendelsesprosedyrer (*handles…*) fordi det ikke er noe grensesnitt til brukeren. Tilsvarende bør biblioteket *ikke* kommunisere med brukeren gjennom input- og meldingsbokser. Biblioteket skal hjelpe applikasjonsprogrammer og man overlater til dem å "snakke" med brukeren – de har jo alltid minst ett skjema.

Man må tenke igjennom hvilken synlighet medlemmene skal ha – noe skal være Public så det kan sees utenfra (i det applikasjonsprogrammet som bruker biblioteket) og noe skal kanskje være skjult (Private) så det bare kan brukes innenfor denne klassen (”hjelpevariable” og ”hjelperutiner”).

Hvis et medlem skal kunne kalles fra *klassen* må det deklareres *Shared*. Da kalles medlemmet et *klassemedlem.* Hvis medlemmet ikke er deklarert *Shared*, så er det et *objektmedlem.* Da må det programmet som skal bruke den, skape et *objekt* som bruker medlemmet. I eksemplet nedenfor skal jeg vise begge dele.

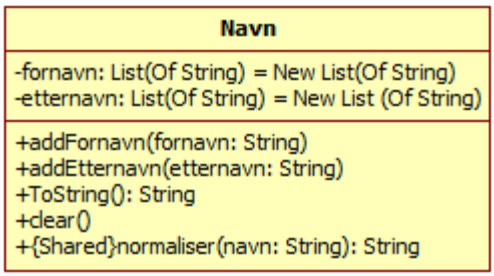
Class eller Module?

Både *module* og *class* representerer et *navnerom* (name space), og begge kan kompileres til et programbibliotek. Da kan man distribuere bare den kompilerte versjonen. Det meste av det som Microsoft har laget til oss i VB er distribuert på denne måten. Da kan vi ikke se hvordan de har gjort det (kildekoden). I begge tilfelle kan man referere til dem i et annet program, og bruke metoder og variable som er definert der. Forskjellen er at med en *klasse* kan det skapes *objekter* av den definerte typen. De kan inneholde attributter, prosedyrer og funksjoner. Hvis man f.eks. deklarerer en *klasse* som inneholder variabelen *Public navn As String*, vil hvert objekt som skapes ha sitt eget *navn*. Hvis man deklarer *Public navn As String* i en *modul*, vil det bare finnes én variabel *navn* som er synlig i hele programmet. Moduler er altså fine for variable og prosedyrer som skal være tilgjengelige for *flere skjemaer* i et program. Biblioteker med klasser passer når man vil lage generelle klasser som kan brukes i mange programmer. Både biblioteker og moduler må importeres til de programmene der de skal brukes.

Om man deklarerer en klasse eller en modul, er derfor avhengig av hvordan og til hva den skal brukes. I eksemplet her, skal vi lage et *klassebibliotek*.

Klassen "Navn" i biblioteket "Navnebibliotek"

I dette eksemplet skal jeg lage et bibliotek *Navnebibliotek* med én klasse *Navn*. I UML-notasjon ser den slik ut:



Attributter

Vi gir klassen to egenskaper:

```
Public Class Navn
    Private fornavn As New List(Of String)
    Private etternavn As New List(Of String)
```

Legg merke til at både *fornavn* og *etternavn* er deklarert som *lister* – se eget notat om dem.

Tilgangsmetoder

Legg også merke til at attributtene er deklarert *Private* – det vil si at de ikke er synlige utenfor klassen. Det er tryggere, fordi det tvinger de programmererne som skal bruke dem, til å gå gjennom metoder som vi tilbyr. Slike metoder kalles da *tilgangsmetoder*. I tilgangsmetodene kan vi legge kontroller og formatteringer som da *alltid* vil bli utført når noen vil endre eller lese attributtverdien. Det bør vanligvis være én funksjon for lesing og én for skriving av attributtene, men her har vi flere fornavn og flere etternavn, og skal kunne legge til elementer[31]:

[31] Legg spesielt merke til at klassen *ikke* gir feilmelding i meldingsboks (den får ikke kommunisere med brukeren) men kaster feil. Feilen må applikasjonsprogrammet, som faktisk *har* kontakt med brukeren, håndtere.

```
Public Sub addFornavn(ByVal nyttFornavn As String)
    If nyttFornavn Is Nothing OrElse nyttFornavn = "" Then
        Throw New Exception("Fornavn må ha en verdi")
    Else
        fornavn.Add(Navn.normaliser(nyttFornavn))
    End If
End Sub

Public Sub addEtternavn(ByVal nyttEtternavn As String)
    If nyttEtternavn Is Nothing OrElse nyttEtternavn = "" Then
        Throw New Exception("Etternavn må ha en verdi")
    Else
        etternavn.Add(Navn.normaliser(nyttEtternavn))
    End If
End Sub
```

Add føyer til et nytt element bakerst i listen. Det burde også vært mulig å slette ett *fornavn* eller *etternavn*, men det skal vi se bort fra her, fordi koden blir nokså lik.

"Normalisering" av navnene

Alle navn skal "normaliseres", det vil si at de skal ha stor forbokstav og ellers bare små, uansett hvordan de ble skrevet av brukeren[32]:

```
Public Shared Function normaliser(ByVal navn As String) As String
    If navn Is Nothing OrElse navn = "" Then 'Merk bruken av OrElse istedenfor Or
        Return navn
    Else
        Return UCase(navn(0)) & LCase(navn.Substring(1))
        'Note: substring returnerer Empty hvis startindeksen=lengden
    End If
End Function
```

Denne er deklarert *Shared* og kan derfor brukes også av programmer som ikke har laget et navneobjekt. De kan henvise til klassenavnet *Navn* og skrive f.eks.

```
Dim etNavn As String = "Knut"
etNavn = Navnebibliotek.Navn.normaliser(etNavn)
```

Hvis de *importerer* navnerommet *Navnebibliotek* ved å skrive

```
Imports Navnebibliotek
```

øverst i programmet sitt, kan de også unngå å skrive *Navnebibliotek* hele tiden:

```
Dim etNavn As String = "Knut"
etNavn = Navn.normaliser(etNavn)
```

[32] Dette er ikke helt realistisk – navn som "von Hamburg", "McAllister" og andre, skrives annerledes. Biblioteket burde også tatt hensyn til det, f.eks. ved at brukeren kunne skrive en "\" foran for å indikere at navnet allerede er normalisert.

Hente fullt navn

For å hente hele navnet ut igjen, bruker vi her den vanlige funksjonen *ToString()* som alle objekter har, men vi skal overstyre den:

```
Public Overrides Function ToString() As String
    Dim fulltnavn As String = String.Empty
    For Each envn As String In etternavn
        fulltnavn &= " " & envn 'mellomrom foran hvert etternavn
    Next
    fulltnavn = fulltnavn.TrimStart() 'fjerner alle ekstra mellomrom forfra
    fulltnavn &= ", "
    For Each fnvn As String In fornavn
        fulltnavn &= fnvn & " " 'mellomrom bak hvert fornavn
    Next
    fulltnavn = fulltnavn.TrimEnd() 'fjerner alle mellomrom bakfra
    If fulltnavn = "," Then fulltnavn = "" 'har hverken fornavn eller etternavn
    Return fulltnavn
End Function
```

Slette hele navnet

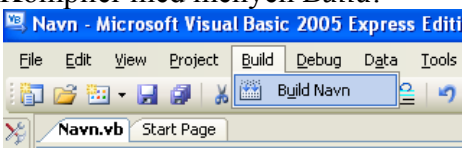
Vi kan også lage en metode som sletter alle navn:

```
Public Sub clear()
    fornavn.Clear()
    etternavn.Clear()
End Sub
```

Kompilere klassen til et klassebibliotek

En slik klasse kan ikke eksekveres alene, fordi alle Windowsprogrammer må ha en *Form* (men den behøver ikke være synlig). Den kan imidlertid *kompileres*. Det gjør det mye enklere å bruke den, og de som bruker den, kan ikke endre kildekoden – den er da ikke synlig.

Kompilér med menyen *Build*:



Det vil da lages en *dll-fil* som lagres i mappen *Bin/Release* (og i *Bin/Debug*) i prosjektmappen.

Dokumentere klassen

Vi har naturligvis – som vanlig – laget kommentarer i koden. Siden programmerere som bruker klassebiblioteket ikke får se koden, er det til liten hjelp for dem. Vi bør derfor alltid lage *dokumentasjon* som de andre programmererne får se. VB har en egen måte å lage slik dokumentasjon på (og det har også Java).

Slik dokumentasjon legges inn som kommentarer, på en spesiell form:

- ✓ Kommentarene må stå *rett over* en klasse eller medlem
- ✓ Kommentarene begynner med *tre apostrofer* (”enkeltfnutter”)
- ✓ Det brukes XML-kode (likner på HTML), med visse lovlige tagger
- ✓ XML-syntaksen kontrolleres og gir feil/advarsler mens de skrives inn

Med det samme du har skrevet tre apostrofer, viser programmeringsmiljøet en mal med de to vanligste taggene:

```
''' <summary>
'''
''' </summary>
''' <remarks></remarks>
```

Andre vanlige tagger er

```
''' <exception></exception>
''' <returns></returns>
```

Disse to er avhengige av at metoden kaster feil respektive er en funksjon som gir returverdi.

Det finnes mange andre tagger – se selv i VB Hjelp (søk etter *"XML Tags"*). Noen av dem kan også brukes til formattering av teksten mellom to andre tagger, f.eks. <para/> som gir linjeskift. Noen tagger har parametre – se eksemplene nedenfor.

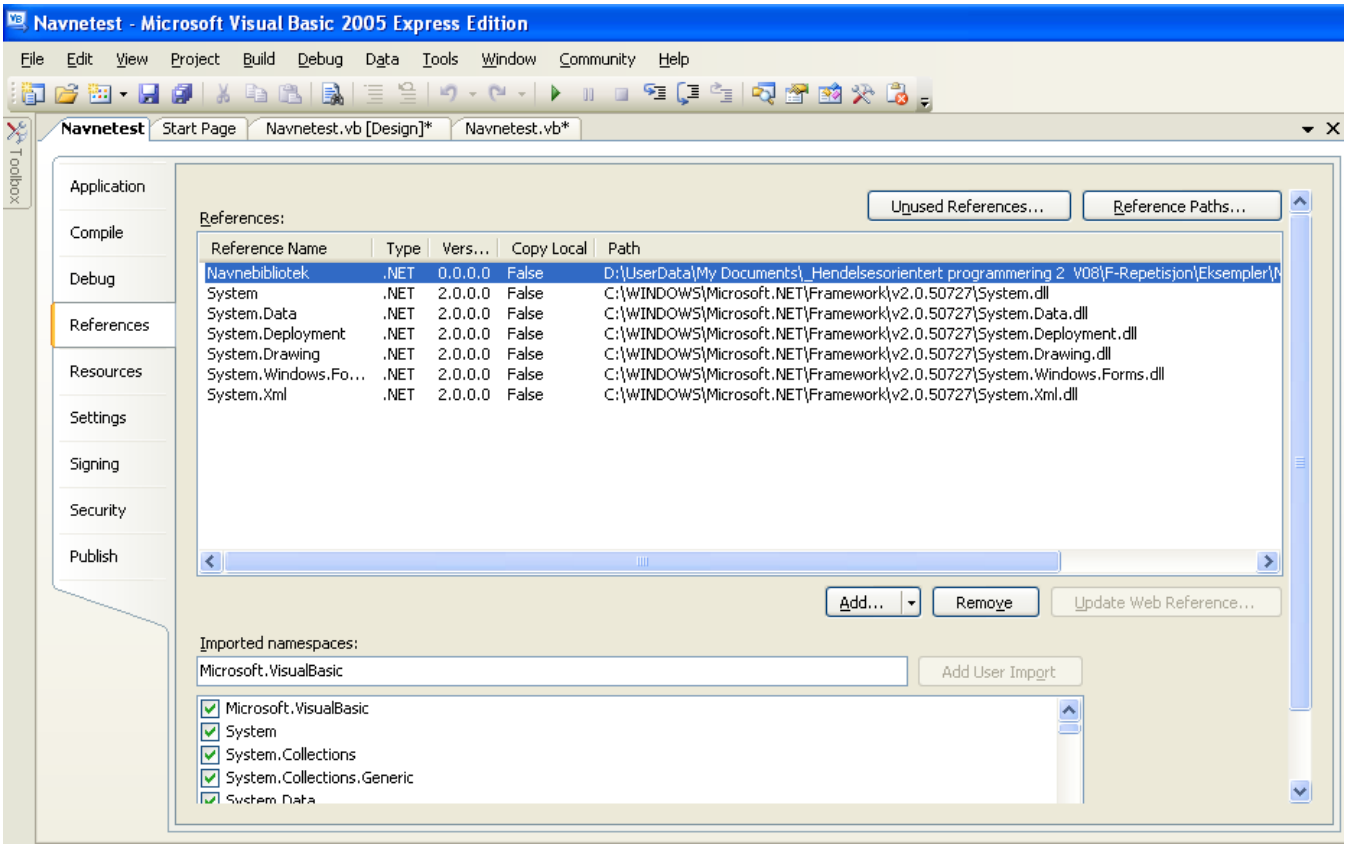
Her er klassebiblioteket ovenfor dokumentert slik:

```
''' <summary>
'''    Klassen Navn inneholder metoder for "normalisering" av navn
'''    <para/>OBS! Noen av metodene kaster feil!
''' </summary>
''' <remarks>
'''    Skrevet av: Knut W. Hansson
'''    <para/>Godkjent av: Ikke godkjent
''' </remarks>
Public Class Navn
    ''' <summary>En String-liste som kan ha fritt antall fornavn</summary>
    Private fornavn As New List(Of String)
    ''' <summary>En String-liste som kan ha fritt antall etternavn</summary>
    Private etternavn As New List(Of String)

    ''' <summary>Legger til et nytt fornavn</summary>
    ''' <param name="nyttFornavn">Et fornavn som skal legges til</param>
    ''' <exception cref="Exception">
    ''' Kaster feil hvis nyttFornavn er Nothing eller Empty
    ''' </exception>
    Public Sub addFornavn(ByVal nyttFornavn As String)
        If nyttFornavn Is Nothing OrElse nyttFornavn = "" Then
            Throw New Exception("Fornavn må ha en verdi")
        Else
            fornavn.Add(Navn.normaliser(nyttFornavn))
        End If
    End Sub

    ''' <summary>Legger til nytt etternavn</summary>
    ''' <param name="nyttEtternavn">Etternavnet som skal legges til</param>
    ''' <exception cref="Exception">
    ''' Kaster feil hvis nyttEtternavn er Nothing eller Empty
    ''' </exception>
    Public Sub addEtternavn(ByVal nyttEtternavn As String)
        If nyttEtternavn Is Nothing OrElse nyttEtternavn = "" Then
            Throw New Exception("Etternavn må ha en verdi")
        Else
            etternavn.Add(Navn.normaliser(nyttEtternavn))
        End If
    End Sub
```

```
    ''' <summary>
    ''' Returnerer hele navnet på "normalisert" form
    ''' dvs. på formen "Dyllan Hansen, Knut Erik"</summary>
    ''' <returns>Hele navnet på bedriftens "normaliserte" form.
    ''' Returner tom streng hvis både for- og etternavn mangler
    ''' </returns>
    Public Overrides Function ToString() As String
        Dim fulltnavn As String = String.Empty
        For Each envn As String In etternavn
            fulltnavn &= " " & envn 'mellomrom foran hvert etternavn
        Next
        fulltnavn = fulltnavn.TrimStart() 'fjerner alle ekstra mellomrom forfra
        fulltnavn &= ", "
        For Each fnvn As String In fornavn
            fulltnavn &= fnvn & " " 'mellomrom bak hvert fornavn
        Next
        fulltnavn = fulltnavn.TrimEnd() 'fjerner alle mellomrom bakfra
        If fulltnavn = "," Then fulltnavn = "" 'har hverken fornavn eller etternavn
        Return fulltnavn
    End Function

    ''' <summary>Sletter alle fornavn og alle etternavn</summary>
    Public Sub clear()
        fornavn.Clear()
        etternavn.Clear()
    End Sub

    ''' <summary>
    ''' Gjør om et fornavn eller et etternavn til "normalisert" form
    ''' </summary>
    ''' <param name="navn">Ett enkelt navn</param>
    ''' <returns>
    ''' Navnet på "normalisert" form, dvs. med store forbokstaver
    ''' men ellers bare små, for eksempel "Arne"
    ''' </returns>
    Public Shared Function normaliser(ByVal navn As String) As String
        If navn Is Nothing OrElse navn = "" Then
            Return navn
        Else
            Return UCase(navn(0)) & LCase(navn.Substring(1))
            'Note: substring returnerer Empty hvis startindeksen=lengden
        End If
    End Function
End Class
```

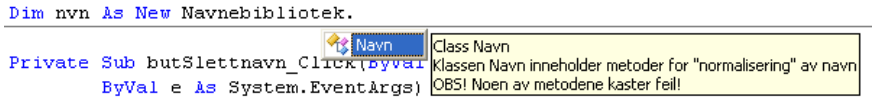
Bruk av et bibliotek

For å få *brukt* et slikt bibliotek, må man først knytte det til sitt eget program. Det gjør man enklest i *"My Project"* under *"References"*. Klikk knappen *"Add"* og finn frem til riktig dll-fil.



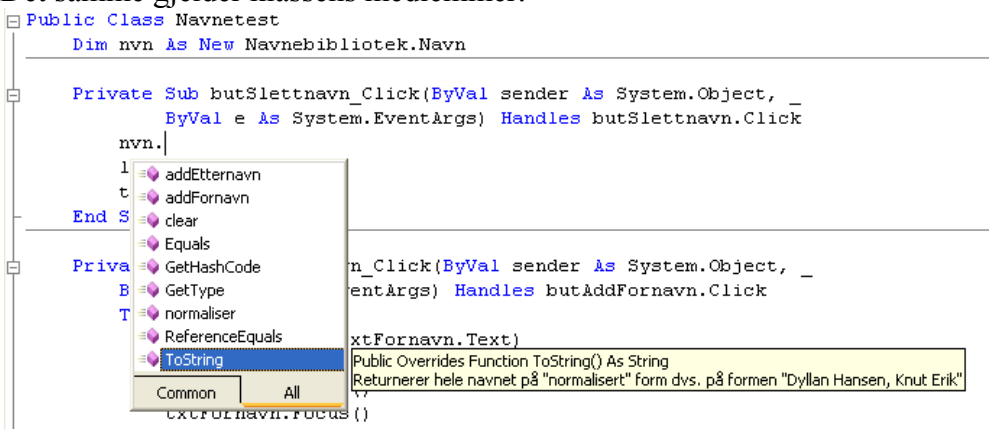
Etter at det er gjort, er navnerommet (her *"Navnebibliotek"*) blitt tilgjengelig og alt som ligger i det (her klassen *"Navn"*). Det innebærer to ting:

1) For det første vil programmeringsmiljøet kjenne igjen navnerommet, og komme med forslag (*intellisense*) på samme måte som om du brukte noe som Microsoft har programmert. Det kan se slik ut:

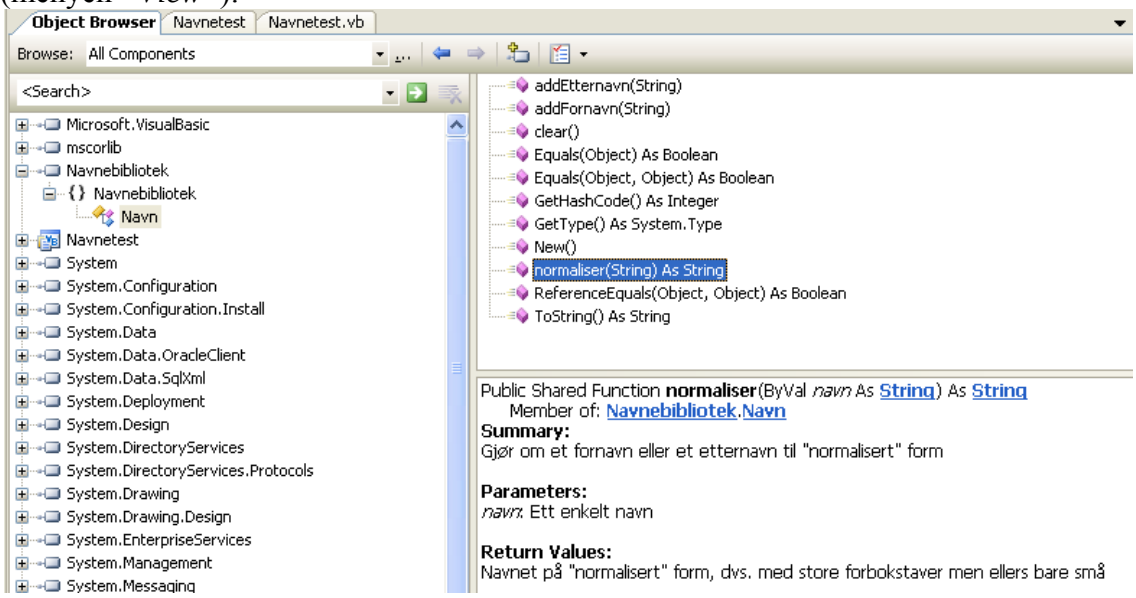


Legg merke til den gule ToolTip-teksten, som inneholder den teksten som er gitt i dokumentasjonen i taggen <*summary*>. Det er altså viktig at den skrives forståelig.

Det samme gjelder klassens medlemmer:



2) For det andre, vil *all* dokumentasjonen komme frem hvis man slår opp i *"Object Browser"* (menyen *"View"*):



Konklusjon

Det er enkelt å lage egne klasse- og programbibliotek. De kompileres ferdig (til *"Microsoft Intermediate Language"*, *"MSIL"* kode, som igjen kompileres til *Native Code* tilpasset den aktuelle prosessor), og kan da enkelt distribueres til utenforstående som hverken skal se kildekoden eller skal kunne endre den. De nødvendige filene vedlegges programmet hvis man "publiserer" det (*"Publish"* under menyen *"Build"*).

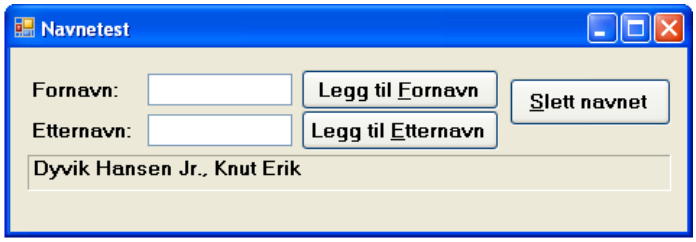
Det er spesielt viktig – siden man ofte ikke vil distribuere kildekoden – å dokumentere biblioteket godt.

Prøvekjøring av biblioteket

Biblioteket bør naturligvis kvalitetssikres nøye. Som et absolutt minimum må det *prøvekjøres*.

Da må man lage et nytt prosjekt, og legge inn referanse til biblioteket der. Så lager man et lite program, der man bruker alle deler av biblioteket. Samtidig legger man merke til hvilke ToolTips man får frem (det er også en del av kvalitetssikringen).

Her er grensesnittet for en slik "test bench":



Koden står på neste side. Jeg har her valgt å *la være* å importere navnerommet, så da må jeg skrive *Navnebilbliotek* når jeg skal bruke det.

Hvis prøvekjøringen viser feil, må biblioteket endres, og kompileres pånytt. Da kan du risikere at testprogrammet plutselig ikke lenger "kjenner" til biblioteket. Gå da inn i *My Project*|*References* og fjern referansen (*Remove*), og legg den så til igjen (*Add*).

```
Public Class Navnetest
    Dim nvn As New Navnebibliotek.Navn

    Private Sub butSlettnavn_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butSlettnavn.Click
        nvn.clear()
        visnavn()
        txtFornavn.Focus()
    End Sub

    Private Sub butAddFornavn_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAddFornavn.Click
        Try
            nvn.addFornavn(txtFornavn.Text)
            visnavn()
            txtFornavn.Clear()
            txtFornavn.Focus()
        Catch ex As Exception
            MsgBox(ex.Message)
            txtFornavn.Focus()
        End Try
    End Sub

    Private Sub butAddEtternavn_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butAddEtternavn.Click
        Try
            nvn.addEtternavn(txtEtternavn.Text)
            visnavn()
            txtEtternavn.Clear()
            txtEtternavn.Focus()
        Catch ex As Exception
            MsgBox(ex.Message)
            txtEtternavn.Focus()
        End Try
    End Sub
```

```
    Private Sub visnavn()
        lblNavn.Text = nvn.ToString
    End Sub

    Sub txtFornavn_KeyPress(ByVal sender As Object, _
             ByVal e As KeyPressEventArgs) Handles txtFornavn.KeyPress
        'Sjekker om tegnet er 'Enter' og kaller da butAddFornavn_Click
        If e.KeyChar = Chr(13) Then 'trykket enter
            e.Handled = True
            butAddFornavn_Click(sender, e)
        End If
    End Sub

    Sub txtEtternavn_KeyPress(ByVal sender As Object, _
         ByVal e As KeyPressEventArgs) Handles txtEtternavn.KeyPress
        'Sjekker om tegnet er 'Enter' og kaller da butAddEtternavn_Click
        If e.KeyChar = Chr(13) Then 'trykket enter
            e.Handled = True
            butAddEtternavn_Click(sender, e)
        End If
    End Sub
End Class
```

Her bør du spesielt merke deg *txtFornavn_KeyPress* og *txtEtternavn_KeyPress* som viser hvordan man kan sjekke input i et tekstfelt, tegn for tegn på to, litt forskjellige måter. *KeyPress* slår til med det samme tasten er trykket ned, og før den er sendt til tekstfeltet som input. Hvis tasten er Enter (= *Chr(13)* = *Keys.Enter*) så undertrykkes tegnet med *e.Handled* og den riktige hendelsesprosedyren kalles. Tilsvarende kan vi sikre at brukeren bare taster siffer. Lovlig er da *Keys.D0…Keys.D9*, *Keys.NumPad0…Keys.NumPad9*, *Keys.Back* og *Keys.Delete*.

Tema F2 - Litt om "lenkede lister"

Definisjon

National Institute of Standards and Technology (NIST) definerer en liste slik[33]:

> **Definition:** A *list* (a collection of items accessible one after another beginning at the *head* and ending at the *tail*) implemented by each item having a *link* to the next item.
> **Also known as** *singly linked list*.
> **Specialization** (... is a kind of *Me* …) *doubly linked list*, *ordered linked list*, *circular list*.
> **Generalization** (I am a kind of ...) *bag*.

Liste i VB.NET

Lenkede lister er en datastruktur som er mye brukt, fordi den har en del fordeler. Vanligvis må vi lage slike lister selv, og selv lage metoder som manipulerer dem. Her er det lett å gå i vannet!

Heldigvis har Microsoft laget en ferdig definisjon for oss, som heter *List (Of T)* der *T* er nodenes datatype/klasse. Det er vanskelig å vite nøyaktig hvordan Microsoft har laget sin versjon av dette, men utfra definisjonen kan vi *tenke* oss at det er gjort som forklart her[34]. Hvis du vil bruke VB Hjelp, søker du etter "*List (T)*".

Jeg har laget et eksempel (*TestListe*) som brukes nedenfor.

Deklarere en liste (Dim, Public, Private osv.)

Vi deklarerer en variabel som skal referere til en liste med strenger:

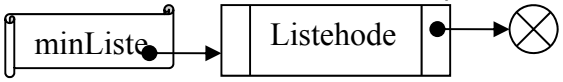
```
Dim minListe As List(Of String)
```

Deklarerer en variabel som skal kunne referere til en liste. Forløpig peker den ikke til noe:

Skape listen (New)

Deretter skaper vi listen, og setter variabelen til å peke til den:

```
minListe = New List(Of String)
```



Listehodet er et *objekt* som har diverse medlemmer til manipulering av listen, og en peker til første streng i listen. I utgangspunktet peker den til *Nothing*. Listehodet kan bare peke til strenger, fordi den er deklarert som *List (Of String)*.

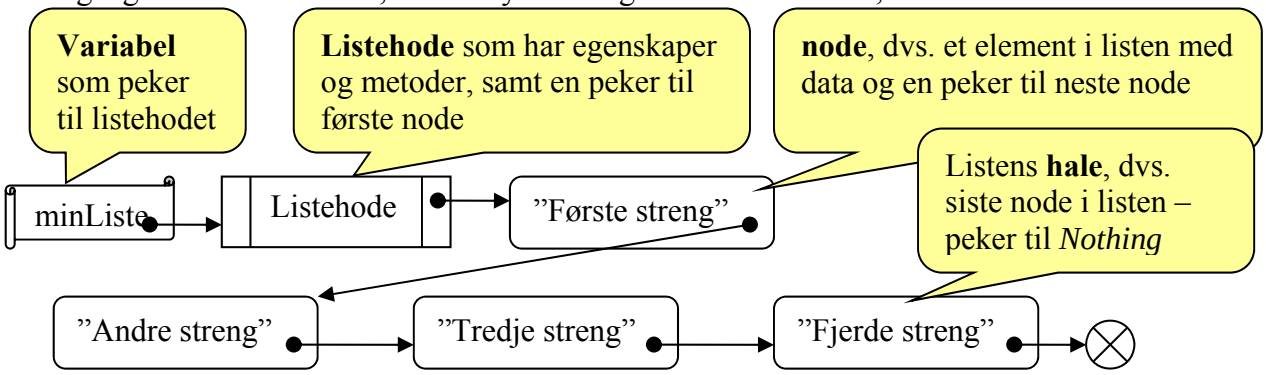
Legg til noder bakerst i listen (Add)

Vi legger nå fire strenger til listen med metoden *Add* som listehodet tilbyr:

```
minListe.Add("Første streng")
minListe.Add("Andre streng")
minListe.Add("Tredje streng")
minListe.Add("Fjerde streng")
```

[33] http://www.nist.gov/dads/HTML/linkedList.html
[34] I realiteten er *list* implementert som en skjult *array* som utvides og pakkes ved behov.

Hver gang det foretas en *Add*, blir en ny node lagt til *bakerst* i listen, så nå ser listen slik ut:

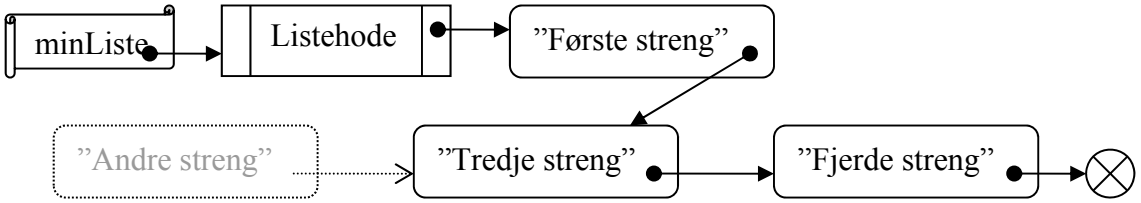


Slette en node med kjent indeks (RemoveAt)

For å finne en bestemt node i en slik liste, skal vi prinsipielt alltid lete sekvensielt fra listehodet. Microsofts *list(Of T)* har imidlertid forenklet dette, så vi kan henvise til en bestemt node med indeks (akkurat som en array). Vi kan derfor slette den andre noden slik (listen er nullbasert, så den første noden har nummer 0):

```
minListe.RemoveAt(1)
```

Da ser listen slik ut:

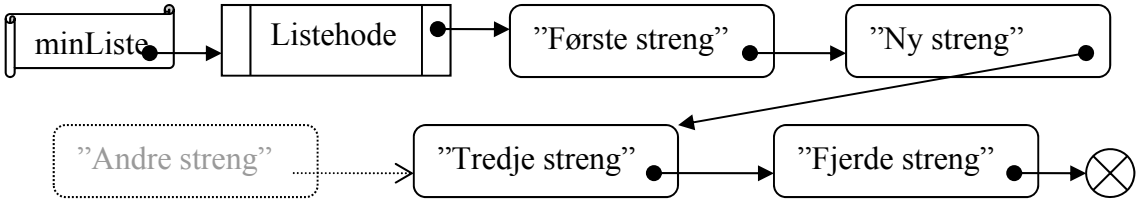


Å slette en node (når den først er funnet), innebærer helt enkelt å sette noden foran til å peke til noden etter, og så fristille RAM der den slettede noden var lagret. En ”garbage collector” finner objekter som ingen lenger peker til, slik som ”Andre streng” i eksemplet. Den plassen i RAM som disse opptar, blir da fristilt. Det er ”garbage collector” som selv bestemmer når det skal skje[35].

Legge til en node inne i listen (Insert)

Vi legger nå til en node mellom første og andre:

```
minListe.Insert(1, "Ny streng")
```



Den nye strengen skapes i RAM, og settes til å peke til samme node som strengen foran. Deretter settes strengen foran til å peke på den nye.

Array eller liste

Konklusjonen er at det er svært enkelt og raskt å sette inn nye noder, og å slette en node. Det er derimot tungt å finne en bestemt node i listen, fordi vi må lete sekvensielt fra listehodet. Det er ikke mulig for kompilatoren å beregne seg frem til hvor node *n* er lagret, fordi nodene skapes én og én, der det er plass.

[35] Garbage Collector heter GC. Den startes automatisk med jevne mellomrom, og alltid når RAM er full og programmet ber om mer plass. Den kan tvinges til å starte med kommandoen GC.Collect(). Ledig minne oppgis med GC.GetTotalMemory(False).

En array gir rask tilgang til et bestemt element *n*, men har følgende ulemper i forhold til lister:

- ✓ Den har fast, begrenset størrelse (kan utvides, men det er tungt)
- ✓ Når et element slettes, blir det liggende igjen på samme plass – en array er derfor ikke nødvendigvis "pakket" (den kan pakkes, men det er tungt)

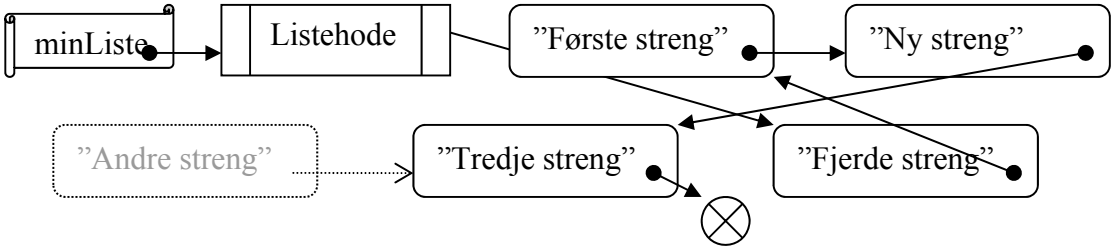
Lister har derimot (nesten) ingen begrensning, og er alltid "pakket".

Sortere en liste

Hvis vi f.eks. vil sortere *minListe* skriver vi bare:

```
minListe.Sort()
```

Da vil pekerne endres, så nodene står i sortert rekkefølge når vi følger pekerne fra listehodet:



Gå igjennom alle nodene

Hvis vil gå igjennom alle nodene, bruker vi *For Each*. Her skriver vi alle nodene i en tekstboks:

```
'Skriv ut alle nodene
For Each s As String In minListe
    txtListe.Text &= s & vbNewLine
Next
```

Objektet bak *For Each* – her *s* – må være av samme klasse (type) som nodene i listen.

Det er jo også mulig å bruke indeks og *Count*:

```
'Alternativ (tyngre) utskrift
For i As Integer = 0 To minListe.Count - 1
    txtListe.Text &= minListe(i) & vbNewLine
Next
```

Listeobjektenes medlemmer

En *List(Of T)* har mange medlemmer – her er noen eksempler:

.*Count* viser hvor mange noder det er i listen
.*Add (Item As T)* legger *Item* til på slutten av listen.
.*Insert (index As Integer, item as T)* setter *item* inn i listen på plassen angitt av *index*.
.*Addrange (collection As List (Of T))* legger til hele den andre listen *collection* bakerst (enklere enn *for-next* eller *For Each*)
.*Clear ()* sletter hele listen (men ikke listehodet)
.*RemoveAt (index As Integer)* fjerner den noden som er angitt av indeksen
.*Sort ()* sorterer listen (sammenlikningsoperatorene må være gyldige for objektene i listen)

Det finnes også mange metoder for å søke i listen.

Andre typer av lister

Det finnes mange varianter av lister[36]. Noen har peker begge veier mellom nodene (både til "neste" og til "forrige") så man kan lete sekvensielt i begge retninger. Andre har bare en ekstra peker fra listehodet til halen (da er det meget raskt å legge til en ny node bakerst). Noen bygger opp en indeks for nodene (da blir det meget raskt å finne en bestemt node).

[36] Du kan lese mer om lister på http://en.wikipedia.org/wiki/Linked_list

Tema G1 – Dynamisk grafikk

Dynamisk grafikk dreier seg om grafikk som utføres mens programmet kjøres. Det som står her, gjelder skjema og andre kontroller.

Grafikk som tegnes mens kontrollen er dekket av noe annet, vises ikke. Grafikken vil også forsvinne hvis vinduet dekkes av et annet vindu, eller minimeres. Det er *ikke* enkelt å løse dette, og vi skal se bort fra det her.

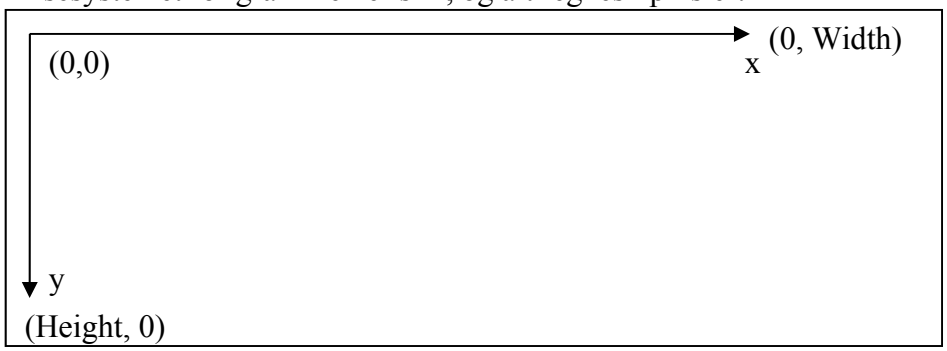
Klargjøring

Før vi kan bruke grafikk, må vi gjøre hele skjemaet eller bare en kontroll grafisk. Her er *picGraf* en *PictureBox*, mens *grafskjema* er et skjema:

```
Public bildekontroll As Graphics = picGraf.CreateGraphics() 'gjøres grafisk
Public grafskjema As Graphics = Me.CreateGraphics() 'skjemaet gjøres grafisk
```

Deretter henviser vi til objektet når vi tegner.

Aksesystemet for grafikken er slik, og alt regnes i piksler:



Tegning, dvs. å trekke streker, krever en ***Pen*** med farge og mange andre egenskaper. Videre angir vi x- og y-ordinater for det øvre, venstre hjørne for den firkanten ellipsen skal tegnes inne i, deretter bredde og høyde. For å plotte et punkt på én piksel, bruker vi *DrawEllipse()* inne i et lite rektangel (her bare 1 piksel bredt og 1 piksel høyt).

```
'Plot et lite punkt øverst til venstre:
bildekontroll.DrawEllipse(Pens.Black, 0, 0, 1, 1)
'Plot et lite punkt øverst til høyre:
bildekontroll.DrawEllipse(Pens.Black, picGraf.Width - 1, 0, 1, 1)
'Plot en sirkel i midten med litt bred penn
Dim rødblyant As New Pen(Color.Red, 4) 'farge, bredde
bildekontroll.DrawEllipse(rødblyant, picGraf.Width \ 2, _
    picGraf.Height \ 2, 15, 15)
```

Legg spesielt merke til at vi ikke kan tegne en slik ellipse fra ordinatene x = *picGraf.Width* og y = *picGraf.Height* fordi den da vil tegnes *utenfor* tegneområdet.

For å **male**, dvs. plotte et litt større punkt, bruker vi en ***Brush*** og *FillEllipse()* som fyller ellipsen med farge.

```
'Plot et større punkt (fem piksler) nederst til venstre:
bildekontroll.FillEllipse(Brushes.DarkBlue, 0, picGraf.Height - 5, 5, 5)
'Plot et større punkt (fem piksler) nederst til høyre:
bildekontroll.FillEllipse(Brushes.DarkBlue, picGraf.Width - 5, _
    picGraf.Height - 5, 5, 5)
```

Igjen må vi selv passe på – ved å trekke fra størrelsen av ellipsen, at ikke grafikken går utenfor tegneområdet.

Generelt tegnes ellipser alltid inne i et rektangel, og hvis rektangelet er et kvadrat, blir ellipsen en sirkel.

For å tegne rette linjer, bruker vi *DrawLine()* og en *Pen*:

```
'Tegn en rett linje vannrett midt på grafikken (y-akse)
bildekontroll.DrawLine(Pens.DarkGreen, 0, picGraf.Height \ 2, _
    picGraf.Width, picGraf.Height \ 2)
'Tegn en rett linje loddrett midt på grafikken (x-akse):
bildekontroll.DrawLine(Pens.DarkGreen, picGraf.Width \ 2, 0, _
    picGraf.Width \ 2, picGraf.Height)
```

For å slette grafikken, bruker vi *Clear* men da må vi også oppgi bakgrunnsfargen:

```
'Fjern all grafikk
bildekontroll.Clear(Color.White)
```

Pen og Brush

Det finnes mange figurer som er ferdige og mange av dem kommer i to varianter: *Draw* og *Fill*. Det brukes en *Pen* til å tegne og en *Brush* til å male, dvs. fylle figuren med farge. I eksemplene ovenfor brukte jeg ferdig redskap (med farge på) som plukkes fra "en hylle i skjulet", men hvis vi vil sette noen egenskaper på dem, må vi lage redskapen selv:

```
Dim minPenn As Pen = New Pen(Color.Blue) 'skaper en penn
minPenn.DashStyle = Drawing2D.DashStyle.Dot 'skal tegn prikkede linjer
minPenn.Width = 5 'fem piksler bred
minPenn.EndCap = Drawing2D.LineCap.Triangle 'skal ha trekant i hver ende
minPenn.LineJoin = Drawing2D.LineJoin.Round 'hjørner skal avrundes
'Tegn linje 1 med koordinater for hver ende
bildekontroll.DrawLine(minPenn, 50, 50, 100, 200)
'Tegn rektangel
minPenn.Color = Color.DarkRed  'skift farge
bildekontroll.DrawRectangle(minPenn, 200, 200, 300, 200)
'Tegn linje 2 med punkt i hver ende
'a) Skaper punkter
Dim punkt1 As Point = New Point(400, 200) 'skaper et punkt
Dim punkt2 As Point = New Point(500, 400) 'skaper et punkt til
punkt1.X = 15 'endrer x-ordinat
'b) Tegner
minPenn.Color = Color.DarkGreen
bildekontroll.DrawLine(minPenn, punkt1, punkt2) 'tegner en linje
```

Det er lite aktuelt å lage pensler, fordi de har så få egenskaper vi kan endre. Bruk heller standardpenslene *Brushes.White* osv.

Det går naturligvis an å lage sine egne farger utfra ARGB-koden:

```
Dim pensel As SolidBrush = New SolidBrush(Color.Black)
pensel.Color = Color.FromArgb(A, R, G, B) 'gi penselen farge
```

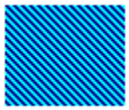
Verdiene A, R, G og B er Integer i intervallet [0..255]. A angir i hvilken grad penselen (fargen) skal være transparent, der 0=Usynlig og 255=heldekkende. De andre angir mengden av rødt, grønt og blått.

Det finnes imidlertid så mange standardfarger er det er lite aktuelt å lage egne farger. Eller du kan la brukeren velge farge selv, med fargepaletten (en kontroll i verktøykassens *Dialogs*):

```
Dim minPenn As Pen = New Pen(Color.Blue) 'default farge
If MyColorDialog.ShowDialog() = DialogResult.OK Then 'vis dialogen
    minPenn.Color = MyColorDialog.Color 'hent valgt farge
End If
```

Du kan også (i navnerommet System.Drawing.Drawing2D) skape *HatchBrush* pensler med mønstre. Denne koden gir f.eks. nedenstående resultat med tett, skrå linjer:

```
Dim spesialpensel As Drawing2D.HatchBrush = _
      New Drawing2D.HatchBrush(Drawing2D.HatchStyle.DarkDownwardDiagonal, _
      Color.Cyan, Color.DarkBlue)
bildekontroll.FillRectangle(spesialpensel, 20, 30, 60, 50)
```



Punkter og firkanter

Det er ofte bruk for å angir *punkter* med x- og y-ordinat. Det finnes en punktklasse *Point* som inneholder både x og y:

```
Dim punkt1 As Point = New Point(400, 200) 'skaper et punkt
Dim punkt2 As Point = New Point(500, 400) 'skaper et punkt til
punkt1.X = 15 'endrer x-ordinat
bildekontroll.DrawLine(minPenn, punkt1, punkt2) 'tegner en linje
```

Du kan også skape *firkanter*:

```
Dim firkant As Rectangle = New Rectangle(250, 250, 100, 200)
bildekontroll.DrawRectangle(minPenn, firkant)
bildekontroll.FillEllipse(Brushes.MistyRose, firkant)
```

(Ellipsen blir tegnet inne i firkanten.)

Mer om aksesystemet

Når vi skal grafe en funksjon, er det svært tungvint at y-en går ovenfra og ned. Vi vil også gjerne ha nullpunktet (origo) et sted ute i det grafiske vinduet og ikke helt opp til venstre.

For å markere origo må vi da regne om og plotte punktet (Width\2, Height\2). For å markere punktet (x, y) = (500, 1000) må vi tilsvarende regne om og plotte (Width\2+500, Height\2–1000). Merk at vi må *trekke fra* for å få riktig y! Selvom vi regner ut origo i variabler X0 og Y0 og bruker dem isteden for *Width* og *Height*, må vi stadig tenke for å få dette riktig.

Når vi skal plotte en funksjon i en graf med akser, blir dette svært tungvint og fører til mye omregning. Hvis vi f.eks. vil ha en skala der hver x og y er én cm på aksene, og vil plotte punktet (4,-6) må skrive noe slikt: Width\2+4*567, Height\2-(-6)*567.

Dette er litt vrient å holde styr på. Vi kan ønske oss en klasse som kan omregne på denne måten. Den har jeg laget, og den kan du bare legge til ditt program. Den ligger i en modul som heter *ModGrafikk*. Der finner du klassen *GrafKlasse* som tilbyr følgende metoder:

```
Public Sub New( _
    ByVal kontroll As Control, _
    ByVal minX As Double, ByVal minY As Double, _
    ByVal maxX As Double, ByVal maxY As Double)
```

New skaper et nytt GrafKlasse-objekt og setter variable. Du må oppgi kontrollen du har tenkt å tegne på. Du må også angi minste x- og y-verdi du har tenkt å plotte, samt største x- og y-verdi som det er bruk for. *New* vil sørge for at aksene kommer med i grafikken, selv om f.eks. både minste og største y-verdi er positive. Den vil også sørge for at hele grafen kommer med (den er avhengig av riktige min- og maks-verdier).

```
Public Function Xplot(ByVal Xverdi As Double) As Integer
```

Xplot returnerer den x (piksler fra venstre kant) du bør bruke i de grafiske metodene for å plotte en bestemt x-verdi.

```
Public Function Yplot(ByVal Yverdi As Double) As Integer
```
Yplot beregner tilsvarende for en y-verdi.

```
Public Sub tegnakser(ByVal bildekontroll As Graphics, ByVal penn As Pen)
```
Tegnakser tegner aksene på passe sted på den oppgitte grafiske flaten, slik at aksenen passer til minste og største verdier for x og y.

Her er et eksempel på bruk, for å plotte grafen y = 2*x -1 for x = -2 til 4 (da blir y = -5 til 7):

```
Private Sub butGraf_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butGraf.Click
    Dim bildekontroll As Graphics = picGraf.CreateGraphics()
    Dim graf As GrafKlasse = New GrafKlasse(picGraf, -2, -5, 4, 7)
    graf.tegnakser(bildekontroll, Pens.Violet)
    Dim Xpunkt As Integer
    Dim Ypunkt As Integer
    For x As Double = -2 To 4 Step 0.01
        Xpunkt = graf.Xplot(x)
        Ypunkt = graf.Yplot(2 * x - 1)
        bildekontroll.FillEllipse(Brushes.Blue, Xpunkt, Ypunkt, 3, 3)
    Next
    bildekontroll.Dispose() 'anbefales av MS for å hindre minnelekkasje
End Sub
```

Hvis du har bruk for andre ting, kan du jo bare legge til dine egne variable og metoder.

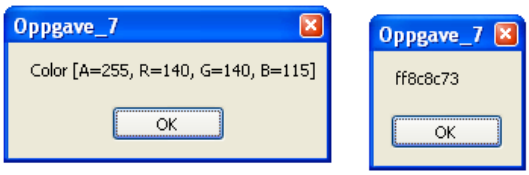
Tegne et bilde og finne farge på en piksel i bildet

```
Dim bilde As New Bitmap("C:\Windows\Greenstone.bmp")
bildekontroll.DrawImage(bilde, 20, 30)
```

Med metoden *DrawImage* kan du tegne et bilde, eller en del av et bilde, fra fil og direkte på skjemaet. Parametrene angir øvre venstre hjørne av bildet. Bildet må være en bitmap (ikke vektorgrafikk), og det er de fleste bildeformater.

Når du har en bitmap, kan du finne fargen i hvert punkt med *GetPixel(x,y)* som returnerer et *Color*-objekt:

```
Dim fargepunkt As Color = bilde.GetPixel(4, 5)
MsgBox(fargepunkt.ToString)
MsgBox(fargepunkt.Name)
```



Skrive på den grafiske flaten

Når du skriver på en grafisk flate, må du også ”tegne” bokstavene. Du må oppgi både font, pensel og hvor du vil ha teksten (øvre, venstre hjørne):

```
Dim minFont As New Font("Arial", 24, FontStyle.Bold)
bildekontroll.DrawString("Hello World!", minFont, Brushes.Cyan, 10, 10)
```

Det kan være aktuelt, f.eks. når tekst skal midtstilles, å vite lengden og høyden av den:

```
Dim høyde As Single = _
    bildekontroll.MeasureString("Hello World!", minFont).Height
```

Du gjør tilsvarende for bredden. (Egentlig returnerer *MeasureString* et objekt av typen *SizeF* som har både bredde og høyde.)

Tema G2 – Farger

Det er stor forskjell på farger fra lyskilder – som f.eks. PC-skjermer – og farger fra gjenstander som reflekterer lys – f.eks. et bilde malt med vannfarger. I lyskilder, *legges fargene sammen* og vi snakker om *additive farger*. Bildet, derimot, mottar lyset fra en annen lyskilde, absorberer noe av det og reflekterer resten. Det *trekkes altså fargene ifra* og vi snakker om *subtraktive* farger[37].

Anta f.eks. at vi angir farger med rødt, grønt og blått og bruker én bit til hver farge[38]. Da kan vi angi åtte forskjellige farger med kode 000-111. Når de blandes additivt, legges fargene sammen med en **OR-operasjon**, f.eks. slik:

	R	G	B
Fiolett lyskilde	1	0	1
Cyan lyskilde	0	1	1
Resultat med OR	**1**	**1**	**1**

"Bitvis OR" innebærer at vi ser på bit for bit – her kolonne for kolonne. Hvis minst én av dem er 1, blir resultatet 1 – ellers 0.

Anta at vi har de samme to fargene, men blander dem subtraktivt. Det tilsvarer å sette et filter bak den første fargen, som bare slipper noe igjennom. Vi må da bruke **AND-operasjon**:

	R	G	B
Fiolett lyskilde	1	0	1
Cyan filter	0	1	1
Resultat med AND	**0**	**0**	**1**

"Bitvis AND" innebærer at bare hvis begge bits er 1, blir resultatet 1 – ellers 0.

Programmet *Farger* demonstrerer disse to måtene å blande farger på.

Når vi bruker datamaskin, vil fargene på skjermen virke additivt. På en skriver, blander vi blekk, pulver eller annet med hverandre, og de virker subtraktivt. Det er derfor behov for forskjellige modeller for forskjellig behov, og det er ikke enkelt å få en utskrift i farger til å se ut som den gjør på skjermen. I tillegg vil opplevelsen "farges"☺ av den som ser: Fargen er svak, intens, pen, kvalm, stygg, kul, jentete, maskulin, varm, kald, opphissende, beroligende…

Det er tre, tekniske fargesystemer: **RGB**, **HSL** og **CMYK**. *RGB* er basert på utsendt lys (additiv modell) og er systemet som brukes i TV og monitorer. *HSL* er basert på hvordan mennesker oppfatter fargene. *CMYK* er tilpasset skrivere med subtraktive farger. Nedenfor har jeg hentet "hjelp" fra Jasc PaintShop Pro versjon 8 som forklarer dette rimelig enkelt.

Understanding Color and Color Models

We usually think of color as a quality inherent in an object—a red car or a green frog. But color is really what we see as a result of three factors interacting: light, the object, and the observer. As rays of light hit the object, the object absorbs some light and reflects some light. We see the reflected light and perceive it as color. Different colors reflect light of different wavelengths. Human eyes are able to perceive thousands of colors in the visible spectrum of light.

[37] Det finnes fargestoffer som omdanner usynlig lys til synlig, og som derved kan reflektere mer synlig lys enn de mottar. De brukes i redningsvester, vernevester, merkebøyer osv. Den første som kom, ble kalt "Day-Glo" – en ekkel oransje farge "etter mitt syn". Vaskemiddelfabrikantene tilsetter noe for at det hvite skal se *enda* hvitere ut – det omdanner visstnok ultrafiolett til synlig lys. Derfor kan et lommetørkle på snøen se hvitere ut enn snøen det ligger på.
[38] I en tidlig versjon av Microsoft Visual Basic, var det faktisk bare åtte farger som ble kodet på denne måten.

When you apply ink to paper, the colors we see result from the light that the ink reflects. Computer monitors use emitted light rather than reflected light—the colors we see result from light emitted from the screen.

To describe how color is produced or perceived, we use color models. Computer monitors display colors by producing varying amounts of red, green, and blue light—the RGB color model. Human eyes perceive color by its hue, saturation, and lightness levels—the HSL color model. With Paint Shop Pro you can select colors using either the RGB or HSL color model. You can also output images using the CMYK (Cyan, Magenta, Yellow, Black) model, which is used for high-end printing applications.

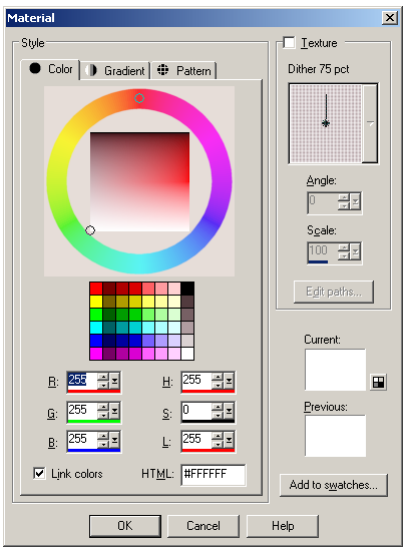
RGB Model (Red, Green, Blue)

All colors on your computer screen are created by mixing red, green, and blue light in varying proportions and intensities. When these primary colors are mixed in equal proportions, they create yellow, cyan, and magenta. Adding all the colors together creates white.

Each primary color (red, green, and blue) is assigned a value from 0 (none of the color present) to 255 (the color at full strength). For example, pure red is produced by combining a red value of 255, a green value of 0, and a blue value of 0. Yellow is a combination of a red value of 255, a green value of 255, and a blue value of 0. *Setting all three values to 255 produces white; setting all three values to 0 produces black. When all three colors are set to the same value (such as 120, 120, 120), the result is grey.*

HSL Model (Hue = Farge, Saturation = Fargemetning, Lightness = Lysintensitet)

The HSL model is based on how the human eye perceives color using the characteristics of hue, saturation, and lightness. Each characteristic is assigned a value from 0 to 255. The three characteristics are described as follows:



Hue: The color reflected from an object, such as red, yellow, or orange. Each hue value is assigned based on its position on the color wheel. On the Jasc Color Picker's Color wheel, colors are assigned counter-clockwise from the top. Red is at the top (value 0) and as you move around the wheel the colors go through orange, yellow, green, blue, purple[39], and back to red.

Saturation: The purity or vividness of the color. Saturation represents the amount of grey in the color, from 0 (entirely grey) to 255 (fully saturated color).

Lightness: The perceived amount or intensity of light in the color. Lightness ranges from 0 (no light, or black) to 255 (total lightness, or white). At 50 percent lightness, or a value of 128, a color is considered pure. For example, pure red has a hue of 255, a saturation of 255 (100 percent) and a lightness of 128 (50 percent). For pure blue, the hue is 170, saturation is 255 and lightness is 128.

[39] Tilsvarer rekkefølgen i spektrumet til hvitt lys.

CMYK Model
(Cyan=blågrønn, Magenta=rødblå, Yellow, black)

The CMYK model is based on the fact that ink on paper both absorbs and reflects light. As white light strikes the ink, part of the color spectrum is absorbed and part is reflected back to your eyes (resulting in the color you see).

In this model, the primary colors cyan (C), magenta (M), and yellow (Y) combine in varying proportions to produce a variety of colors. When the three colors are combined, they produce black. Because impurities in the ink make it difficult to produce a true black, a fourth color, black (K), is added.

Combining inks in this way is called four-color process printing. It is used by printing services and high-end color printers.

Although you cannot create images in Paint Shop Pro using the CMYK model, you can produce color separations that can be printed on CMYK printers. There are two ways to do this: You can split the images into CMYK channels or you can print color separation pages.

CMYK channels are simply four separate greyscale images that represent the percentage and location of cyan, magenta, yellow, and black in the image.

"Web Safe" farger – ikke lenger aktuelt

Datamaskiner gjenga tidligere ikke farger likt, og de gjorde det forskjellig på Macintosh og PC. Derfor kunne en webside, som så OK ut når du så på den med MSIE på en PC, se helt annerledes ut i Safari på en Macintosh.

Det var vanlig å anta at disse miljøene hadde 216 felles farger, med fargekode med to og to like siffer, der alle sifrene er delelig med tre, f.eks.: $3366FF_H$, 996633_H, $CC00CC_H$. Derfor burde du tidligere angi slike farger på websider.

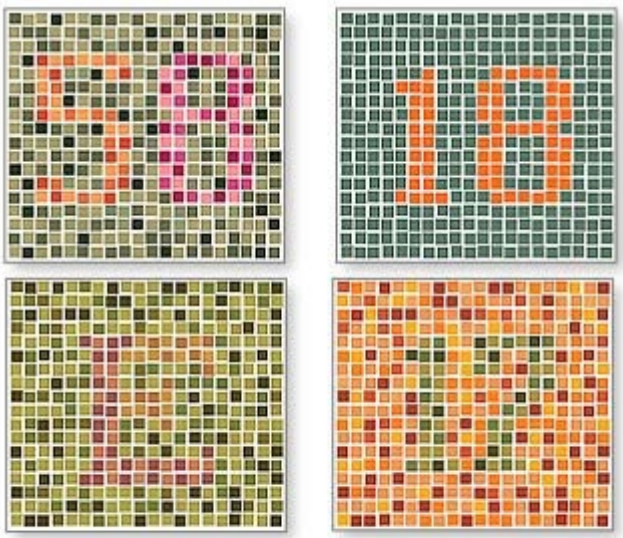
I dag kan de aller fleste maskiner gjengi millioner av farger, og dette er blitt lite aktuelt. Hvis du vil være helt sikker, kan du finne web safe fargene hos
http://www.w3schools.com/html/html_colors.asp.

Synet vårt

Øyet har **tapper** som oppfatter lyset og bølgelengden (og dermed *lysfargen*). De er avhengig av relativt godt lys. I tillegg har øyet **staver** som bare oppfatter lysintensiteten og dermed bare ser *gråtoner*. De er mer lysømfintlige og i svakt lys ser vi derfor bare med stavene og alt blir gråtoner ("i mørket er alle katter grå").

Omtrent 4% av befolkningen har feil i fargesynet. Du kan se hvordan det arter seg på
http://webexhibits.org/causesofcolor/2D.html,
eller ta en test selv på http://colorvisiontesting.com/online%20test.htm

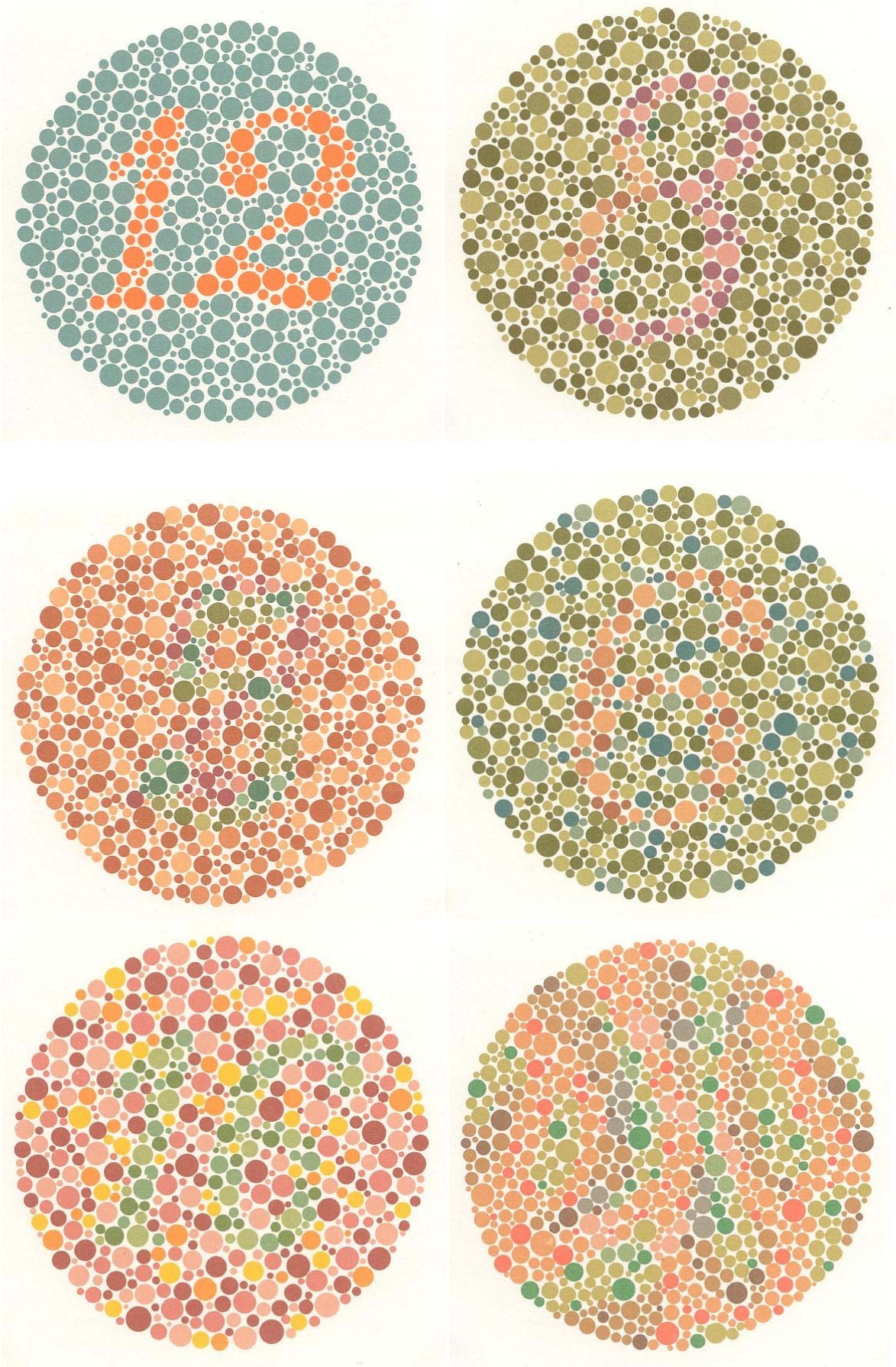
Disse bildene er fra http://www.nlm.nih.gov/medlineplus/ency/imagepages/9962.htm:



Various tests for color blindness

ADAM.

På neste side er bilder fra http://www.doktoren.no/fargesyn-test.htm.



Tema H1 – Lyd og film i Visual Basic

Husk at både lyd og video bruker filer, og derfor kan mye galt skje. Du bør derfor bruke feilfeller (*Try-Catch*).

Bakgrunnslyd

Det er veldig enkelt å spille bakgrunnslyd av typen *wav*. Du bruker *My.Computer.Audio.Play*:

```
Dim fil As String = My.Application.Info.DirectoryPath & "C:\minlyd.wav"
My.Computer.Audio.Play(fil, AudioPlayMode.BackgroundLoop)
```

Som du ser, må du oppgi en wav-fil som skal spilles (full sti) og kan angi en spillemodus som er enten

- ✓ *AudioPlayMode.BackGround* som er default og spiller wav-filen én gang asynkront (dvs. programmet fortsetter)
- ✓ *AudioPlayMode.BackgroundLoop* som spiller wav-filen om og om igjen asynkront
- ✓ *AudioPlayMode.WaitToComplete* som spiller wav-filen én gang synkront (dvs. programmet venter til filen er ferdigspilt).

Som vanlig, kan vi jo hente filnavnet fra brukeren med en *OpenFileDialog*.

For å stoppe lyden, skriver du

```
My.Computer.Audio.Stop()
```

Du kan også spille av systemlyder (hentet fra Windows-oppsettet):

```
My.Computer.Audio.PlaySystemSound(Media.SystemSounds.Exclamation)
```

Multimedia

Hvis du vil vise brukeren multimedia på vanlig måte, anbefaler jeg å bruke den vanlige Windows Media Player. Du kan starte den (”shelle”) og oppgi filnavnet som skal spilles, slik:

```
Dim id As Integer
Dim programsti As String
'1) Hva skal spilles?
Åpnefil.Filter = "Alle filer (*.*)|*.*")
Åpnefil.ShowDialog()
'2) sett programsti med parametre ("switches")
'For å kalle på programmet, må full sti oppgis
'og parameteret (-p for play) må ha et filnavn i anførselstegn
programsti = "C:\Program Files\Windows Media Player\wmplayer.exe" _
    & " -p """ & Åpnefil.FileName & """"
MsgBox(programsti) 'vis programstien med switches
'3) Start programmet (Shell returnerer prosess-id
'wait satt til False, dvs spilles asynkront)
id = Shell(programsti, AppWinStyle.MaximizedFocus, False)
```

Note: Alle programmer kan startes fra VB på denne måten.

MM-kontroll

Hvis du simpelthen skal spille en MM-fil, anbefaler jeg å bruke et standardprogram og ikke å lage ditt eget! Poenget med å bruke MM-kontrollen, er å styre MM-spilleren fra et program, f.eks. til å spille en lydfil når programmet ditt åpnes/lukkes, spille en MM-fil (fanfare?) når brukeren får til noe osv. MM-kontrollen vil derfor i praksis gjerne være usynlig.

Det er mulig å spille multimedia innenfor skjemaet også. Da kan man bruke en Windows Media Player kontroll. Det viser jeg ikke her.

Advarsel

MM-kontrollen og MM-spillere gir stadig nye overraskelser. Du kan f.eks. oppleve at programmet ditt er stanset og spilleren lukket, men likevel hører du lyden gå. Du kan også oppleve at spilleren nekter å la seg lukke, eller at den ikke vil åpne seg osv. Jeg har også opplevd å få flere lydfiler spilt oppå hverandre, og ikke klart å stoppe noen av dem.

Tema H2 – Dato og klokkeslett

Mange programmeringsspråk har datatype og funksjoner for datoer og klokkeslett. Det har også VB.

Strukturen DateTime

> **Note:** Tidligere har VB brukt **datatypen** *Date* og den virker ennå[40]. **Strukturen** *DateTime* er kommet i senere versjoner og skal overta, så her holder jeg meg til den.
> En *struktur* kan sees på som en klasse, men instansene er en **samling verdier** og ikke et **objekt**. I motsetning til et objekt, kan en struktur aldri være *Nothing* – delene av strukturen (verdier i samlingen) initieres til standardverdier som enkle variable. *DateTime* initialiseres nøyaktig som datatypene Integer, Boolean osv. Den kan skapes med *New* om man ønsker det, eller enklest bare deklareres og tilordnes verdier. Du kan lese mer om forskjellene på strukturer og klasser i VB Hjelp – søk etter "Structures and classes".

DateTime brukes enten du bare skal representere en dato, et tidspunkt eller begge deler. Du deklarer den på vanlig måte:

```
Dim datotid As DateTime
```

Defaultverdien for *DateTime* er 1. januar år 1 kl 00:00:00, og nøyaktigheten er 100 nanosekunder (ns = 10^{-9} sek). Siste lovlige *DateTime* er rett før 1. januar år 10000.

Når du skal tilordne den en fast verdi, må du oppgi datoen på engelsk måte og omsluttet av #:

```
Dim datotid As DateTime = #1/29/2008#
```

Når du skriver inn en tid, kan du velge mellom 12- eller 24-timers klokke, f.eks.:

```
Dim datotid As DateTime = #1:10:05 PM#
Dim datotid As DateTime = #13:10:05#
```

Den siste vil likevel umiddelbart bli endret av VB til det samme som den første.

Du kan selvsagt oppgi både dato og tid:

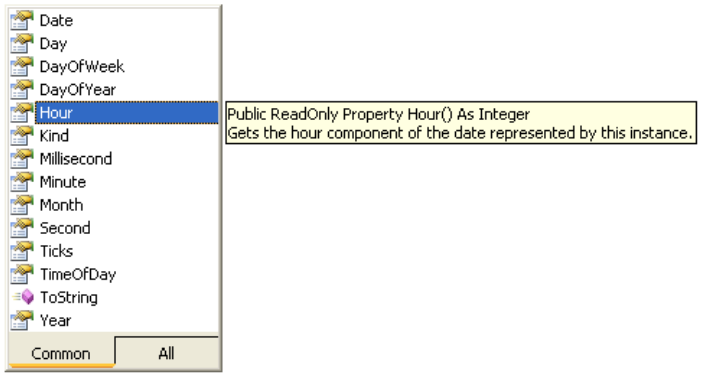
```
Dim datotid As DateTime = #1/29/2008 1:10:05 PM#
```

Du kan også bruke konstruktøren *New DateTime* og skrive f.eks.

```
Dim datotid As New DateTime(2006, 5, 29, 13, 10, 5, 0)
```

Alternativt bruker du *Now* som er en komplett *DateTime*. Som alle *DateTime* har den en rekke egenskaper, f.eks. dato, time, minutt osv.:

```
Dim hour As Integer
hour = DateTime.Now.
```



Eksempel:

```
Dim hour As Integer = DateTime.Now.Hour
```

[40] Faktisk så er nå *Date* blitt et alias for *DateTime*, så de to er nå nøyaktig det samme.

Pass på at de forskjellige egenskapene er av forskjellige typer – f.eks. ser du at *Hour* er en *Integer*, *Ticks* er antall "klokkepulser" (à 100 ns) siden midnatt 1.1.1 og er en Long, mens *TimeOfDay* angir tiden som er gått fra midnatt i dag, og er av typen *TimeSpan*.

Konvertering til/fra DateTime

Du konverter til en *DateTime* med *DateTime.Parse(s)* der *s* er en streng som angir en dato med/uten klokkeslett.

```
Dim datotid As DateTime = DateTime.Parse("29. januar 2008")
```

Du kan også bruke *TryParse*:

```
Dim datotid As DateTime
Dim OKdato As Boolean
OKdato = DateTime.TryParse("29. januar 2008", datotid) 'True
OKdato = DateTime.TryParse("29. tullemåned 2008", datotid) 'False
```

Den første returnerer *True* og setter *datotid* til datoen, den andre gir *False* og setter *datotid* til 0.

Du kan også sjekke om strengen er en dato, med funksjonen *IsDate()*:

```
Dim datostreng As String = "1. mai 2007"
If IsDate(datostreng) Then MsgBox("OK")
```

Du kan sammenlikne datoer med de vanlige sammenlikningsoperatorene (>, <, = osv.).

Konvertering fra dato til streng, gjør du enten med *ToString()* eller med *Format*:

```
Dim streng As String
Dim datotid As DateTime = DateTime.Now
streng = "ToString: " & datotid.ToString & vbNewLine _
& "Long Date: " & Format(datotid, "Long date") & vbNewLine _
& "Short Date: " & Format(datotid, "Short date") & vbNewLine _
& "Long Time: " & Format(datotid, "Long time") & vbNewLine _
& "Short Time: " & Format(datotid, "Short time")
MsgBox(streng)
```

gir denne meldingsboksen:

Regne med datoer

Du kan regne med datoer, men de vanlige regneoperatorene (+, - osv.) er ikke definert. Isteden bruker du *DateTimes* egne metoder *AddYears*, *AddDays*, *AddHours*, *AddMinutes*, osv., f.eks.

```
Dim datotid As DateTime = #1/29/2008# '29. januar 2008
datotid = datotid.AddDays(15) '+15 dager = 13. februar 2008
```

Det er imidlertid lov å legge til et *TimeSpan* som faktisk har regneoperatorene, f.eks. beregner jeg hvor mange dager som er gått siden 1.1.2007:

```
Dim dager As TimeSpan = DateTime.Now - #1/1/2007#
MsgBox(dager.TotalDays) 'viser antall dager som double
```

Andre nyttige metoder for DateTime

Du finner *DateTime Members* i VB Hjelp. Her er noen få, nyttige metoder og egenskaper for *DateTime*:

- ✓ .*DayOfWeek* angir ukedag (på engelsk) og er av typen *DayOfWeek*. Du kan bruke den f.eks. slik:

```
  Dim datotid As DateTime = DateTime.Now
  Select Case datotid.DayOfWeek
Case DayOfWeek.Monday
dag = "Mandag"
… osv
```

- ✓ .*DayOfYear* angir dagsnummeret i året – fra 1 til 366. Fin til renteberegninger o.l.
- ✓ .*ToDay* tilsvarer *Now* men returnerer bare dagens dato (og ikke klokkeslett)
- ✓ .*TimeOfDay* returnerer antall timer, minutter, sekunder osv. siden sist midnatt som en *TimeSpan*.
- ✓ .*DaysInMonth* returnerer antall dager i måneden
- ✓ .*IsDaylightSavingTime* returnerer True hvis det er sommertid
- ✓ .*IsLeapYear* returnerer *True* hvis det er et skuddår

Tema H3 – Timer – når noe skal skje om en stund

Jeg har tidligere gitt dere en rutine som lager visse pauser. Koden for den kan være slik.

```
Dim Slutt As DateTime 'når pausen slutter
Slutt = DateTime.Now.AddSeconds(15) 'Nå + 15 sekunder
Do While DateTime.Now < Slutt
    Application.DoEvents()
Loop
```

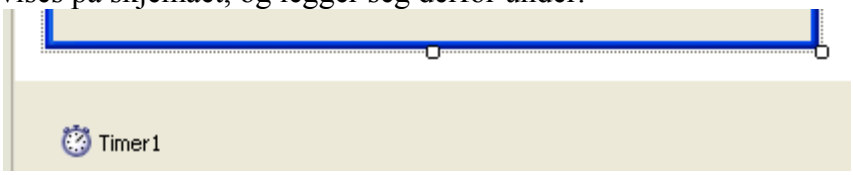
Iterasjonen (”løkken”) går da ”for full fart” i 15 sekunder, og bruker så mye CPU-ressurs som den får tak i, og det er en alvorlig ulempe. Det vil jo forsinke andre programmer som kjører parallelt. Den ble da også presentert som “primitiv”.

Bedre er det da å sette applikasjonen til å "sove". Det gjøres slik:

```
Threading.Thread.Sleep(15000) 'pause i millisekunder
```

På kjøkkenet hos meg har jeg en ”timer”. Den stiller jeg på en viss tid, starter den, og så teller den ned til 0. Da starter den en alarm. I VB har vi også en slik timerkontroll i verktøykassen. Når noe skal skje om en viss tid, er den kjekk å ha. Den bruker ikke særlige ressurser mens den går (under 1 % av prosessorkapasiteten hos meg), og det er en stor fordel. Videre kan programmet gjøre annet i mellomtiden.

Du henter den fra verktøykassen (under *Components*) til skjemaet på vanlig måte. Den kan ikke vises på skjemaet, og legger seg derfor under.



Timerkontrollen har egentlig bare tre egenskaper vi bryr oss om: *Name*, *Interval* og *Enabled*.

- ✓ *Name* har vi jo sett før☺
- ✓ *Interval* er tiden den skal telle ned i millisekunder. Nøyaktigheten er 1/18-dels sekund (ca. 55 millisekunder)[41]. Hvis maskinen blir opptatt med annet, kan timeren bli litt forsinket, så hvis du skal måle tidsforløp, bør du heller anvende systemklokken. Største *interval* er 64 767 millisekunder = noe over ett minutt.
- ✓ *Enabled* = *True* starter timeren. *Enabled* = *False* vil stoppe timeren igjen. Man kan også bruke metoden *Start* og *Stop*.

Når timeren har tellet ned til 0, genererer den hendelsen *Tick*. Deretter starter den umiddelbart med å telle ned igjen. Du lager følgelig en hendelsesprosedyre som håndterer timerens Tick-hendelse, f.eks.

```
Timer1.Interval = 1000 'millisekunder = 1 sekund
Timer1.Enabled = True 'starter nedtellingen

Private Sub Timer1_Tick(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Timer1.Tick
    Beep()
End Sub
```

Siden hendelsen *Tick* her genereres hvert sekund, vil dette programmet gi et ”pip” hvert sekund.

Scheduler/Cron

Jeg finner ikke dokumentert hvordan timerkontollen er laget, men jeg har en mistanke om at den bruker operativsystemets *Scheduler*. Det er et program som operativsystemet bruker til å starte

[41] VB tilbyr også timeren System.Threading.Timer. Den krever ikke et skjema (fint for biblioteker) og har millisekund nøyaktighet. Den kan settes til å slå til først etter hele 2^{32} = 4 294 967 294 millisekunder = ca. 49 døgn. Den er imidlertid betydelig vanskeligere å bruke.

handlinger når en bestemt begivenhet skjer eller et klokkeslett inntreffer (evt. med visse mellomrom). Istedenfor å telle ned fra f.eks. 1000 millisekunder, kan timeren kalle Scheduler og sette opp en hendelse om ett sekund. Deretter kan timeren "hvile" og vente på signalet fra Scheduler. I Windows finner du programmet *Task Scheduler* i mappen *Control Panel|Administrative Tools*. Der kan du selv legge til jobber som skal kjøres på bestemte tidspunkt eller ved oppstart/logon. I Unix/Linux finnes en tilsvarende *Cron* program (gr. *Kronos* = Tid), som sjekker tabellen *CronTab* hvert minutt for å se om noe i tabellen skal startes. Denne virker altså bare hvert minutt, og trigges bare på tid.

Eksempel

I dette programmet er det to knapper: *butStart* og *butStopp* som starter/stopper en timer som gir "pip" hvert sekund. Timerkontrollen er kalt *timerSekund*.

```
Public Class TimerEksempel
    Private Sub TimerEksempel_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        timerSekund.Interval = 1000 'millisekunder = 1 sekund
    End Sub

    Private Sub timerSekund_Tick(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles timerSekund.Tick
        Beep()
    End Sub

    Private Sub butStart_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butStart.Click
        timerSekund.Enabled = True 'starter nedtellingen
        'Alternativt: timerSekund.Start() 'starter nedtellingen
    End Sub

    Private Sub butStopp_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butStopp.Click
        timerSekund.Stop() 'alt: timerSekund.Enabled = False
    End Sub
End Class
```

Intervaller lenger enn ca. ett minutt

Hvis vi vil at noe skal skje f.eks. hvert femte minutt ($5 \cdot 60 \cdot 1000 = 300\,000$ *millisekunder*), kan vi ikke bruke timeren alene, siden den ikke aksepterer intervaller utover 64 767. Da bruker vi timeren slik at den slår til hvert minutt ($60 \cdot 1000 = 60\,000$ *millisekunder*), og så teller vi antall ganger den har slått til:

```
    Private Sub TimerEksempel_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        timerMinutt.Interval = 60000 'millisekunder = 1 minutt
    End Sub

    Private Sub butStart5_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butStart5.Click
        timerMinutt.Start()
    End Sub

    Private Sub timerMinutt_Tick(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles timerMinutt.Tick
        Static minutter As Integer = 0
        minutter += 1
        If minutter Mod 5 = 0 Then
            Beep()
            minutter = 0 'så minutter aldri blir mer enn Integer.MaxValue
        End If
    End Sub
```

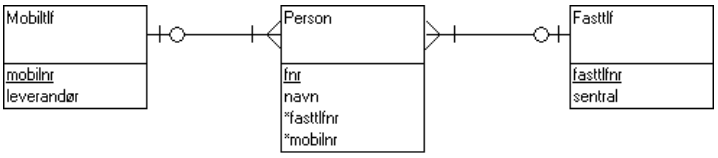
Tema I – Database del 1

Hvorfor program over databasen?

Relasjonsdatabaser har en del klare svakheter. Først og fremst dreier det seg om god domenekontroll og referanseintegritet. Entitetsintegriteten klarer databasene bedre, men også her er det noen problemer.

En database skal f.eks. ha e-postadresse til kundene. Det vil åpenbart bli definert i databasen som et tekstfelt (*Varchar(30)* e.l.). Domenet for e-postadresser er imidlertid ikke en hvilken som helst streng – det er klare syntaksregler for hvordan en e-postadresse er bygget opp[42], og det er helt håpløst å kontrollere det med vanlig DDL.

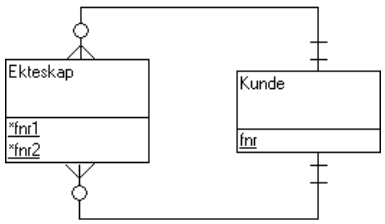
Her skal f.eks. en *Person* ha én og bare én telefon:



Note: Hver person skal ha registrert enten en mobiltlf eller en fasttlf, men ikke begge

I relasjonsdatabasen er det ikke mulig å sikre at en post i persontabellen *enten* skal knyttes til én post i mobiltelefontabellen *eller* til én post i fasttelefontabellen (men ikke til begge). Vi må tillate null både for fremmednøkkelen *fasttlfnr* og for *mobilnr*. Dessuten er reglene for fødselsnummer så kompliserte at vi ikke kan regne med at databasen kan sjekke domenet "Lovlig, norsk fødselsnummer".

I denne modellen viser vi ekteskap mellom kunder.



Ekteskapstabellen kan da inneholde poster der (1) *fnr1*=Per, og *fnr2*=Kari, og samtidig (2) at *fnr1*=Kari og *fnr2*=Per, slik:

Ekteskap

fnr1	fnr2
Per	Kari
Kari	Per

Dette er en form for dubletter – ekteskapet registreres to ganger med oppdateringsproblemer og plasskrav som resultat. Det er ingen løsning å kreve at *fnr1* og *fnr2* viser til hvert sitt kjønn. Egentlig skulle vi presisert at hvis et fødselsnummer finnes i *fnr1*, så skal det ikke kunne finnes i *fnr2*, men det kan vi ikke uttrykke med DDL.

Problemet for begge disse eksemplene er at den logikken som kreves ikke kan defineres i DDL, som egentlig er ganske primitivt. Også SQL har mangler.

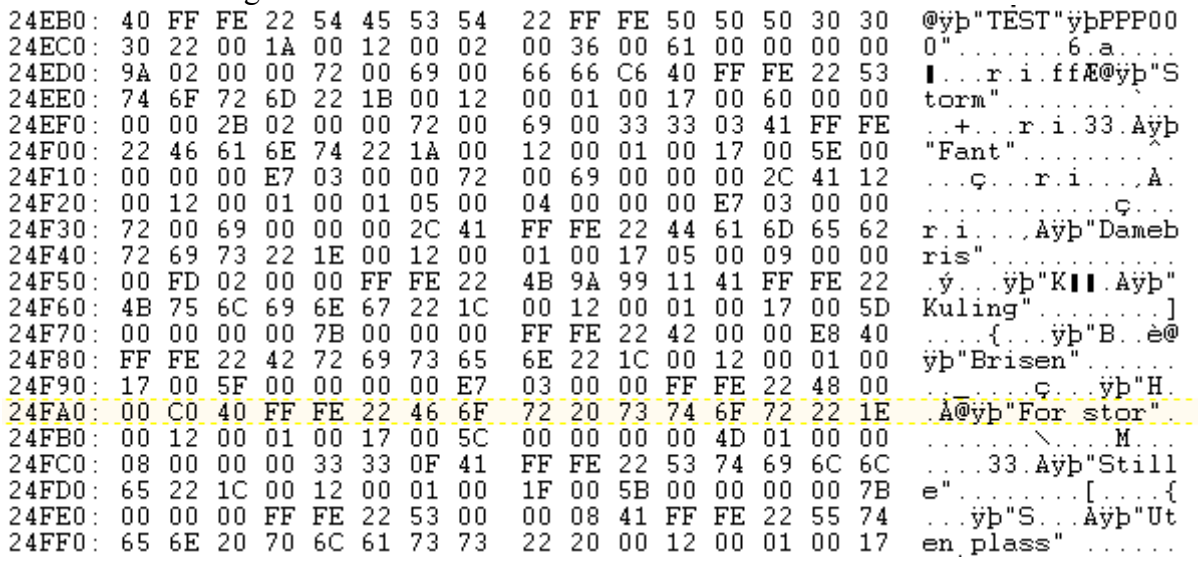
Slike problemer løses meget lettere med et imperativt språk som f.eks. VB. Derfor er det aktuelt å ha et imperativt program som lag over databasen for å sikre "business rules", data- og

[42] En enkel beskrivelse finnes på http://en.wikipedia.org/wiki/E-mail_address. (Noen databasesystemer har utvidede skranker kalt *check constraints*, slik at databasen *kan* foreta en rimelig kontroll, men det er påbygninger, utenfor SQL. MySQL tillater det i syntaksen, men ignorerer det.)

referanseintegritet. Med et imperativt språk kan vi også søke enkelt og få (for) mange resultater som programmet så går igjennom for å velge bare akkurat de postene brukeren vil ha.

Tre-lags arkitektur

Dataene i en database er fysisk representert på en harddisk. De kan nok leses med en teksteditor, men de er lite forståelige. Her er f.eks. et bilde av en Accessdatabase:

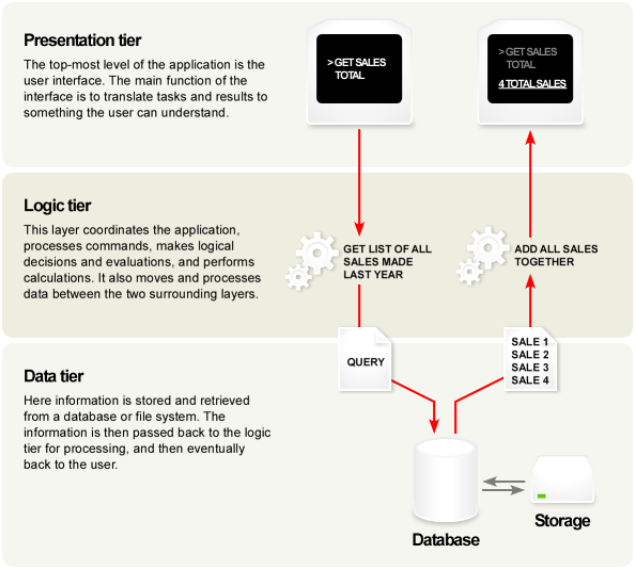


Hvis du ser nøye etter, så er båtene "Fant", "Damebris" og andre lagret her. Men det vil ikke være særlig enkelt å endre disse dataene på fornuftig måte! Det vil også være svært utrygt å la brukerne gjøre noe slikt: De mangler kompetansen og vil ødelegge for hverandre. For å løse dette, har alle databaser et program – Data Base Management System, DBMS – som gir tilgang til dataene på en mer fornuftig måte. DBMS skal bistå med

- å sikre domeneintegritet (dataene har lovlige verdier), entitetsintegritet (primærnøkler og alternative nøkler er unike og ikke null) og referanseintegritet (alle fremmednøkler som ikke er null, peker til en eksisterende post)
- adgangskontroll
- samtidighet og konfliktløsning
- automatisk retting av feil ("recovery")
- databaseadministrasjon (f.eks. skjemautvikling, sikkerhetskopiering og brukeradministrasjon)

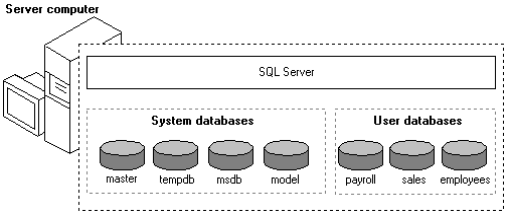
Også DBMS er kompliserte for vanlige brukere og presentasjonen av resultater er ofte mangelfull. Derfor legges ofte et annet program oppå for å håndtere brukerne. Da får man enklere grensesnitt, f.eks. skjemaer med knapper, tekstfelt o.l. Mellom dette programmet og DBMS legger man gjerne enda et program som gjør om brukerens ønsker ("Vis meg alle kunder i Oslo") til kommandoer som DBMS kan tolke ("select * from tblKunder where sted = "Oslo";"). På denne måten oppstår det tre "lag", kalt

- Presentasjonslag som håndterer grensesnittet med brukeren. Det kan være en nettside eller en applikasjon. Det fanger opp brukernes ønsker og presenterer resultatene på en "pen" og forståelig måte, slik de ønsker.
- Logisk lag (også kalt applikasjonslag, business lag, datatilgangslag og middellag) gjør om brukerens ønsker til DBMS kommandoer – gjerne SQL – og henter dataene i "rå form".
- Datalag som lagrer dataene fysisk og henter/endrer dem på en kontrollert måte. Datalaget inkluderer DBMS som tolker kommandoen fra det logiske laget og omgjør dem til kontrollert, fysisk manipulering av data. Dataene deles gjerne i de "egentlige" dataene og metadata (også kalt skjema) som inneholder "data om dataene", f.eks. hvilke tabeller og indekser som finnes, kolonner, domener, brukere, rettigheter osv.



Tre-lags modellen - prinsippskisse

Note: I MS Access er alle lagene slått sammen i ett produkt, så brukeren kan få dataene ”pent” presentert og enkelt håndtert, samtidig som alt omgjøres til SQL setninger og databasen oppdateres. Dette er uvanlig, og gir store problemer med skalerbarhet (Access blir tung og vanskelig ved store datamengder og mange brukere). Microsoft har da også laget MS SQL Server som bare implementerer datalaget og er det som brukes i praksis for ”ordentlige” databaser.



Med en oppdeling av systemet i flere lag, oppnår vi først og fremst at lagene blir relativt uavhengige av hverandre. Vi kan f.eks. lage flere presentasjonslag (nettsider, applikasjon, Applet) over samme logiske lag. Hvis det logiske laget bruker SQL, kan vi også bytte databaseprodukt uten å endre særlig på det logiske laget (fordi SQL er standardisert). Det blir også enklere å legge systemdelene på adskilte maskiner, slik at vi kan ha en dedikert, kraftig maskin for datalaget (database server) og legge presentasjonslaget på hver klient, osv. Derved spres arbeidsbelastningen på flere prosessorer (men det blir mer kommunikasjon).

Fordelene ved lagdeling er mye større enn ulempene. Derfor er det blitt svært vanlig – nesten enerådende – for ”ordentlige” databasesystemer å være lagdelt. Dere må altså lære hvordan det gjøres.

Kontakt mellom presentasjonslaget og datalaget

Vi skal lage presentasjonslaget med Visual Basic. Vi kunne brukt nesten et hvilket som helst programmeringsspråk, da de alle er gode på brukergrensesnitt og logiske kontroller av input. Til datalaget skal vi bruke MySQL, men igjen kunne vi brukt MS Access, MS SQL Server, Oracle eller et av de mange andre produktene. Det logiske laget skal vi holde sammen med presentasjonslaget, men her får vi mye assistanse av objekter. Til dette benytter vi et klassebibliotek spesielt laget for MySQL i .NET-miljø.

Det er også mulig å få svært mye hjelp av programmeringsmiljøet i Visual Basic, hvis databasen er en av de som miljøet ”kjenner”. I utgangspunktet leveres Express-utgaven med ferdige klasser for MS SQL Server og MS Access. Den fulle versjonen har mulighet for flere, herunder også eksterne

databaser (Express må ha basen lokalt). Da kan man (jeg tenker å demonstrere det) å programmere et presentasjons- og logisk lag praktisk talt uten å røre tastaturet – alt skjer med ”klikk” og ”dra”. Når vi ikke skal gjøre det slik i dette kurset, så er det fordi det bare gir kunnskap om et, bestemt programmeringsmiljø (Visual Studio) og ikke er overførbart til andre programmeringsspråk og databaser.

Med klassebiblioteker kan vi også knytte VB-programmet til alle ODBC databaser, MySQL, Oracle, PostgreSQL, Sybase og VistaDB. Vi må da hente, installere og referere til et passende klassebibliotek, som vanligvis tilbys gratis av databaseprodusenten (ofte mot registrering).

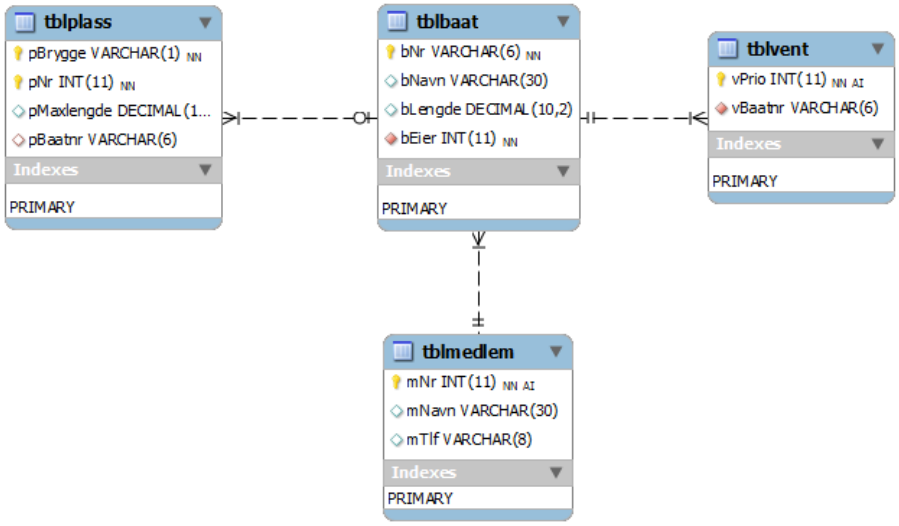
Forberedelser

MySQL databasen

Til disse eksemplene skal vi benytte en MySQL database. Jeg forutsetter at du har MySQL Server installert lokalt på maskinen din – selv har jeg nå (mars 2012) MySQL versjon 5.6.3 og bruker MySQL Workbench 5.2.35. Det skal ikke spille noen rolle om du har andre versjoner. Du må opprette databasen ”baatregister”. Nederst i dette notatet finner du script som jeg håper☺ virker. Det gjør det i alle fall hos meg.

Jeg har brukeren ”Knut” med passord 'Tunk" og – litt uforsiktig – med alle rettigheter.

Databasen er strukturert slik:



Du ser at alle kolonnene i tabellen *tblplass* har førstebokstav *p*, alle i *tblbaat* begynner med *b* osv. (Det er et triks jeg ofte bruker for å få alle kolonnenavn unike. Da slipper jeg alltid å måtte angi tabellnavnet for å presisere hvilken tabell det gjelder.)

```
Scratch
1  use baatregister;
2  select pBrygge, pNr, bNr from tblPlass, tblBaat where pBaatnr = bNr;
```

Overview | Output | Snippets | Result (1)

Fetched 6 records. Duration: 0.000 sec, fetched in: 0.000 sec

pBrygge	pNr	bNr
A	2	BAC103
A	3	KLM202
B	1	TIC004
B	2	LBA019
B	3	BOB017
A	1	FAB205

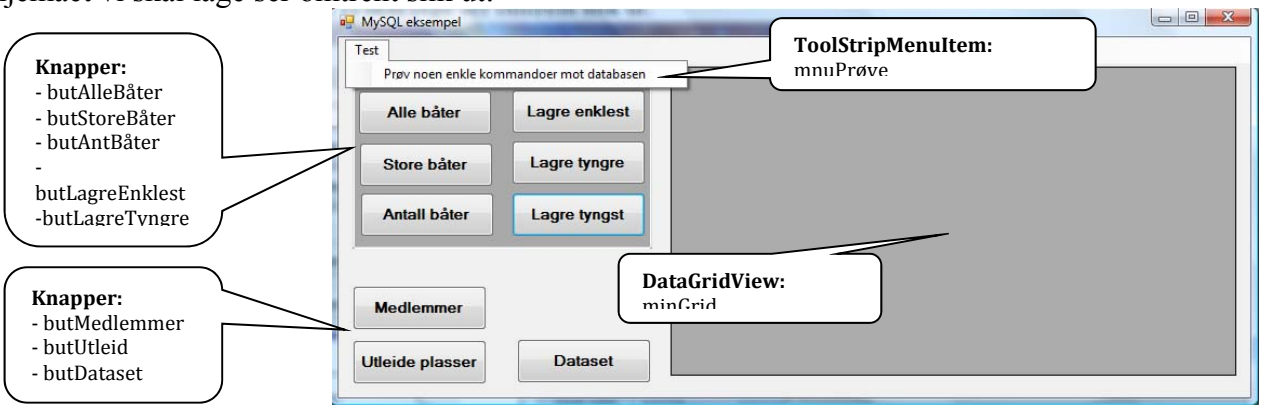
Hente/installer MySQL biblioteket for .NET

Du må også installere "*MySQL Connector/NET*". Du henter den fra http://dev.mysql.com/downloads/connector/net/6.3.html#downloads .
Den installerer *MySQL.Data.dll* som vanligvis legges i en submappe til *MySQL* programmet. Du finner manual og annet på http://dev.mysql.com/doc/refman/5.5/en/connector-net.html.

Klargjøre VB-programmet for bruk av MySQL

Start et nytt VB-prosjekt. Kall det *MySQL_eksempel* og skjemaet *frmMySQL_eksempel*. Sett passende heading på skjemaet. Så må du legge til en referanse til *MySQL.Data.dll* i programmet (*Project|Add Reference…*). Du gjør det som i prosjektets egenskaper under fanen *Browse* som med de bibliotekene vi laget selv tidligere.

Skjemaet vi skal lage ser omtrent slik ut:



Vi lager knappene, menyen og datagriden, og setter alle navn og tekster slik vi ønsker. Så er vi klare for å programmere.

Knytte programmet til databasen

For å gjøre programmeringen enklere, bør du importere MySQLs navnerom:

```
Imports MySql.Data.MySqlClient
```

helt øverst i programmet (foran skjemaklassen). Husk også å sette på Option Strict!

1. Deklarasjoner på skjemanivå

Objekter som brukes i mange av prosedyrene i eksemplet, er deklarert på skjemanivå. Jeg kommer senere tilbake til hva disse objektene brukes til.

```
Public Class frmMySQL_eksempel
    Private mittDatasett As DataSet '(VB, ikke MySql.Data.MySqlClient)
    Private minConnectionString As String '(VB)
    Private minConnection As MySqlConnection 'MySql.Data.MySqlClient
    Private minCommand As MySqlCommand 'MySql.Data.MySqlClient
    Private minAdapter As MySqlDataAdapter 'MySql.Data.MySqlClient
    Private minDatareader As MySqlDataReader 'MySql.Data.MySqlClient
    Private tblMedlem As DataTable '(VB, ikke MySql.Data.MySqlClient)
    Private tblBåt As New DataTable() '(VB, ikke MySql.Data.MySqlClient)
```

2. Skape nødvendige koblinger osv. ved oppstart

Når programmet starter, må det knyttes til databasen. Det må vi programmere:

```
Private Sub frmMySQL_eksempel_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
    'Skaper et connection- og et command-objekt
    '1. Skap MySQLConnection for brukeren "Knut" passord "Tunk"
    minConnectionString = _
        "Server=localhost;" & _
        "Database=baatregister;" & _
        "Uid=Knut;" & _
        "Pwd=Tunk;" & _
        "Connect Timeout=30;" & _
        "CharSet=utf8;"
    minConnection = New MySqlConnection(minConnectionString)
    '2' Åpne forbindelsen:
    minConnection.Open()
    '3. Skap MySqlCommand (uten SelectCommandText)
    minCommand = New MySqlCommand()
    minCommand.Connection = minConnection
End Sub
```

Hvis du har annet brukernavn og passord (som selvsagt burde vært hentet med inputboks), er det enkelt å bytte om i connectionstring.

Pass på at du setter *CharSet* til det du har valgt for databasen, ellers vil du lett få ”rare” tegn i programmet. Her skulle det f.eks. stått ”båtNavn”:

	bNr	bÃ¥tNavn	bLengde	bEier
▸	KLM202	Stille	8.95	333
	LBA019	Fant	8.20	555

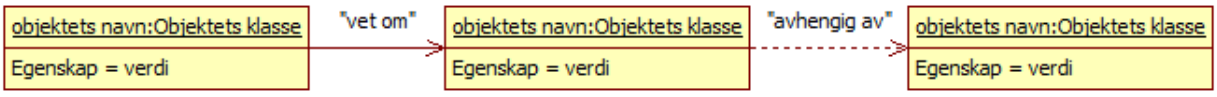
Note: Det meste av det som gjøres i load-prosedyren ovenfor kan gå galt. Det burde følgelig vært en try-catch for å håndtere dette. Det er ikke tatt med her, da dette bare er en demo.

Note: Programmet åpner en *Connection* – den bør lukkes igjen på et passende sted før programmet avslutter:
`minConnection.Close()`

Da er det bare å prøvekjøre, og håpe at du ikke får feilmeldinger!

Litt syntaks for diagrammene nedenfor

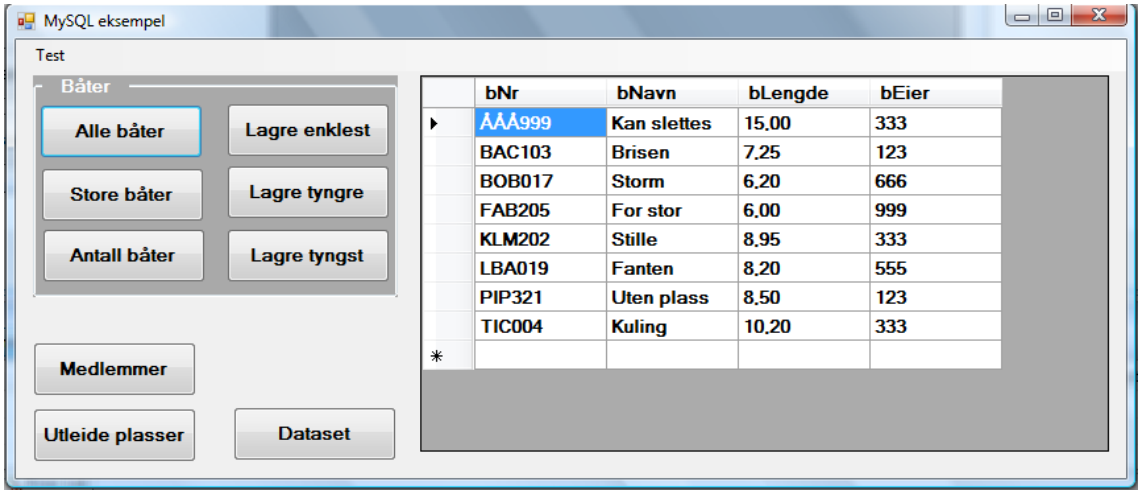
I diagrammene nedenfor har jeg brukt en litt tilpasset variant av UML. Objektene er tegnet inn som rektangler. Mellom dem er det to typer av relasjoner: ”Vet om” og ”avhengig av”.



Når objekt A ”vet om” objekt B, har A en egenskap (variabel) som referer til B. Når objekt C er ”avhengig av” object D, så bruker C en tjeneste som D tilbyr. Streken under navn og klasse indikerer at dette er et objekt og ikke en klasse.

3 Hente data med DataAdapter og vise dem på skjemaet i en grid

Dette er klart den enkleste måten å vise data på. For brukeren er det også ganske naturlig å se dem som tabell.

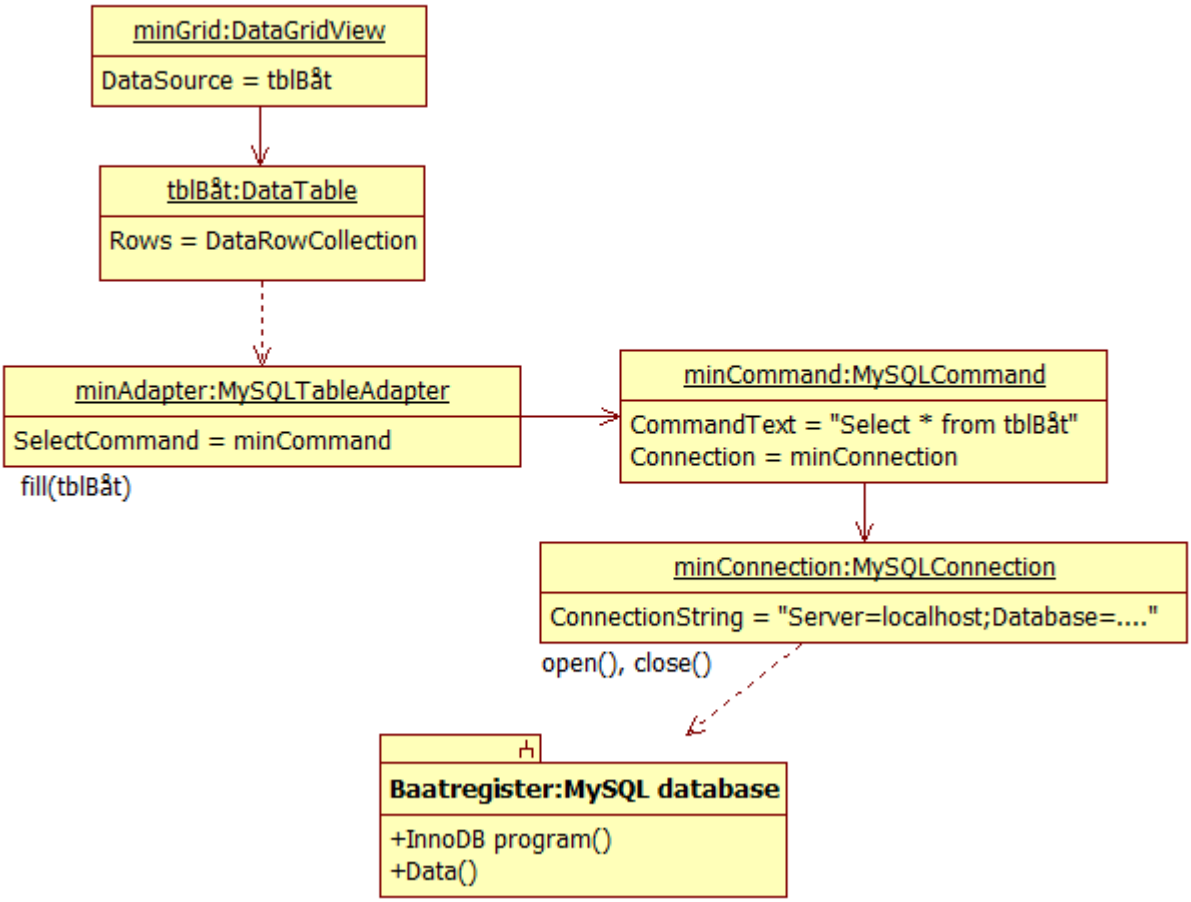



```
Private Sub butAlleBåter_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAlleBåter.Click
    'Fyller en tabell med data og viser dem i en grid
    '1. Skap tabellen:
    tblBåt = New DataTable("tblBåt")
    '2. Skap adapter (Connection er satt ved oppstart):
    minCommand.CommandText = "Select * from tblBaat"
    minAdapter = New MySqlDataAdapter(minCommand)
    '3. Fyll data i tabellen:
    minAdapter.Fill(tblBåt)
    '4. Fyll en datagrid med tabellens data:
    minGrid.DataSource = tblBåt
End Sub
```

Vi skaper her en vanlig datatabell (i RAM), en *MySQLDataadapter* med passende spørring – med eller uten semikolon – og fyller tabellen med metoden *Fill()* som bruker spørringen. *minGrid* er en gridkontroll og ved å sette *DataSorce* knytter vi den til tabellen. Vi kan endre innholdet ved å endre select-setningen og fylle tabellen igjen med den nye spørringen. Legg merke til at *minCommand* ikke er en streng, men et *MySQLCommand*-objekt med egenskapen *CommandText*.

Vi behøver ikke å spesifisere hvilke kolonner griden skal ha – den tilpasser seg det som kommer (den har en samling med kolonner og en samling med rader). Derimot kan det være aktuelt å tilpasse griden mht til hva brukeren skal få lov til å gjøre og om størrelsen av kolonner/rader automatisk skal tilpasses innholdet. Griden har mange egenskaper som regulerer dette. I oppgaven som du skal gjøre i forbindelse med denne teorien, skal du forsøke å finne ut av det.

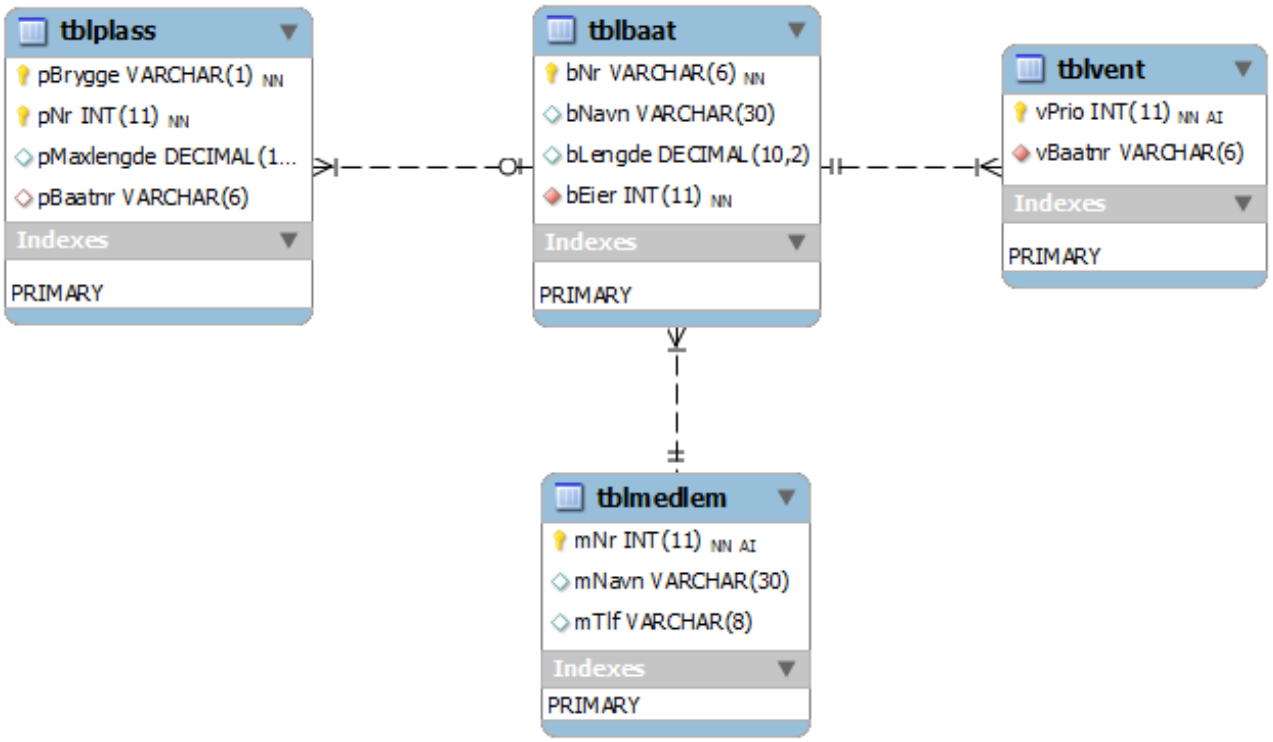
Fyll en tabell med rader (og vis dem i en grid)



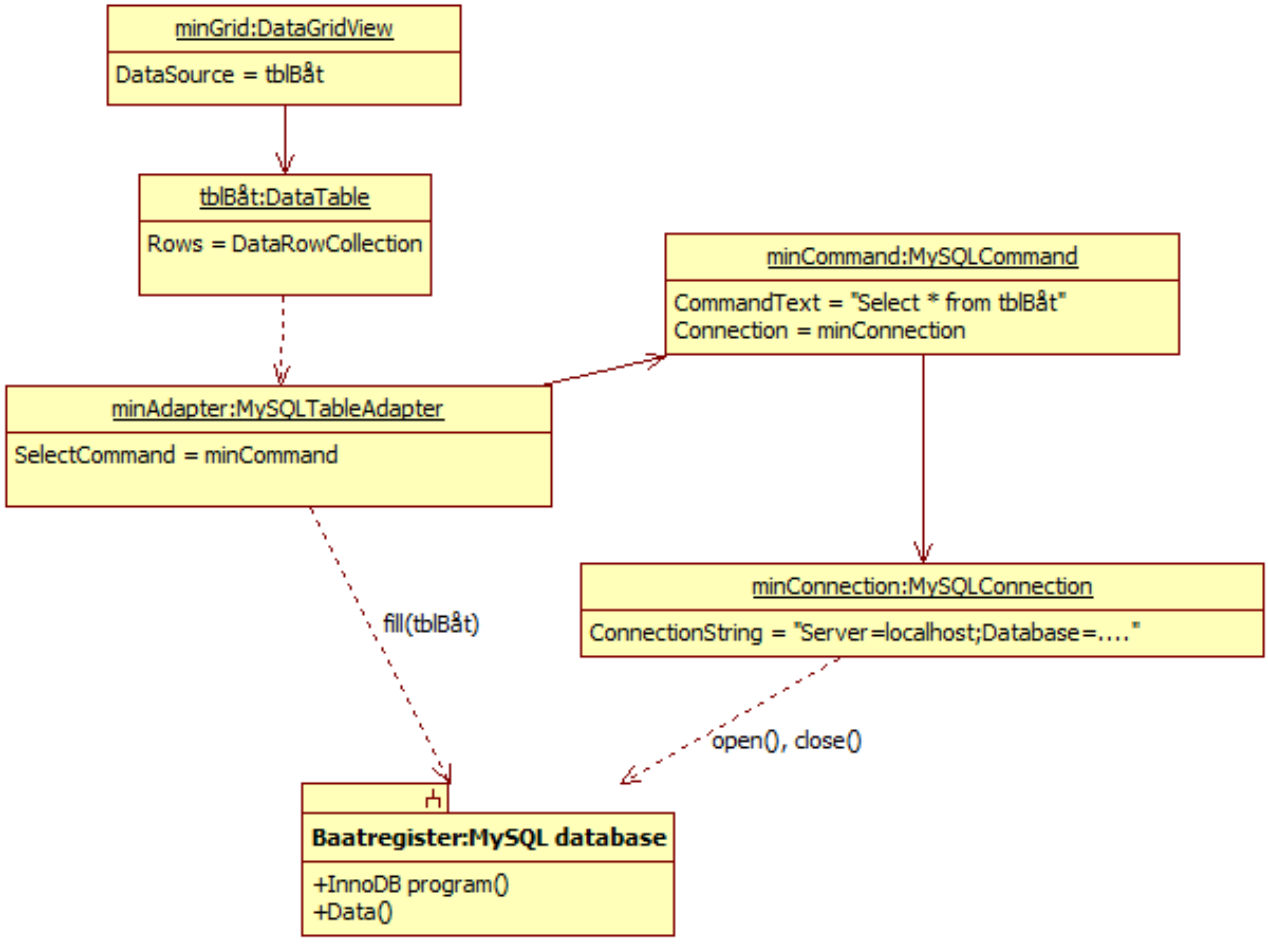
Script for databasen *baatregister*

```
-- START SKRIPTET

-- DROP KNUT (HVIS HAN FINNES)
---DROP USER 'Knut'@'localhost'; 'fjern kommentaren for å slett 'Knut'

-- SKAP DATABASEN OG GI RETTIGHETER
-- --------------------------------
CREATE DATABASE IF NOT EXISTS baatregister
DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8_danish_ci;
USE baatregister;
CREATE USER 'Knut'@'localhost' IDENTIFIED BY 'Tunk';
GRANT ALL ON baatregister.* TO 'Knut'@'localhost';


-- FJERN EVT. GAMLE TABELLER
-- -------------------------
DROP TABLE IF EXISTS baatregister.tblvent;
DROP TABLE IF EXISTS baatregister.tblplass;
DROP TABLE IF EXISTS baatregister.tblbaat;
DROP TABLE IF EXISTS baatregister.tblmedlem;


-- SKAP TABELLER
-- -------------
CREATE TABLE baatregister.tblmedlem (
  `mNr` int(11) NOT NULL AUTO_INCREMENT,
  `mNavn` varchar(30) DEFAULT NULL,
  `mTlf` varchar(8) DEFAULT NULL,
  PRIMARY KEY (`mNr`)
)
ENGINE=innodb DEFAULT CHARSET=utf8;

CREATE TABLE baatregister.tblbaat (
  `bNr` varchar(6) NOT NULL DEFAULT '',
  `bNavn` varchar(30) DEFAULT NULL,
  `bLengde` decimal(10,2) DEFAULT NULL,
  `bEier` int(11) NOT NULL,
  INDEX (bEier),
  FOREIGN KEY (`bEier`) references tblmedlem (`mNr`),
  PRIMARY KEY (`bNr`)
)
ENGINE=innodb DEFAULT CHARSET=utf8;

CREATE TABLE baatregister.tblplass (
  `pBrygge` varchar(1) NOT NULL DEFAULT '',
  `pNr` int(11) NOT NULL DEFAULT '0',
  `pMaxlengde` decimal(10,2) DEFAULT NULL,
  `pBaatnr` varchar(6) DEFAULT NULL,
  INDEX (pBaatnr),
  FOREIGN KEY (`pBaatnr`) references tblbaat (`bNr`),
  PRIMARY KEY (`pBrygge`,`pNr`)
)
ENGINE=innodb DEFAULT CHARSET=utf8;

CREATE TABLE baatregister.tblvent (
  `vPrio` int(11) NOT NULL AUTO_INCREMENT,
  `vBaatnr` varchar(6) NOT NULL,
  INDEX(vBaatnr),
  FOREIGN KEY (`vBaatnr`) references tblbaat (`bNr`),
  PRIMARY KEY (`vPrio`)
)
ENGINE=innodb DEFAULT CHARSET=utf8;

--- DATA PÅ NESTE SIDE ---
```

```
-- LEGG INN DATA
-- -------------
-- Setter inn egne verdier for mNr istedenfor å benytte AutoIncrement
-- for ikke å få kluss med fremmednøkkelen i tblBaat.
-- Kan også bruke Null og håpe det beste.
insert into baatregister.tblMedlem values (1,'Ola Nordmann','-');
insert into baatregister.tblMedlem values (2,'Knut Hansen','22334455');
insert into baatregister.tblMedlem values (3,'Kari Trestakk','99996666');
insert into baatregister.tblMedlem values (4,'Karsten Malling','33333333');
insert into baatregister.tblMedlem values (5,'Mille Pettersen','55998822');
insert into baatregister.tblMedlem values (6,'Hans P. Jensen','-');
insert into baatregister.tblMedlem values (7,'Thor Uten båt','-');
insert into baatregister.tblMedlem values (8,'Olga Wille-Hansteen','11112222');

insert into baatregister.tblBaat values ('BAC103','Brisen',7.25,1);
insert into baatregister.tblBaat values ('BOB017','Storm',6.20,4);
insert into baatregister.tblBaat values ('FAB205','For stor',6.00,5);
insert into baatregister.tblBaat values ('KLM202','Stille',8.95,2);
insert into baatregister.tblBaat values ('LBA019','Fant',8.20,3);
insert into baatregister.tblBaat values ('PIP321','Uten plass',8.50,1);
insert into baatregister.tblBaat values ('TIC004','Kuling',10.20,2);

insert into baatregister.tblPlass values ('A',1,5.00,'FAB205');
insert into baatregister.tblPlass values ('A',2,7.50,'BAC103');
insert into baatregister.tblPlass values ('A',3,9.20,'KLM202');
insert into baatregister.tblPlass values ('B',1,12.00,'TIC004');
insert into baatregister.tblPlass values ('B',2,12.00,'LBA019');
insert into baatregister.tblPlass values ('B',3,10.00,'BOB017');
insert into baatregister.tblPlass values ('C',1,3.00,null);
insert into baatregister.tblPlass values ('C',2,3.00,null);
insert into baatregister.tblPlass values ('C',3,4.00,null);

insert into baatregister.tblVent values (1,'PIP321');


-- SE TABELLINNHOLD
-- ----------------
select * from baatregister.tblMedlem;
select * from baatregister.tblBaat;
select * from baatregister.tblPlass;
select * from baatregister.tblVent;

-- SLUTT PÅ SKRIPTET
```



Fyll en tabell med rader (og vis dem i en grid)



Tema J – Database del 2

Visual Basic Collections

Collection er en ordnet *samling* av elementer (objekter). At samlingen er ordnet, innebærer at elementene i samlingen har en gitt rekkefølge. Det er ikke noe krav at elementene skal ha forskjellig verdi[43].

VB bruker selv samlinger i mange sammenhenger. F.eks. er alle kontrollene på et skjema med i samlingen *Controls* og det samme er kontrollene i en *GroupBox*. En listeboks har en samling tekster i *Items*. Et datasett har en samling *Tables*. Radene i en datatabell er med i samlingen *Rows* og radenes kolonner i samlingen *Columns* og så videre.

Standard er at samlingen består av objekter av klassen *Object*. Det kan skape litt problemer, fordi alle objektene som lagres i en slik samling må gjøres om til objekter av klassen *Object* – det er automatisk men det tar litt tid. Særlig for store samlinger kan det bety noe. Videre vil vi alltid få et objekt av typen Object når vi henter et element fra samlingen, og det må vi da selv konvertere. Jeg anbefaler derfor at du heller bruker samlinger der du kan angi type (kalt *T* nedenfor), og det er disse jeg omtaler her.

Samlinger kan deles i tre typer:

1. **Indekserte samlinger.** Dette er samling der du kan hente ut elementer i den rekkefølge du ønsker, og hentede elementer blir liggende i samlingen. Du kan bruk en indeks (”elementnummeret”) til å hente ut et bestemt element, og du kan bestemme hvor i samlingen et nytt element skal legges til.
   Typisk indeksert samling er *List (Of T)* og *LinkedList (Of T)*. Bak en slik samling ligger i virkeligheten en vanlig array, men den øker størrelse automatisk ved behov.
2. **Ordnede samlinger.** Dette er samlinger der rekkefølgen for innlegging styrer rekkefølgen du kan få tak i dem igjen. Når et element er hentet fra en slik samling, blir det fjernet fra samlingen.
   Typisk er *Stack (Of T)* og *Queue (Of T)*. I en *stakkliste* er det sist innlagte som først hentes ut (LIFO = ”last in first out”). I en *kø* er det først innlagte som først hentes ut (FIFO = ”first in first out”, som en ”rettferdig kø” i en butikk). Å legge til et element heter *Push* og å hente heter *Pop*, og i tillegg kan man ”titte” på neste element (uten å fjerne det) med *Peep*.
3. **Katalogsamlinger.** Dette er samlinger der det lagres *verdipar* – en *unik nøkkel* og en *tilhørende verdi*. Samlingen holdes vanligvis sorter etter nøkkelen, og elementene kan hentes enten med indeks (”elementnummeret”) eller ved å angi nøkkelverdien.
   Typisk er klassen *Dictionary (Of Tkey, Tvalue)* der første type er typen for nøkkelen (f.eks. Integer) og den andre er typen for verdien (f.eks. String). Nøkkeltypen må kunne sorteres, og det stiller spesielle krav til klassene hvis vi bruker egendefinerte objekter.

Samlinger har metoder for å legge til, finne, fjerne, og hente et element. Videre kan man finne antall elementer, sjekke om et element finnes i samlingen og tømme hele samlingen. Navnene på disse metodene og egenskapene varierer. Sjekk i VB hjelp.

Den enkleste måten å gå igjennom alle elementene, er med *For Each…Next*. Syntaksen er slik:

```
For Each element [As datatype] In group
    [statements]
    [Exit For]
    [statements]
Next [element]
```

[43] En *mengde* er altså ikke en samling, da elementene i en mengde som kjent er uten ordning/rekkefølge og dessuten ikke kan være like.

Eksempel:

```
Dim tekstsamling As New Collection()
tekstsamling.Add("Knut")
tekstsamling.Add("Hansson")
For Each tekst As String In tekstsamling
    MsgBox(tekst)
Next
```

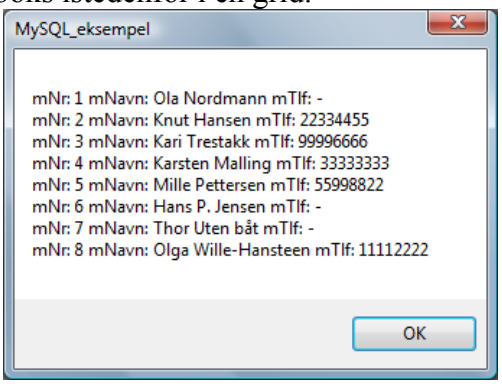
Les mer om samlinger på http://articles.techrepublic.com.com/5100-10878_11-1045372.html og søk "System.Collections.Generic Namespace" i VB hjelp når du skal lage en samling.

Mer om databaser

I del 1 viste jeg hvordan man kan knytte en MySQL database til et VB-program og hvordan man relativt enkelt (?) kan lage et skjema med en grid som fylles fra databasen. Jeg skal nå forklare hvordan vi kan få tak i, vise og oppdatere dataene med VB-programmet på andre måter.

4. Hente data med DataAdapter til en meldingsboks

I dette eksemplet skal vi igjen hente dataene med en *DataAdapter*, men når tabellen (i RAM) er fylt, skal vi vise dem i en meldingsboks istedenfor i en grid.



Dataene hentes på samme måte, men teksten i meldingsboksen må bygges opp linje for linje:

```
Private Sub butMedlemmer_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butMedlemmer.Click
    ' A. Fyller data i en tabell (som butAlleBåter)
    '1. Skaper en tabell:
    tblMedlem = New DataTable() 'skaper en tabell
    '2. Skaper en adapter (bruker en annen New - uten commandobjekt)
    minAdapter = New MySqlDataAdapter _
        ("Select * from tblMedlem", minConnection)
    minAdapter.Fill(tblMedlem)

    'B. Tabellen er fylt - vis alle radene i en meldingsboks
    'Gjøres generelt - vil automatisk tilpasse seg gjeldene kolonner
    '3. Henter rad for rad fra tabellen i løkke.
    'Kolonnenavnene er i tblMedlem.Columns mens radene er i tblMedlem.Rows
    Dim utstring As String = "" 'til meldingsboksen
    For Each rad As DataRow In tblMedlem.Rows 'rad for rad
        'Kolonne for kolonne:
        For Each col As DataColumn In tblMedlem.Columns
            utstring &= col.ColumnName & ": " 'kolonnenavnet
            utstring &= rad(col).ToString & " " 'kolonneverdien på denne raden
        Next
        utstring &= vbNewLine 'avslutt hver rad med linjeskift
    Next
    '4. Vis meldingen
    MsgBox(utstring)
End Sub
```

I punkt A fyller vi tabellen. I punkt B går vi igjennom tabellen rad for rad med *For Each rad*. Hver rad har data i kolonner som vi henter med *For Each col*. Hver kolonne har et navn som vi henter med *col.ColumnName* og verdiene finner vi ved å oppgi hvilken kolonne vi vil ha verdien for med *rad(col)*.

Tabellens samling *Rows* med verdier. Verdiene er en samling *Items* som vi kan hente med indeks, kolonnenavn og annet

mNr	mNavn	mTlf
1	Ola Nordmann	-
2	Knut Hansen	22334455
…	…	…

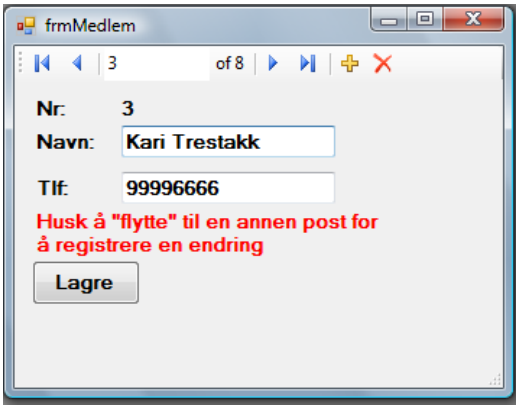
Tabellens samling *Columns* hver med et *ColumnName*

Vi vet jo her hva kolonnene heter (i dag), men det viste blir generisk fordi vi henter kolonnenavnene fra tabellen selv. Hvis noen gjør endringer i databasen, vil denne koden vise de nye kolonnenavnene uten ytterligere vedlikehold. Det gjelder også hvis det legges til eller fjernes kolonner i databasen.

Modellen blir som i forrige eksempel – det er bare griden som mangler.

5. Vise data i tekstbokser og labels

Ovenfor har vi sett hvordan dataene vises i en grid eller en meldingsboks. Vi kan naturligvis også bruke tekstbokser og labler og her er et enkelt eksempel.



Det er laget en *BindingNavigator* i design (kalt *navMedlem*). Den legger til en knapperad øverst, så brukeren kan "bla" mellom postene, skape en ny post og slette en post. Hvis noen av nappene her ikke skal være tilgjengelig, kan knappen enkelt slettes (i design).

Deklarasjoner:

```
Private medlemAdapter As MySqlDataAdapter
Private medlemConnection As MySqlConnection
Private medlemCommand As MySqlCommand
Private bindMedlem As BindingSource
Private tblMedlem As DataTable
```

Hendelsesprosedyre for *Load*:

```
    Private Sub frmMedlem_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        'Skap de nødvendige objektene
        '(BindingNavigator navMedlem er skapt i design)
        medlemConnection = New MySqlConnection _
            ("Server=localhost;" & _
            "Database=baatregister;" & _
            "Uid=Knut;" & _
            "Pwd=Tunk;" & _
            "Connect Timeout=30;" & _
            "CharSet=utf8;")
        medlemConnection.Open()
        medlemCommand = New MySqlCommand _
            ("Select * from tblmedlem", medlemConnection)
        medlemAdapter = New MySqlDataAdapter(medlemCommand)
        tblMedlem = New DataTable()
        bindMedlem = New BindingSource()
        'Fyll tabellen
        tblMedlem.Clear()
        medlemAdapter.Fill(tblMedlem)
        'Knytt opp bindingsource, bindingnavigator og feltene
```

Mår vi skulle fylle et *DataGridView*, knyttet vi grid'en direkte til tabellen ved å angi grid'ens *DataSource*. Når vi skal knytte andre kontroller til tabellen, må vi gå igjenom en *BindingSource*. Vi knytter *BindingSource* til tabellen og én eller flere kontroller til *BindingSource* – se figur neste side. Når det skjer endringer i tabellen, formidles endringen av *BindingSource* til alle kontrollene som er tilknyttet den. Slik kan det se ut:

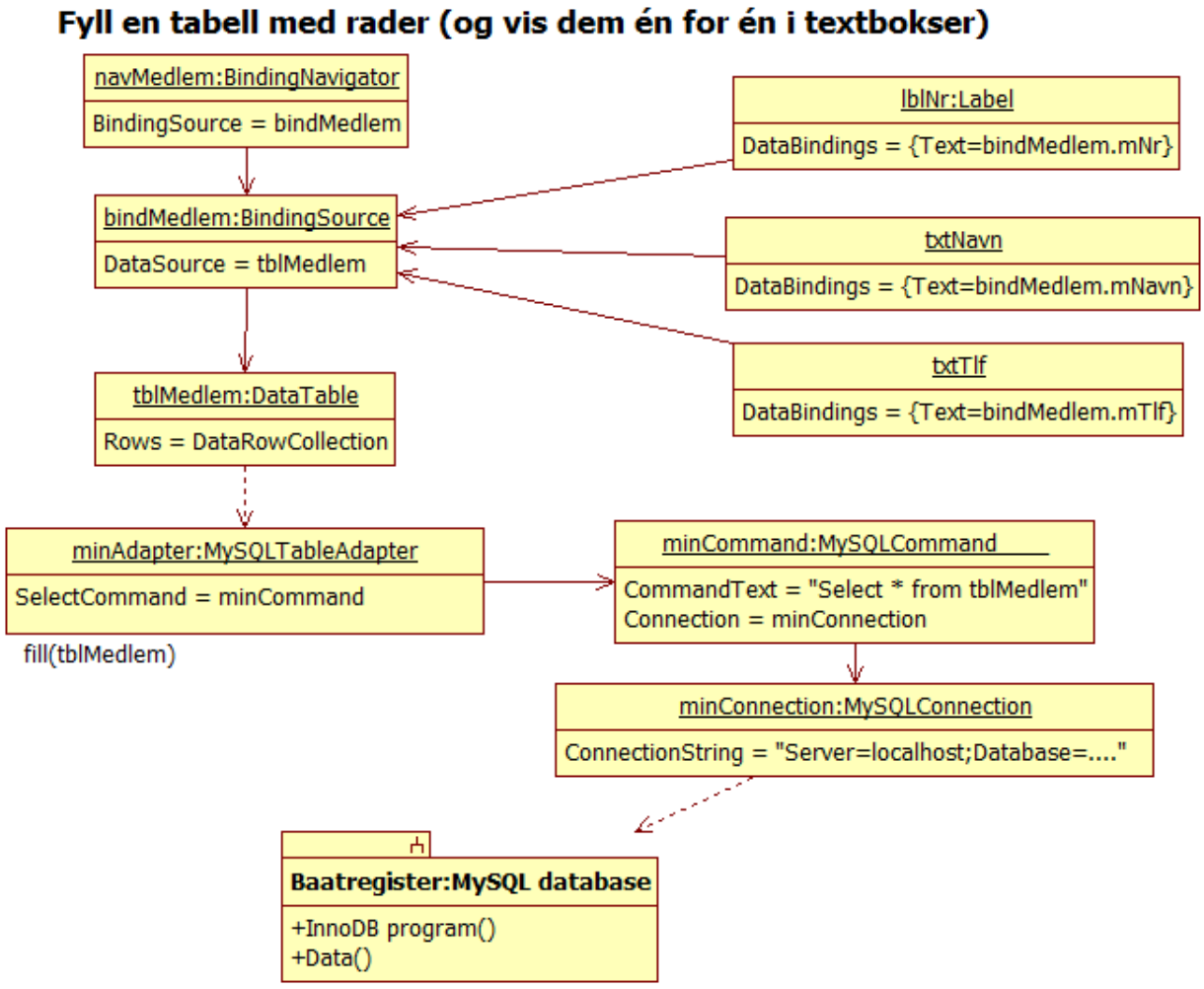
```
        bindMedlem.DataSource = tblMedlem
        navMedlem.BindingSource = bindMedlem
        lblNr.DataBindings.Clear()
        lblNr.DataBindings.Add("Text", bindMedlem, "mNr")
        txtNavn.DataBindings.Clear()
        txtNavn.DataBindings.Add("Text", bindMedlem, "mNavn")
        txtTlf.DataBindings.Clear()
        txtTlf.DataBindings.Add("Text", bindMedlem, "mTlf")
    End Sub
```

Syntaksen for å knytte kontrollene til *BindingSource* er kontrollens *Databindings.Add* med følgende argumenter: (1) egenskapen ved kontrollen som skal bindes, (2) hvilken *BindingSource* som er datakilde og (3) hvilket datamedlem i *BindingSource* som skal bindes.

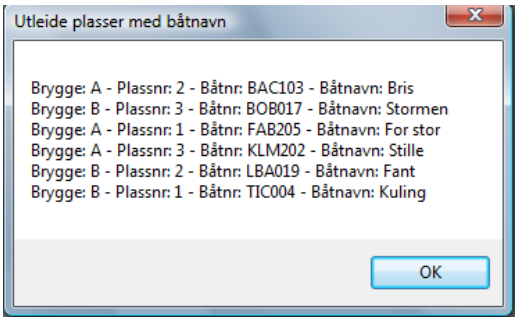
Det meste av denne koden er ellers omtalt ovenfor.

Hvis brukeren får gjøre endringer av dataene, vil endringer i kontrollene formidles av *DataBinding* tilbake til tabellen. Den ligger jo i RAM, så den egentlige MySQL-databasen endres ikke av det. Endringene må evt. aktivt formidles til databasen. Det er omtalt under punkt 9 nedenfor.

Modellen for objektene ser her slik ut:



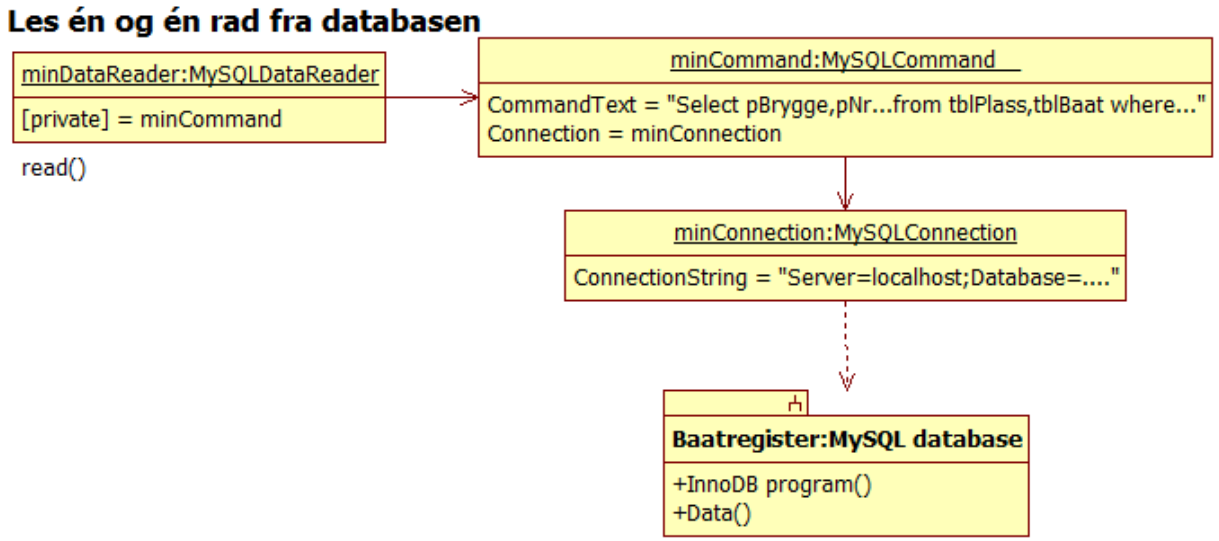
6. Hente data rad for rad med DataReader



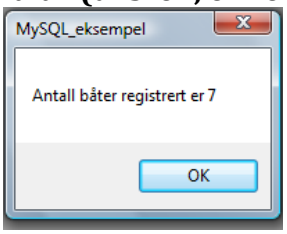
En datareader leser rad for rad fra databasetabellen. (Det likner på det vi gjør med en *StreamReader* som leser en tekstfil linje for linje.) Vi må altså lese i en løkke. Siden vi henter dataene rad for rad, kan vi behandle radene uten å lagre dem alle i en tabell i RAM først. I større databaser, vil det være en fordel, fordi hele tabellen kan ta svært stor plass i RAM og det tar unødvendig lang tid å hente hele tabellen hvis vi kan finne raden vi leter etter tidlig.

```
Private Sub butUtleid_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butUtleid.Click
    'Henter mange rader med en løkke
    Dim utstreng As String = ""

    '1. Sett SQL-tekst
    minCommand.CommandText = "SELECT pBrygge, pNr, pBaatnr, bNavn " _
        & "FROM tblPlass, tblBaat " _
        & "WHERE tblPlass.pBaatnr = tblBaat.bNr;"

    '2. Hent data i løkke
    Try
        '2a. Skap en leser
        minDatareader = minCommand.ExecuteReader() 'returnerer en datareader
        '2b. Les én og én rad fra databasen til slutt
        Do While minDataReader.Read() 'flytter til neste rad - False ved slutt
            '2c. Ta vare på dataene i raden
            utstreng &= "Brygge: " & minDataReader(0).ToString() _
                & " - Plassnr: " & minDataReader(1).ToString() _
                & " - Båtnr: " & minDataReader(2).ToString() _
                & " - Båtnavn: " & minDataReader(3).ToString _
                & vbCrLf
        Loop
        '2c. Lukk leseren (må gjøres her - leseren er lokal i Try
        minDataReader.Close()
    Catch ex As Exception
        MsgBox(ex.Message)
    Finally
        '3. Vis resultatet
        MsgBox(utstreng, , "Utleide plasser med båtnavn")
    End Try
End Sub
```

Legg merke til bruken av *Finally* som blir utført uansett. Etter en evt. feil, får vi da i alle fall se dem som ble hente før feilen oppsto.

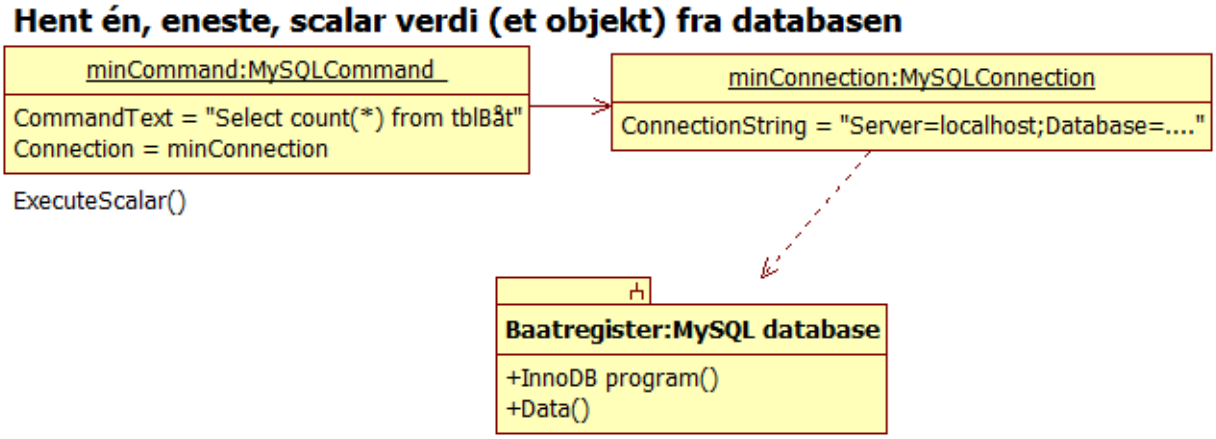


7. Spørringer som returnerer en skalar (dvs. én, enkel verdi)

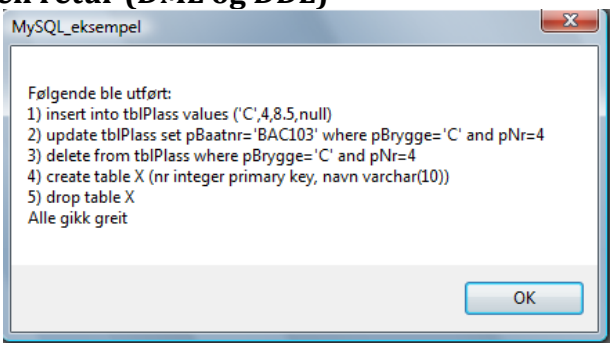


Noen ganger skal man bare ha én, eneste verdi i retur, f.eks. et antall. Da kan man forenkle. Man bruker kommando-objektets metode *ExecuteScalar* som returnerer ett, enkelt *objekt* (altså ikke en streng, slik man kanskje ville trodd).

```
Private Sub butAntBåter_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butAntBåter.Click
    'Henter ett enkeltdata fra databasen med ExecuteScalar
    Dim antall As Object
    '1. Sett SQL-tekst
    minCommand.CommandText = "SELECT count(*) from tblbaat;"
    minCommand.Connection = minConnection
    '2. Hent antall båter (en skalar)
    antall = minCommand.ExecuteScalar() 'OBS! Returnerer et objekt
    MsgBox("Antall båter registrert er " & antall.ToString)
End Sub
```



8. Utføre handlinger uten retur (DML og DDL)

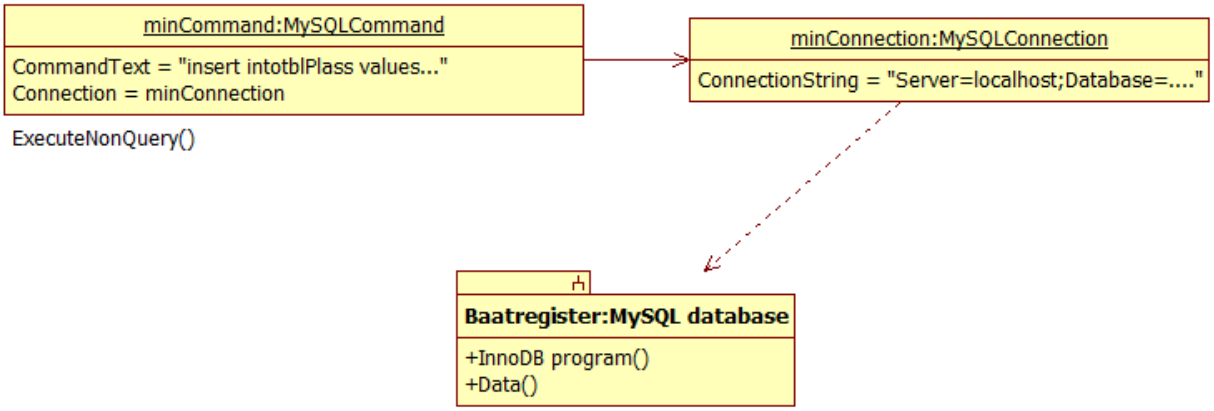


Slike handlinger kan være *update*, *insert*, *delete*, *create* og *drop*. Da brukes metoden *ExecuteNonQuery* som er en prosedyre (og den gir følgelig ingen returverdi):

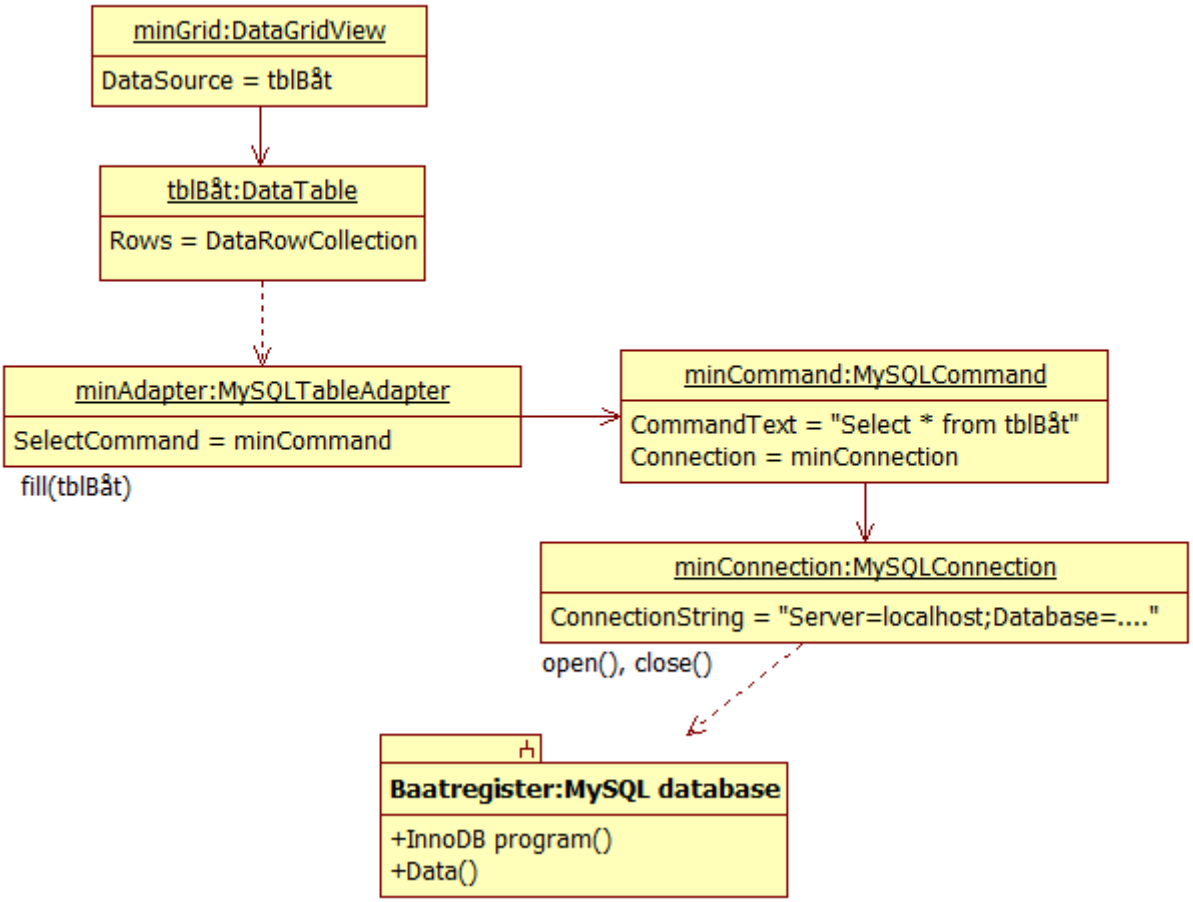
```
    Private Sub mnuPrøve_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuPrøve.Click
        'Prøv ut noen enkle kommandoer
        Dim tekst(5) As String
        Try
            tekst(1) = "insert into tblPlass values ('C',4,8.5,null)"
            tekst(2) = "update tblPlass set pBaatnr='BAC103' " _
                & "where pBrygge='C' and pNr=4"
            tekst(3) = "delete from tblPlass where pBrygge='C' and pNr=4"
            tekst(4) = "create table X (nr integer primary key, navn varchar(10))"
            tekst(5) = "drop table X"
            minCommand.CommandText = tekst(1) 'insert
            minCommand.ExecuteNonQuery()
            minCommand.CommandText = tekst(2) 'update
            minCommand.ExecuteNonQuery()
            minCommand.CommandText = tekst(3) 'delete
            minCommand.ExecuteNonQuery()
            minCommand.CommandText = tekst(4) 'create
            minCommand.ExecuteNonQuery()
            minCommand.CommandText = tekst(5) 'drop
            minCommand.ExecuteNonQuery()
            'Tilbakemelding:
            tekst(0) = "Følgende ble utført: " & vbCrLf
            For i = 1 To 5
                tekst(0) &= i & ") " & tekst(i) & vbCrLf
            Next
            MsgBox(tekst(0) & "Alle gikk greit")
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub
```

EcexuteNonQuery oppdaterer den underliggende databasen direkte.

Utfør handling i databasen uten retur (insert, update, delete, create, drop)



Fyll en tabell med rader (og vis dem i en grid)



9. Oppdatere den underliggende databasen

Hvis man har oppdatert tabellene i RAM, f.eks. ved at brukeren endrer i en datagrid som er knyttet til en tabell med gridens *DataSource*, er endringene *transparente*. Det innebærer at de ikke synes i databasen, og andre brukere vil ikke se dem. De er altså bare gjort i RAM. Man vil jo da gjerne at databasen oppdateres tilsvarende, og det kan man sikre ved avslutning, evt. ved en knapp.

Det er tre måter å gjøre dette på. Jeg anbefaler den første, som jeg har kalt ”Enkel”, men jeg viser også de andre to, da de viser bedre hva som foregår.

Alle endringer av tabellen i RAM gjøres midlertidig. Det vil si at nye rader synes, men tabellen ”vet” at de er nye, slettede rader blir liggende men merkes slettet og synes ikke, mens alle endrede verdier har en ”før” og en ”etter”-verdi. Vi kan gjerne filtrere slik at vi bare ser noen av dem, f.eks. bare de nye eller bare de endrede (filtrering på bare de slettede gir en tom tabell – de synes jo ikke).

9A. ”Enkel oppdatering med god hjelp”
Dette er den vanlige og foretrukne metoden

For å oppdatere må man selvsagt ha kontakt med databasen som før. Videre må man lage en Insert-setning for hver rad som er ny, en Delete-setning for hver rad som er slettet og en Update-setning for hver rad som er endret. Uansett må man ha tabellens primærnøkkel så databasen kan finne raden og sjekke entitets-, referanse- og domeneintegritet. Som forklart nedenfor (9B og 9C), er det ganske tungt å skrive disse setningene selv fordi selve DML-setningen må inneholde data fra tabellen.

Isteden ber vi en *MySqlCommandBuilder* lage alle oppdateringssetningene. Den baserer seg på *Select*-setningen og lager tilsvarende *update*, *delete* og *insert*-setninger. Adapteren må altså først ha satt en lovlig spørring og den må være ”oppdaterbar”[44]. Adapteren utfører oppdateringen – den må følgelig selvsagt ha kontakt med databasen.

```
Private Sub butLagreEnklest_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butLagreEnklest.Click
    minAdapter = New MySqlDataAdapter(minCommand)
    minAdapter.SelectCommand = New MySqlCommand("Select * from tblbaat", minConnection)
    'Skaper en CommandBuilder som bygger de manglende DML-kommandoene
    'for den tilknyttede adapteren. Den bygger på adapterens Select-command
    'som må inneholde pimærnøkkelen.
    Dim minBuilder As New MySqlCommandBuilder(minAdapter)
    minAdapter.Update(tblBåt)
    tblBåt.AcceptChanges()
End Sub
```

Commandbuilder og *adapteren* knytter forbindelse seg imellom og commandbuilder ”lytter” til endringer gjennom adapteren. Hvis man endrer select-setningen, må man bruke commandbuilders metode *RefreshSchema* (her *minBuilder.RefreshSchema*) så det lages nye DML-setninger basert på den nye spørringen.

Når rader i tabellen endres (slettes, legges til eller får ny verdi), merkes raden i tabellen tilsvarende. Ved endrede verdier tas det også vare på tidligere verdi. Tabellens *AcceptChanges* fører til at alle endringer i tabellen gjennomføres permanent. Etter dette finnes ingen slettede, ingen rader er merket ny og ingen ”før”-verdier av felt finnes.

[44] En omfattende beskrivelse av hva som kreves for at den skal være ”oppdaterbar” er gitt i MySQL-manualen http://dev.mysql.com/doc/refman/5.0/en/view-updatability.html. Enkelt sagt vil enkle spørringer med bare én tabell være oppdaterbar – også om den har *where*-klausul. Det blir fort problemer hvis det er brukt aggregering (*sum*, *count* o.l.), nøstede spørringer eller ord som *distinct*, *having*, *group by*. Videre *kan* spørringer med flere tabeller være oppdaterbare, men bare på visse strenge vilkår (bl.a. kun *inner joins*). Du får feilmelding hvis du oppdaterer ulovlig, så det er jo bare å prøvekjøre med feilfelle!

EXTRA: Andre oppdateringsmetoder

9B. "En tyngre oppdatering"[45]
Denne er komplisert og er med her bare for fullstendighetens skyld

Her skal vi skrive DML-setningene selv, men vi gjør det generisk, dvs. uten å kjenne verdien av de enkelte felt.

Først ser vi på update-setningen:

```
    Private Sub butLagreTyngre_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butLagreTyngre.Click
        Try
            '1. Lag nødvendig updatecommand og knytt til adapteren
            Dim updateCommand As New MySqlCommand()
            updateCommand.Connection = minConnection
            updateCommand.CommandText = _
                "UPDATE tblbaat SET bNavn=@bNavn, bLengde=@bLengde, " & _
                    "bEier=@bEier WHERE bNr=@bNr"
            updateCommand.Parameters.Clear() 'i tilfelle det er noen fra før
            updateCommand.Parameters.Add _
                ("@bNr", MySqlDbType.VarChar, 6, "bNr")
            updateCommand.Parameters.Add _
                ("@bNavn", MySqlDbType.VarChar, 30, "bNavn")
            updateCommand.Parameters.Add _
                ("@bLengde", MySqlDbType.Decimal, 10, "bLengde") 'ikke 10,2
            updateCommand.Parameters.Add _
                ("@bEier", MySqlDbType.Int32, 11, "bEier")
            minAdapter.UpdateCommand = updateCommand
```

Parametrene som deklareres er navnet på parameteret (en streng, deretter hvilken datatype den har i databasen, hvor lang den er (antall sifre eller tegn) og hvilken kolonne i tabellen (i RAM) som inneholder denn verdien. Adapteren trenger dette for å gjøre om fra VBs datatyper til databasens datatyper. I databasen er *bLengde* deklarert som *VarChar(10,2)* men her skriver vi bare lengde 10. Det siste totallet gjelder bare hvordan tallet skal formateres – ikke hvordan det lagres.

Insert-setningen følger samme mønster (se programeksemplet), men Delete-setningen blir enklere:

```
            '3. Lag nødvendig deletecommand og knytt den til adapteren
            Dim deleteCommand As New MySqlCommand()
            deleteCommand.Connection = minConnection
            deleteCommand.CommandText = _
                "DELETE FROM tblbaat WHERE bNr=@bNr"
            deleteCommand.Parameters.Clear()
            deleteCommand.Parameters.Add("@bNr", MySqlDbType.VarChar, 6, "bNr")
            'deleteCommand.Parameters("@bNr").SourceVersion _
            '    = DataRowVersion.Original 'hvis man vil være HELT sikker
            minAdapter.DeleteCommand = deleteCommand
```

Her *kan* det være aktuelt å presisere at primærnøkkelen *bNr* skal være i "original" versjon (="før"-verdi), fordi brukeren kan ha endret verdien av den først og deretter slettet den. Jeg synes imidlertid ikke vi bør tillate brukeren å endre primærnøkkelen, og da blir dette unødvendig.

[45] Se evt. forklaring på http://dev.mysql.com/doc/refman/5.0/es/connector-net-examples-mysqldataadapter.html#connector-net-examples-mysqldataadapter-ctor

Når alle tre kommandoene er på plass, er det bare å be adapteren oppdatere og akseptere endringer i tabellen, som før:

```
            '4. Be adapteren om å oppdatere
            minAdapter.Update(tblBåt)
            tblBåt.AcceptChanges()
        Catch ex As Exception
            If ex.InnerException Is Nothing Then
                MsgBox(ex.Message)
            Else
                MsgBox(ex.Message & ex.InnerException.Message)
            End If
        End Try
```

Legg merke til hvordan jeg behandler feil. Hvis det er databasen som returnerer en feil (pga beskrankninger som brytes) vil den følge med som en "indre feil" – en "inner exception". Den vil vi også gjerne se.

9 C. "Den tyngste måten"

Denne er ekstra komplisert og er med her bare for fullstendighetens skyld

Her går vi selv gjennom tabellen, rad for rad, og bygger opp en passende DML-setning som vi så utfører osv.

Først litt housekeeping:

```
    Private Sub butLagreTyngst_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butLagreTyngst.Click
        'Lagrer alle endrede data til databasen
        Dim feil As Boolean = False 'merker seg om det skjer feil underveis
        Dim tmpTabell As New DataTable()
```

Deretter tar vi endringer, insettinger og slettinger hver for seg. Legg merke til at vi bygge opp den riktige oppdaterings- og insettingssetningen før vi kjører *EcecuteNonQuery*. Vi henter alle endringene med et filter, avhengig av om vi vil ha bare endrede (*DataRowState.Modified*) eller bare innsettinger (*DataRowState.Added*).

```
        '1. Endrede rader
        Try
            tmpTabell = tblBåt.GetChanges(DataRowState.Modified)
            If Not (tmpTabell Is Nothing) Then
                For Each rad As DataRow In tmpTabell.Rows
                    minCommand.CommandText = _
                        "Update tblBaat set " _
                        & "bNavn='" & rad.Item(1).ToString & "'," _
                        & "bLengde=" & rad.Item(2).ToString.Replace(",", ".") _
                        & "," & "bEier=" & rad.Item(3).ToString & " " _
                        & "where bNr='" & rad.Item(0).ToString & "';"
                    minCommand.ExecuteNonQuery()
                Next
            End If
        Catch ex As Exception
            MsgBox(ex.Message, , "Feil ved lagring av endrede rader")
            feil = True
        End Try
```

Helt tilsvarende gjør vi for nye rader.

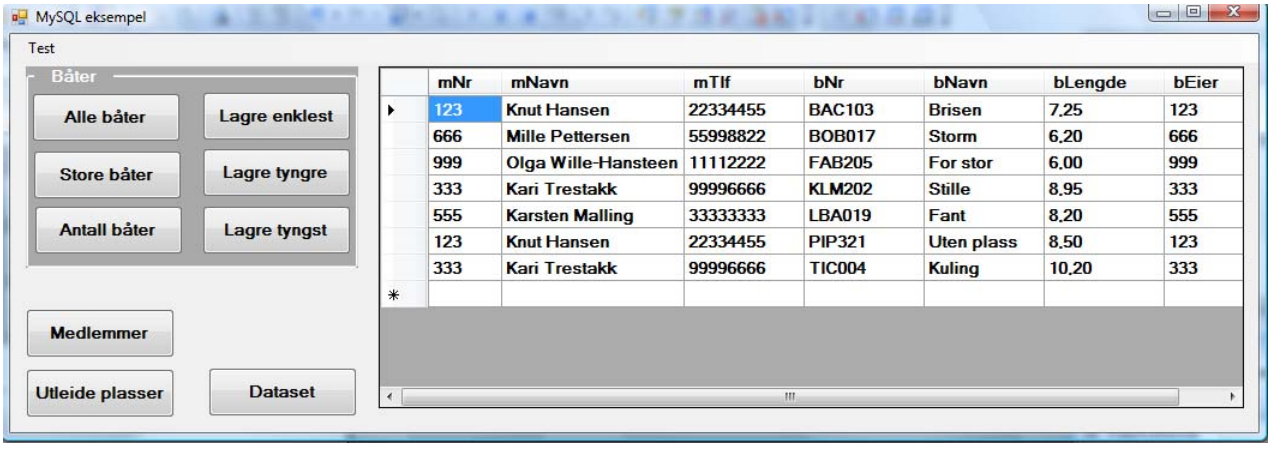
Når det gjelder slettingene, er det verre, fordi de ikke lenger finnes i tabellen i RAM (de er jo slettet!). Vi kan da lage et *View*, der vi henter alle de opprinnelige postene. Dette viewet kan vi så filtrere så vi bare får de som er slettet (*DataViewRowState.Deleted*).

```
        '3. Rader som er slettet
        Try
            'Lager et view av opprinnelig tabell():
            Dim slettetView As New DataView(tblBåt)
            'Ta bare med de radene som er slettet:
            slettetView.RowStateFilter = DataViewRowState.Deleted
            If slettetView.Count > 0 Then
                For Each rad As DataRowView In slettetView
                    minCommand.CommandText = "delete from tblBaat where " _
                        & "bNr='" & rad.Item(0).ToString & "';"
                    minCommand.ExecuteNonQuery()
                Next
            End If
        Catch ex As Exception
            MsgBox(ex.Message, , "Feil ved sletting av rader")
            feil = True
        End Try

        'Endringene er lagret (hvis ikke feil er True)
        If Not feil Then tblBåt.AcceptChanges()
    End Sub
```

10. Bruk av datasett

Brukes bare når det er mange tabeller, så det blir vanskelig å holde orden på dem

Hvis man ønsker det, kan man samle alle tabellene og spørringene man lager i ett datasett. Man oppnår da først og fremst at det blir færre variable (av typen *DataTable*). I eksempelprogrammet er det bare to tabeller, men det kunne jo vært svært mange og vanskelig å holde orden på dem alle sammen. Det blir også enklere å sende alle sammen som ett aktuelt argument til en funksjon/prosedyre.



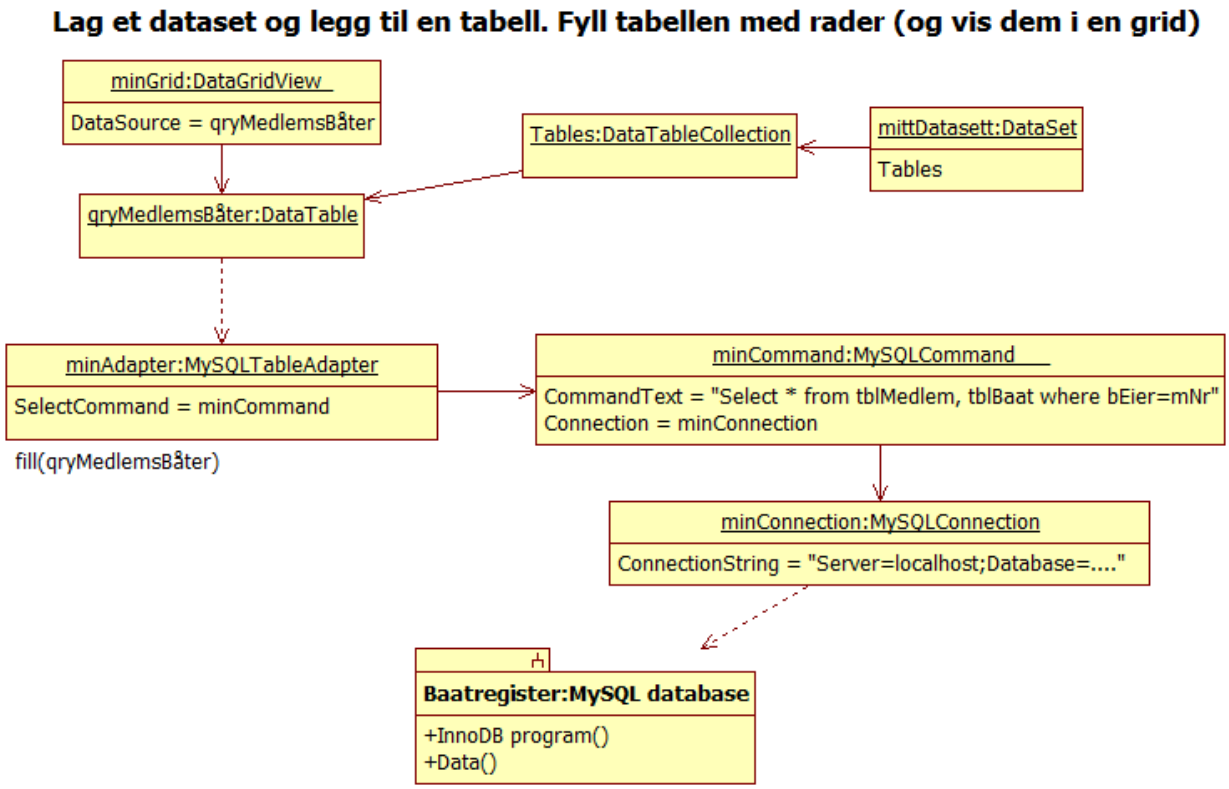
mNr	mNavn	mTlf	bNr	bNavn	bLengde	bEier
123	Knut Hansen	22334455	BAC103	Brisen	7,25	123
666	Mille Pettersen	55998822	BOB017	Storm	6,20	666
999	Olga Wille-Hansteen	11112222	FAB205	For stor	6,00	999
333	Kari Trestakk	99996666	KLM202	Stille	8,95	333
555	Karsten Malling	33333333	LBA019	Fant	8,20	555
123	Knut Hansen	22334455	PIP321	Uten plass	8,50	123
333	Kari Trestakk	99996666	TIC004	Kuling	10,20	333

Videre kan man enklere oppdatere databasen, ved å oppdatere hele datasett på én gang (ikke tabell for tabell).

Her skapes f.eks. en ny tabell basert på en spørring, og tabellen legges til i datasettet:

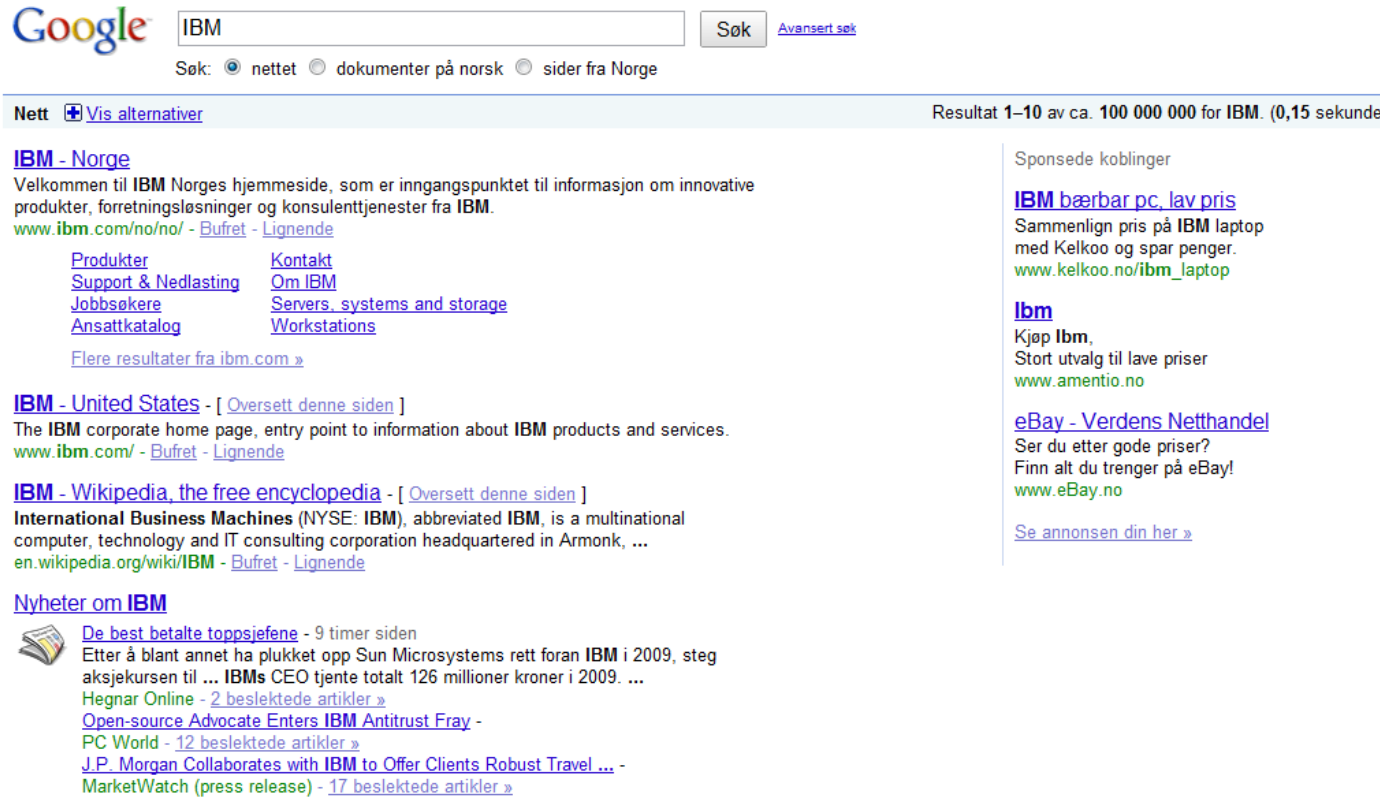
```
Private Sub butDataset_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butDataset.Click
    mittDatasett = New DataSet()
    'Fyller en ny tabell med data og legger den til datasettet
    '1. Skap tabellen:
    Dim qryMedlemsBåter As DataTable = New DataTable("qryMedlemsBåter")
    '2. Skap adapter:
    minAdapter = New MySqlDataAdapter _
        ("Select * from tblMedlem, tblBaat where bEier=mNr", minConnection)
    '3. Fyll data i tabellen:
    minAdapter.Fill(qryMedlemsBåter)
    '4. Legg tabellen til i datasettet
    mittDatasett.Tables.Add(qryMedlemsBåter)
    '5. Vis data fra datasettet
    minGrid.DataSource = Nothing 'tøm grid'en
    minGrid.DataSource = mittDatasett.Tables("qryMedlemsBåter")
End Sub
```

Man kan legge til så mange tabeller man vil i datasettet (*Add*), og refererer senere til dem med navn eller indeks. Tabellen utgjør en samling (*DataTableCollection*) så man kan også bla igjennom dem med *For Each*.



Lese alt til minnet med én gang, eller lese litt og litt?

En analogi



100.000 poster à ca. 250 bytes = 25.000.000.000 (ca. 25 GB). Hvis alt skal i minnet, vil det sannsynligvis skape

a) Svært mye nett-trafikk
b) Kreve svært mye minne og/eller mye ”swapping” til/fra disk (”Page File”)

På den annen side vil det kreve litt ekstra kommunikasjon å hente noen poster av gangen. Men brukerne ser antakelig bare på de første to-tre sidene uansett…

Legg også merke til skalerbarheten her: Søket tok 0,15 sekunder!

Tema K – Tråder, API, dynamiske kontroller, shell og notification area

1. Tråder

Kilde: Utdrag fra Wikipedia

A **thread** in computer science is short for a *thread of execution*. Threads are a way for a program to split itself into two or more simultaneously running tasks. (The name "thread" is by analogy with the way that a number of threads are interwoven to make a piece of fabric). Multiple threads can be executed in parallel on many computer systems. This *multithreading* generally occurs by time slicing (where a single processor switches between different threads) or by multiprocessing (where threads are executed on separate processors). Threads are similar to processes, but differ in the way that they share resources.

Multithreading

Multithreading is a popular programming and execution model that allows multiple threads to exist within the context of a single process, sharing the process' resources but able to execute independently.

…

Operating systems generally implement threads in one of two ways: *preemptive multithreading* or *cooperative multithreading*.

1) Preemptive multithreading is generally considered the superior implementation, as it allows the operating system to determine when a context switch should occur. The disadvantage to preemptive multithreading is that the system may make a context switch at an inappropriate time, causing priority inversion or other bad effects which may be avoided by cooperative multithreading.
2) Cooperative multithreading, on the other hand, relies on the threads themselves to relinquish control once they are at a stopping point. This can create problems if a thread is waiting for a resource to become available.

Treads vs. processes

Threads are distinguished from traditional multi-tasking operating system processes in that processes are typically independent, carry considerable state information, have separate address spaces, and interact only through system-provided inter-process communication mechanisms. Multiple threads, on the other hand, typically share the state information of a single process, and share memory and other resources directly. Context switching between threads in the same process is typically faster than context switching between processes.

VB

I VB kan man skape så mange tråder man vil (begrenset bare av RAM). Hvis det blir flere enn det maksimale antall som er definert, så vil de overskytende settes i kø og startes så snart en annen avslutter. Det maksimale antallet kan settes av programmet – default er 500.

Tråder bruker mye mindre ressurser enn prosesser, blant annet fordi de deler minne med de andre trådene. Dermed går det også raskere å bytte mellom trådene (bl.a. er prosessens minne allerede i RAM).

Hvis programmet lukkes (prosessen stanses), så vil alle trådene stanses også.

Eksempel på tråder

I dette eksemplet har vi et hovedprogram som skaper to, ekstra tråder. Hver tråd åpner sitt eget vindu og viser – ved å telle fra 1 til 8.000 – at de er aktive. De tre vinduene deler prosessoren mellom seg (og evt. andre programmer som OS har gående eller starter).

Hovedprogrammet

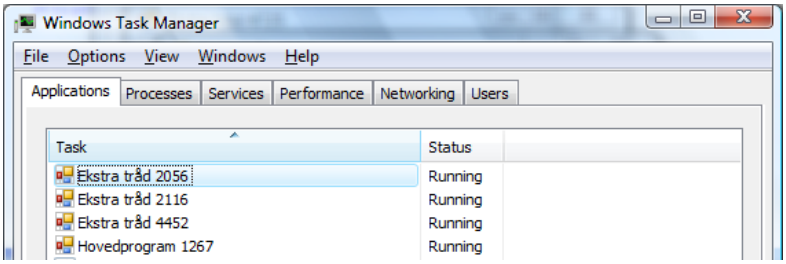
```
Public Class frmTråder
   'Program som demonstrerer tråder
   'Hovedprogrammet starter to ekstra tråder som åpner sine egne vinduer
   'og starter å telle. Hovedprogrammet teller også, og alt skjer tidsdelt
   'Note: Man må bruke "DoEvents", ellers blir programmet
   'for opptatt med tellingen i for-løkken til å vise endringer. Det er ikke enkelt
   'å få tilgang til hovedskjemaet mens telling pågår

   Private Sub startTråd()
      'Prosedyre som kjører asynkront i egen tråd
      frmEnTråd.Show() 'vis nytt vindu for denne tråden
   End Sub

   Private Sub butStarttråder_Click(ByVal sender As System.Object, _
           ByVal e As System.EventArgs) Handles butStarttråder.Click
      'Skap og start tråder
      For j As Integer = 1 To 2 'så mange man vil
         Dim tråd As New Threading.Thread(AddressOf startTråd)
         tråd.Start()
      Next

      'Vis med en teller at også hovedprogrammet er aktivt
      For i As Integer = 0 To 8000
         lblTeller.Text = i.ToString
         Me.Text = "Hovedprogram " & i.ToString
         If i Mod 1000 = 1 Then 'ta en liten pause
            Threading.Thread.Sleep(500)
         End If
         Application.DoEvents()
      Next
   End Sub
End Class
```

Vinduer for trådene.

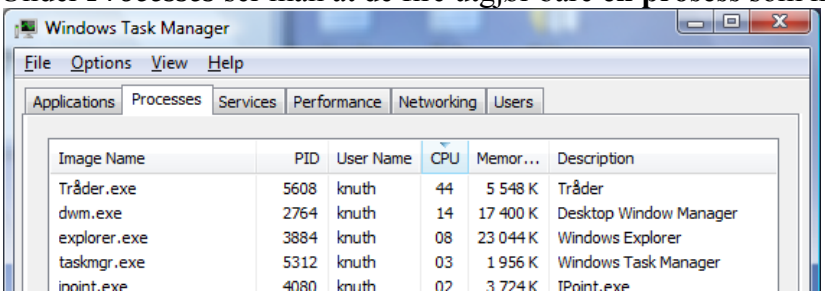
Dette er et eget vindu og altså laget ved å legge til en form.

```
Public Class frmEnTråd
   Private Sub frmEnTråd_Load(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles MyBase.Load
      'Vindu for ekstra tråd
      Me.Show() 'må vise vinduet før telling overtar aktiviteten
      'Vis med en teller at tråden er aktiv tidsdelt
      For i As Integer = 0 To 8000
         lblTeller.Text = i.ToString()
         Me.Text = "Ekstra tråd " & i.ToString
         Application.DoEvents()
      Next
   End Sub
End Class
```

Task Manager viser at det er **flere applikasjoner** som kjører hver for seg:



Under *Processes* ser man at de fire utgjør bare **én prosess** som her har fått 44% av CPU:



2. API

> "An **application programming interface** (**API**) is the interface that a computer system, library or application provides in order to allow requests for service to be made of it by other computer programs, and/or to allow data to be exchanged between them."
>
> (Wikipedia)

API er altså et *grensesnitt* som tilbys av ett program til andre programmer. Hensikten med API er at man skal få tilgang til funksjoner i et program. Det er "usynlig" hvordan funksjonen er implementert (programmert) men funksjonen har en veldefinert virkemåte. Windows API definerer en rekke funksjoner, konstanter og datatyper som tilbys av operativsystemet Windows.

> Windows APIs are dynamic-link libraries (DLLs) that are part of the Windows operating system. You use them to perform tasks when it is difficult to write equivalent procedures of your own.
>
> (Microsoft Visual Basic Help)

Windows operativsystem tilbyr mange API som vi kan bruke i alle programmer. Det er imidlertid mange vanskeligheter involvert. Den viktigste er kanskje at siden API er skrevet "utenfor" VB[46], bruker de andre datatyper og konstanter enn de som er definert i VB. VB kompilatoren har ingen mulighet til å kontrollere at du bruker riktig datatype, og derfor går det ofte galt og krever gjerne en del eksperimentering før det virker.

I VB .NET har man inkludert mange API i programmeringsspråket. F.eks. måtte vi tidligere kalle en API *GetUserName* for å finne hvem som bruker programmet, nå bruker vi heller *My.User.Name*.

[46] Windows O/S er skrevet i C/C++. Den er kompilert til eksekverbar, "native" kode som kan kjøres på bestemte Intel-maskiner. VB 2008 og de andre .NET-språkene derimot, blir kompilert til et felles, mellomliggende språk kalt Common Intermediate Language og som sies å være "managed code". Under kjøring blir dette kompilert igjen "Just In Time" til "native" kode. Fordelen ved et slikt oppsett er at det er forholdsvis enkelt å kunne kjøre programmene i flere maskinmiljøer. Java (og Mac) har for øvrig en tilsvarende strategi (Java kjører i en "virtuell maskin" som igjen gjør kall til det underliggende OS). Problemene oppstår når et program kompilert til "native" kode brukes i ditt program som kompileres til "managed" kode. Det er problemer både med datatyper, måter å overføre argumenter på og annen kommunikasjon mellom de to språkene, samt "garbage collection" og feilhåndtering. Du kan f.eks. oppleve at feil som oppstår i en API ikke "fanges" av Try/Catch og at feilmeldingene ikke inneholder noen nyttig informasjon – VB 2008 "vet" ikke hva som har skjedd.

> Windows API calls were an important part of Visual Basic programming in the past, but are seldom necessary with Visual Basic 2005. Whenever possible, you should use managed code from the .NET Framework to perform tasks, instead of Windows API calls.
>
> (Microsoft Visual Basic Help)

Mest aktuell i dag er de API som vi finner i *Windows API*. Du finner en oversikt i Wikipedia[47].

For å bruke en API, må vi skrive inn *deklarasjonen*[48] i vårt eget program. *Definisjonen* av funksjonen er skjult – den ligger i operativsystemet. Siden API er programmert i C, er deklarasjonen gjort i såkalte "header-filer" med etternavnet ".h". Vi må ofte slå opp i den for å finne verdien av konstanter vi har bruk for.

Deklarasjonene av API-funksjoner i VB begynner med nøkkelordene *Declare Function*. Ofte skal API brukes i flere skjemaer i ditt program, og legges derfor naturlig *Public* i en modul.

Finne en passende API

En oversikt over API er på http://msdn2.microsoft.com/en-us/library/aa383749.aspx

Der kan du velge om du vil se dem alfabetisk, etter kategori eller etter operativsystem ("Release" – hvis du vil programmere for et bestemt O/S). Du finner også en forklaring på de datatypene som er brukt i header-filene.

Når du har funnet en API du vil prøve – det er ikke alltid like lett – må du merke deg hvilken header-fil som er brukt og hvilket bibliotek (.lib). Syntaksen er C/C++ og er litt uvant. Vi ser på den nedenfor.

Det er ofte angitt konstanter som skal brukes og hva de betyr. Konstantene finner du i header-filene. Det enkleste er da å søke med Google. En fin referanse som du da gjerne finner, er Koders.com, som har kopi av de viktigste header-filene.

Delene i Windows API

Microsoft forklarer:

> Differences in the implementation of the programming elements depend on the capabilities of the underlying operating system. These differences are noted in the documentation for the element.
>
> The Windows API consists of the following functional categories:
>
> - Administration and Management
> - Diagnostics
> - Graphics and Multimedia
> - Networking
> - Security
> - System Services
> - Windows User Interface

Eksempel 1 – Diskplass

For å finne diskplass, kan vi naturligvis høyreklikke på en disk og velge *properties*. Her skal vi gjøre det innefra et program, og bruke en API.

[47] http://en.wikipedia.org/wiki/List_of_Microsoft_Windows_application_programming_interfaces

[48] *Deklarasjonen* av en funksjon er signaturen, dvs. den linjen som begynner med *Function* og angir navnet, parametre med rekkefølge og type og funksjonens type. *Definisjonen* er det som står mellom overskriften og *End Function*.

Vi går til den nevnte referansen på nett (msdn2) og velger "Functions by Category". Der velger vi "Disk Management" og finner disse fire:

The following functions are used in disk management.

Function	Description
CreateFile	Creates or opens a file object.
DeleteFile	Deletes an existing file.
GetDiskFreeSpace	Retrieves information about the specified disk, including the amount of free space on the disk.
GetDiskFreeSpaceEx	Retrieves information about the specified disk, including the amount of free space on the disk.

De to siste ser jo lovende ut, men den første viser seg bare kunne håndtere disker opptil 2 GB (den henger nok igjen fra tidligere O/S). Vi må altså bruke *GetDiskFreeSpaceEx*. Den er forklart slik:

> The **GetDiskFreeSpaceEx** function retrieves information about the amount of space that is available on a disk volume, which is the total amount of space, the total amount of free space, and the total amount of free space available to the user that is associated with the calling thread.

Det ser jo lovende ut. Vi klikker på den, og får nærmere detaljer. Vi merker oss da at den er deklarert slik:

```
BOOL WINAPI GetDiskFreeSpaceEx(
  __in_opt   LPCTSTR lpDirectoryName,
  __out_opt  PULARGE_INTEGER lpFreeBytesAvailable,
  __out_opt  PULARGE_INTEGER lpTotalNumberOfBytes,
  __out_opt  PULARGE_INTEGER lpTotalNumberOfFreeBytes
);
```

Parametre

Denne deklarasjonen er skrevet i C/C++. Der står datatypen foran variabelnavnet/funksjonsnavnet. Vi må altså "oversette" den for å kunne bruke den.

Under forklaringen av datatyper finner vi at

- ✓ BOOL = A Boolean variable (should be TRUE or FALSE). I VB er dette Boolean.
- ✓ LPCTSTR = Pointer to a constant null-terminated string of 8-bit Windows (ANSI) characters. I VB er dette en String-variabel.
- ✓ PULARGE_INTEGER er et 64-bits heltall. I VB er dette en Int64. Forbokstaven P angir at det er en peker (pointer) til en ULARGE_INTEGER. Det innebærer at det må være en variabel (variabelnavnet er pekeren).

Parametrene er også forklart, slik:

lpDirectoryName *[in, optional]*

A directory on the disk.
If this parameter is NULL, the function uses the root of the current disk.
If this parameter is a UNC name, it must include a trailing backslash, for example, "\\MyServer\MyShare\".
This parameter does not have to specify the root directory on a disk. The function accepts any directory on a disk.
The calling application must have FILE_LIST_DIRECTORY access rights for this directory.

lpFreeBytesAvailable *[out, optional]*

A pointer to a variable that receives the total number of free bytes on a disk that are available to the user who is associated with the calling thread.
This parameter can be NULL.
If per-user quotas are being used, this value may be less than the total number of free bytes on a disk.

lpTotalNumberOfBytes *[out, optional]*

A pointer to a variable that receives the total number of bytes on a disk that are available to the user who is associated with the calling thread.
This parameter can be NULL.

If per-user quotas are being used, this value may be less than the total number of bytes on a disk.
To determine the total number of bytes on a disk or volume, use ***IOCTL_DISK_GET_LENGTH_INFO****.*

lpTotalNumberOfFreeBytes [out, optional]
A pointer to a variable that receives the total number of free bytes on a disk.
This parameter can be NULL.

Legg merke til om parameteret er merket [in], [out] eller [in/out]. Parametre som er [in] er de som bare brukes av funksjonen. Hvis de også er merket [optional] er de frivillige. De kan være konstanter eller variabel og overføres *ByVal*. De som er merket [out] eller [in/out] vil ha ny verdi når funksjonen returnerer. De må altså være variabel og overføres *ByRef*.

Returverdien er angitt slik:

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero (0). To get extended error information, call ***GetLastError****.*

Det kan se rart ut at returverdien er 0 eller forskjellig fra 0, når den er deklarert som Boolean. Det har å gjøre med C/C++ som bruker 0=False, ikke-0=True. Det samme gjør VB, så vi kan bruke en vanlig Boolean i VB.

Kommentarene er slik:

The values obtained by this function are of the type ***ULARGE_INTEGER****. Do not truncate these values to 32 bits.*
The ***GetDiskFreeSpaceEx*** *function returns zero (0) for lpTotalNumberOfFreeBytes and lpFreeBytesAvailable for all CD requests unless the disk is an unwritten CD in a CD-RW drive.*
Symbolic link behavior—If the path points to a symbolic link, the operation is performed on the target.

Nederst på hjelpesiden står det angitt hvilke O/S-klienter og servere denne virker i – sjekke at den dekker behovet. Du finner angitt dll-biblioteket – her *Kernel32.dll* og det må du merke deg.

Nå vet vi nok til å skrive koden.

Først må vi deklarere denne funksjonen i programmet vårt (på skjema-nivå eller i en modul):

```
Declare Auto Function diskplass Lib "Kernel32.dll" Alias "GetDiskFreeSpaceEx" ( _
    ByVal lpDirectoryName As String, _
    ByRef lpFreeBytesAvailable As Int64, _
    ByRef lpTotalNumberOfBytes As Int64, _
    ByRef lpTotalNumberOfFreeBytes As Int64) _
    As Boolean
```

- ✓ *Auto* betyr at VB-kompilatoren skal forsøke å ordne eventuelle problemer med String
- ✓ *diskplass* er det navnet jeg selv har valgt for denne funksjonen i dette programmet
- ✓ *Lib* angir hvilket bibliotek denne funksjonen finnes i
- ✓ *Alias* angir hva funksjonen faktisk heter i biblioteket
- ✓ *Parametrene* følger nøye den definisjonen jeg fant for denne API. Som du ser, har jeg oversatt BOOL til *Boolean*, LPCTSTR til *String* og PULARGE_INTEGER til *Int64*.
- ✓ Legg merke til at det er brukt *ByVal* og *ByRef* helt bevisst – *ByVal* for [in] og *ByRef* for [out og in/out].

Hvis vi ikke vil finne på eget funksjonsnavn, kan vi bruke API-navnet direkte, uten Alias:

```
    Declare Auto Function GetDiskFreeSpaceEx Lib "Kernel32.dll" (_
        ByVal lpDirectoryName As String, _
        ByRef lpFreeBytesAvailable As Int64, _
        ByRef lpTotalNumberOfBytes As Int64, _
        ByRef lpTotalNumberOfFreeBytes As Int64) _
    As Boolean
```

Nå er alt klart for å bruke den deklarerte API-funksjonen, f.eks. i en prosedyre. Som vanlig kan vi bruke våre egne variabelnavn som aktuelle argumenter, uansett hva de heter i API-funksjonen:

```
  Private Sub butBeregn_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles butBeregn.Click
    Dim OK As Boolean
    Dim disk As String
    Dim LedigBruker, TotalBytes, LedigBytes, TotalForBruker As Int64
    Const Giga As Int64 = 1073741824 '=2^30

    For disknr As Integer = 65 To 90 '=A til Z (løkkevariabel må være tall)
      disk = CStr(Chr(disknr)) & ":"
      OK = diskplass(disk, LedigBruker, TotalBytes, LedigBytes)
      If Not OK Then
        txtPlass.Text &= "Disk " & disk & " Ikke tilkoblet" & vbCrLf & vbCrLf
      Else
        TotalForBruker += LedigBruker
        txtPlass.Text &= "Disk " & disk & vbCrLf
        txtPlass.Text &= "Brukte bytes:      " & Format((TotalBytes - LedigBytes) / Giga, "Standard") & " GB" &
vbCrLf
        txtPlass.Text &= "Ledige bytes:      " & Format(LedigBytes / Giga, "Standard") & " GB" & vbCrLf
        txtPlass.Text &= "Total diskplass:   " & Format(TotalBytes / Giga, "Standard") & " GB" & vbCrLf
        txtPlass.Text &= "(Ledig for deg:    " & Format(LedigBruker / Giga, "Standard") & " GB)" & vbCrLf
        txtPlass.Text &= vbCrLf
      End If
    Next
    txtPlass.Text &= "Totalt ledig for deg: " & Format(TotalForBruker / Giga, "Standard") & " GB"
    txtPlass.Select(0, 0) 'Fjerner merking av all teksten
  End Sub
```

Utskriften blir ikke særlig vakker. Derfor har jeg formattert og viser svaret i Gigabytes. (Tallet 1073741824 er 2^{30} = 1 GB.)

Eksempel 2 – En knapp som kan flyttes av brukeren

Her skal vi la brukeren flytte på en knapp. Det kan vi oppnå ved å sende en *melding* til knappen. Vi bruker API-funksjonen *SendMessage*, deklarert slik under *Messages and Message Queues*:

```
LRESULT WINAPI SendMessage(
  __in  HWND hWnd,
  __in  UINT Msg,
  __in  WPARAM wParam,
  __in  LPARAM lParam
);
```

Av VB-koden min her, ser du hvilke datatyper som er involvert – alle parametre er [in].

```
Private Declare Auto Function SendMessage Lib "user32" _
        Alias "SendMessage" ( _
        ByVal hWnd As System.IntPtr, _
        ByVal wMsg As Int32, _
        ByVal wParam As Int32, _
        ByVal lParam As Int32) _
    As Long
```

```
Private Const WM_NCLBUTTONDOWN As Long = &HA1&[49]
Private Const HTCAPTION As Long = 2&
```

✓ WM_NCLBUTTONDOWN = Melding: Venstre musknapp er nede, inne i objektet
✓ HTCAPTION = Nærmere presisering: Musen er flyttet

Vi trenger også en API *ReleaseCapture*:

```
Private Declare Auto Function ReleaseCapture Lib "user32" () As Long
```

Slik bruker vi disse for knappen *butKanFlyttes*:

```
Private Sub butKanFLyttes_MouseDown(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.MouseEventArgs) _
        Handles butKanFLyttes.MouseDown
    ReleaseCapture()
    SendMessage(butKanFLyttes.Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0&)
End Sub
```

Virkningen er at når brukeren holder nede venstre musetast over knappen, kan den dras omkring i skjemaet og settes der brukeren ønsker. Det er en ulempe med dette, og det er at vi ødelegger virkningen av vanlig klikk. Det kan vi selvsagt gjøre noe med, men det tar jeg ikke opp her.

3. Dynamiske kontroller

Vanligvis vet vi hvilke og hvor mange kontroller vi trenger, og vi legger dem inn i skjemaet i designmodus. Noen ganger vet vi det ikke, fordi antallet knapper er avhengig av noe som skjer mens programmet kjører. Da må vi legge til kontrollene mens programmet eksekverer.

Legge til en knapp

Anta at vi skal legge til kommandoknapper (Buttons). Da skaper vi først en ny knapp, setter de nødvendige egenskapene for den og legger den til på skjemaet:

```
Private Sub butNytekst_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles butNytekst.Click
  'Legger til en ny tekstboks
  '1. Skap tekstboksen
  Dim nyTxtbox As New TextBox()
  '2. Sette egenskaper
  With nyTxtbox
    .Top = 28
    .Left = 250
    .Width = 150
    .Height = 20
    .BackColor = Color.Blue
    .ForeColor = Color.White
  End With
  '3. Legg til en handling for tekstboksen
  AddHandler nyTxtbox.Click, AddressOf GjørNoe
  '4. Legg tekstboksen til i skjemaet
  Me.Controls.Add(nyTxtbox)
  nyTxtbox.Focus()
End Sub
```

Legg merke til at skjemaet (”Me”) har en *samling* av kontroller kalt *Controls* med metoden *Add(kontroll)*. Her er *KnappToolTip* en *ToolTipControll* på skjemaet.

Legge til handling for en dynamisk kontroll

Akkurat når det gjelder knapper, så vil du jo gjerne at den skal utføre noe – du skal knytte en handling til klikk-hendelsen. Det går ikke å lage hendelsesprosedyrer dynamisk, så vi må lage en

[49] Ved å skrive et tegn bak tallet, angir vi datatypen: I eller %=Integer, D eller @=Decimal, L eller &=Long, R eller #=Double, S =Short. Ved å skrive &H foran tallet, angir du at tallet skal tolkes heksadesimalt. "&HA1&" er altså en heksadesimal long.

prosedyre/funksjon som gjør noe for alle de nye knappene, og så knytte klikkhendelsen for den nye knappen til den. Det er det som gjøres i punkt 3 ovenfor.

```
'3. Legg til en handling for tekstboksen
AddHandler nyTxtbox.Click, AddressOf GjørNoe
```

Da må det finnes en prosedyre *GjørNoe* med nøyaktig de samme parametrene som Click-hendelsen for tekstbokser. *GjørNoe* opptrer som et alias for *nyTxtbox.Click* og overtar for den.

```
Private Sub GjørNoe(ByVal sender As Object, ByVal e As System.EventArgs)
   MsgBox("Du klikket?")
End Sub
```

Legge knappen i et panel

Vi må bestemme hvor knappen skal stå, og hvis vi etterhvert legger til flere, må vi beregne egenskapene *Top* og *Left*. Det medfører jo en del programmering (og tenkearbeid!). Da er det enklere å lage en *containerkontroll* på skjemaet, og så legge knappen inne i den. Hvis vi bruker en *TableLyoutPanel* kan du bestemme hvor mange celler det skal være i tabellen, og bestemme i hvilken celle den nye kontrollen skal havne. Det gir full kontroll, men du må selv holde orden på hvilke celler som er ledige. Enklere er det å bruke en *FlowLayoutPanel*. Der fylles det automatisk på fra venstre mot høyre, og velger ny linje ved behov (den kan settes til å fylle på annerledes også). Du kan sette *AutoSize* til *True*, så vil den også utvide seg mot høyre ved behov (pass på – den kan bli større enn skjemaet, så da bør også skjemaet ha *AutoSize = True*). Også *FlowLayoutPanel* har en samling kontroller kalt *Controls*.

Anta nå at vi har laget en slik *FlowLayouPanel* kalt *panelTekster*. (Det blir det samme om det er gjort under design eller lagt til skjemaet dynamisk.) Da blir programmering enda litt enklere, siden vi slipper å tenke på hvor knappen skal plasseres. Man lager en ny tekstboks på samme måten som ovenfor, men istedenfor å legge den på skjemaet med *Me.Controls.Add(nyTxtbox)* legges den til panelet med

```
flowTekster.Controls.Add(nyTxtBox)
```

Hvilken kontroll ble trykket?

Hvis kontrollene som er lagt inn, skal utføre helt forskjellige ting, må de ha hver sin prosedyre som alias, og dem må du lage i designmodus. Da kan du egentlig like godt lage knappene i designmodus, skjule dem, og vise dem ved behov.

Hvis de derimot skal gjøre omtrent det samme, men med mindre avvik, kan du bruke samme prosedyre for alle, men du må finne ut hvilken kontroll som faktisk ble klikket, så koden kan ta høyde for det. Et parameter for *GjørNoe* er *Sender* som forteller hvilken knapp som sendte klikkhendelsen. Vi kan altså gå igjennom panelets kontroller til vi finner den samme:

```
Sub GjørNoe(ByVal sender As System.Object, _
            ByVal e As System.EventArgs)
    For i As Integer = 0 To panelKnapp.Controls.Count - 1
        If sender.Equals(panelKnapp.Controls(i)) Then
            MsgBox("Du klikket knapp nr " & i.ToString)
        End If
    Next
End Sub
```

Legg merke til måten vi sammenlikner de to kontrollene på – de er objekter og da kan vi ikke sammenlikne med likhetstegn. (Operatoren likhetstegn er ikke definert for objekter. Når vi har hatt bruk for det i tidligere oppgaver, har vi selv definert operatoren som en del av objektets metoder.)

Legg bilde på knapper

Knapper skal ofte ha et bilde istedenfor tekst. Det bestemmer vi med egenskapen *Image*. Den godtar ikke ikoner – bare bitmap bilder (jpg, png, gif o.l.).

```
Dim nyknapp As New Button()
nyknapp.AutoSize = True
nyknapp.AutoSizeMode = Windows.Forms.AutoSizeMode.GrowAndShrink
nyknapp.Image = Image.FromFile("HiBu_liten.gif")
```

Legg merke til at vi ikke kan skape et nytt *Image* med *New* – det er en abstrakt klasse.

Hvis vi nå vil hente et *ikon* til knappen, må vi først skape et *Icon*-objekt og deretter konvertere ikonet til et bilde:

```
nyknapp.Image = New Icon("HiBu.ico").ToBitmap()
```

En spesielt interessant mulighet er å hente ikonet ut av en exe-fil:

```
nyknapp.Image = _
    Icon.ExtractAssociatedIcon("c:\Windows\explorer.exe").ToBitmap()
```

Mange *exe*-filer har *flere* ikoner, og da vil denne hente den *første* av dem.

Lage egne kontroller

Vi kan lage våre egne kontroller basert på de eksisterende, gjennom arv. Da sier vi egentlig at vi vil ha en slik eller slik kontroll, **men med noe ekstra**. Hvis vi f.eks. vil lage en tekstboks med en ekstra egenskap *hasChanged*, kan vi gjøre det slik:

```
Public Class minTekstboks
  Inherits TextBox
  'Min teksboks har ekstra attributt
  Public hasChanged As Boolean = False 'True hvis teksten har vært endret
End Class
```

Legg merke til at klassen er merket *Inherits* hvilket betyr at den har alle egenskapene, hendelsene osv. som en vanlig *TextBox* har, men i tillegg har den altså en Boolsk egenskap.

Da gjenstår bare å legge denne nye typen tekstboks til i skjemaet. Når teksten i tekstboksen endres, setter vi ny verdi for egenskapen *hasChanged*:

```
Private Sub changed(ByVal sender As System.Object, ByVal e As System.EventArgs)
  CType(sender, minTekstboks).hasChanged = True
End Sub
```

Egenskapen *hasChanged* vil da alltid vise om teksten er blitt endret på et teller annet tidspunkt – også om den er satt tilbake til en tom streng.

4. Starte andre programmer i egne prosesser

Det er to måter å starte prosesser på programmatisk: *Shell* og *Process*.

Shell er fra Visual Basic 6.0 (den støttes ikke lenger, men virker ennå) og starter en ny prosess, altså et annet program. Populært kalles dette å ”shelle”, fordi vi bruker kommandoen *Shell*. Den vil ha argument (streng) som forteller hvilket program som skal åpnes, og du bør bestemme hvordan programmet skal vises (minimert, normal eller maksimert). Syntaksen er

```
Public Function Shell( _
   ByVal PathName As String, _
   Optional ByVal Style As AppWinStyle = AppWinStyle.MinimizedFocus, _
   Optional ByVal Wait As Boolean = False, _
   Optional ByVal Timeout As Integer = -1 _
) As Integer
```

Som det fremgår, er det bare *PathName* som er obligatorisk. Parameteret *Wait* er Boolean og angir om hovedapplikasjonen skal vente til prosessen avsluttes eller ikke. Eventuell argumenter til prosessen, må angis i *PathName* etter same syntaks som i en kommandolinje (som i *Run* fra startmenyen), f.eks.

```
Private Sub butInternett_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butInternett.Click
    'Starter internet explorer og viser knuts hjemmeside med Shell
    Dim ID As Integer = Shell("""C:\Program Files\Internet Explorer\iexplore.exe""" _
          & " " & "http://www.knut.hansson.name", _
          AppWinStyle.NormalFocus, False)
End Sub
```

Shell må inneholde et progamnavn – du kan *ikke* åpne en fil. Microsoft anbefaler at programnavnet oppgis med anførselstegn og det skal alltid være mellomrom mellom programnavnet og argumentene.

Den andre metoden, som er en del av .NET, bruker ***Process***. Det er et objekt med mange medlemmer, hvorav én er *Start*. Start har mange forskjellige syntakser, her er en av dem:

```
Public Shared Function Start ( _
  fileName As String, _
  arguments As String _
) As Process
```

Her behøver man altså ikke vite hvordan programmet skal ha argumentene – de oppgis som eget argument.

Eksempel:

```
Private Sub butInternett2_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butInternett2.Click
    'Starter internet explorer og viser knuts hjemmeside med Process.Start
    Dim prosess As Process = Process.Start("""C:\Program Files\Internet Explorer\iexplore.exe""", _
          "http://www.knut.hansson.name") 'returnerer et process-objekt
End Sub
```

Andre egenskaper og metoder til Process gjør det mulig å stoppe/lukke, sette prioritet osv.

5. Notification Area

Linjen nederst på skjermen, kalles *TaskBar*. Til høyre på den finnes *Notification Area*. Programmer som jobber i bakgrunnen (viruskontroll, printerkontroll, lydkontroll, MSN, klokke osv.) og vanligvis ikke viser vindu, har gjerne et ikon der. Slik lager du et slikt ikon i *TaskBar*:

Ikon i Notification Area

Lag et vanlig program, basert på vindu. Deretter gjør du følgende:

- ✓ Legg til en *ContextMenuStrip* og tast inn minst et valg på menyen. Lag kode for klikk-hendelsen(e).
- ✓ Legg til en *NotifyIcon*. Sett følgende egenskaper på den:
  - o **Icon**, dvs. det ikonet som skal vises i *TaskBar*
  - o **Text**, dvs. teksten som vises i *ToolTips* når musen peker mot ikonet
  - o **Visible** må være *True*

Nå vil et ikon vises i *Notification Area* når programmet startes. Dessverre har det en lei tendens til å bli igjen etter at programmet er avsluttet, men det forsvinner hvis brukeren beveger musen over det. (Det finnes måter å avhjelpe dette, men det krever kall til API og tas ikke her.)

BalloonTip

Når programmet skjules, f.eks. ved at brukeren klikker på en knapp, kan du vise en ”ballong” ved ikonet i *Notification Area*. Da setter du først følgende egenskaper for NotifyIcon:

- ✓ **BaloonTipIcon** (*ToolTipIcon.Error*, *.Info*, *.None* eller *.Warning*)
- ✓ **BaloonTipText**
- ✓ **BaloonTipTitle**

For å vise "ballongen", kaller du *NotifyIcon.ShowBaloonTip(<tid>)*. Her er *<tid>* den tiden ballongen skal vises i millisekunder. Operativsystemet har likevel et minimum og maksimum, og hvis du angir <tid> utenfor dette området, blir tiden satt til minimum/maksimum likevel. Bare tid brukeren er aktiv med PC-en teller. Brukeren kan alltid selv lukke "ballongen".

Eksempel

Dette programmet har en *ContextMenuStrip* med to valg, kalt *mnuÅpne* og *mnuAvslutt*. Videre har det en *NotifyIcon* med ovennevnte egenskaper satt.

```
Public Class frmNotifycationDemo
    Private Sub mnuAvslutt_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuAvslutt.Click
        End
    End Sub

    Private Sub mnuÅpne_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuÅpne.Click
        Me.Show()
        Me.WindowState = FormWindowState.Normal 'kan ellers bli minimert
    End Sub

    Private Sub frmTaskTrayDemo_Shown(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Shown
        Me.Hide()
        NotifyIcon.ShowBalloonTip(10000) 'viser ballong i 10 sekunder
    End Sub
End Class
```

Skjule vinduet igjen etter at det er åpnet

Programmet ovenfor vil skjule seg og vise ikon i *Notification Area* når det startes. Når brukeren har bedt om å få det åpnet, vil det vises som vindu. Brukeren vil deretter kanskje skjule det igjen. Til det kan vi naturligvis lage en enkel knapp *butSkjul* med koden *Me.Hide()*. Mer elegant vil det være å skjule vinduet når brukeren klikker minimérknappen. Da kunne vi ønske oss en hendelse for skjemaet av typen *WindowMinimized* eller kanskje *WindowstateChanged*, men ingen av dem finnes.

Isteden kan vi bruke hendelsen *Resize*. Den vil også slå til når skjemaet minimeres, og da kan vi sjekke *Windowstate*. Hvis den er *Minimized* kan vi skjule vinduet:

```
    Private Sub frmTaskTrayDemo_Resize(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Resize
        'Skjuler vinduet når det minimeres
        If Me.WindowState = FormWindowState.Minimized Then Me.Hide()
    End Sub
```

Tema L1 – Testing – V12

Svært mye kan sies om testing – her tar vi bare en liten "smakebit".

Bevisbart riktig kode?

Det er gjort mye, teoretisk forskningsarbeid omkring det å bevise at kode er korrekt. Dessverre viser det seg at det ikke er mulig å bevise at koden er korrekt unntatt for trivielle kodemoduler. Problemet er begrepet "korrekt". Vi oppfatter kode som korrekt hvis den gjør det den skal, dvs. følger spesifikasjonen. Men hvordan vet vi at spesifikasjonen er korrekt. (Evt. kan spesifikasjonen sjekkes mot en overordnet, mer abstrakt spesifikasjon, men er den korrekt da? Og så videre. Ingen spesifikasjon kan bevises mot seg selv.) Dessuten har vi "the halting problem": Alan Touring viste allerede i 1936 at hvis vi har et program og en bestemt, endelig sekvens av input – så kan vi *ikke* alltid bevise at programmet til slutt vil stoppe.

Det er allikevel mulig å bevise at enkle kodemoduler er korrekte i forhold til kjente spesifikasjoner (f.eks. sortering) og da kan man komme et godt stykke på vei ved å dele opp programmet i små, enkle deler.

I praksis er vi allikevel nødt til å sjekke programmoduler med *eksempler*. Jo flere eksempler som gir korrekt output, desto mer sannsynlig er det at koden er korrekt. Det er f.eks. grunnen til at hvis Microsoft har laget en *Sort* som del av programmeringsspråket (f.eks. Array.Sort(array)) så bør man absolutt bruke den fremfor å skrive sin egen. Microsofts *Sort* er jo testet et utall ganger gjennom bruk i svært mange forskjellige programmer.

Blackbox test

I *blackbox* testing, tester vi en modul uten å kjenne detaljene i implementering.

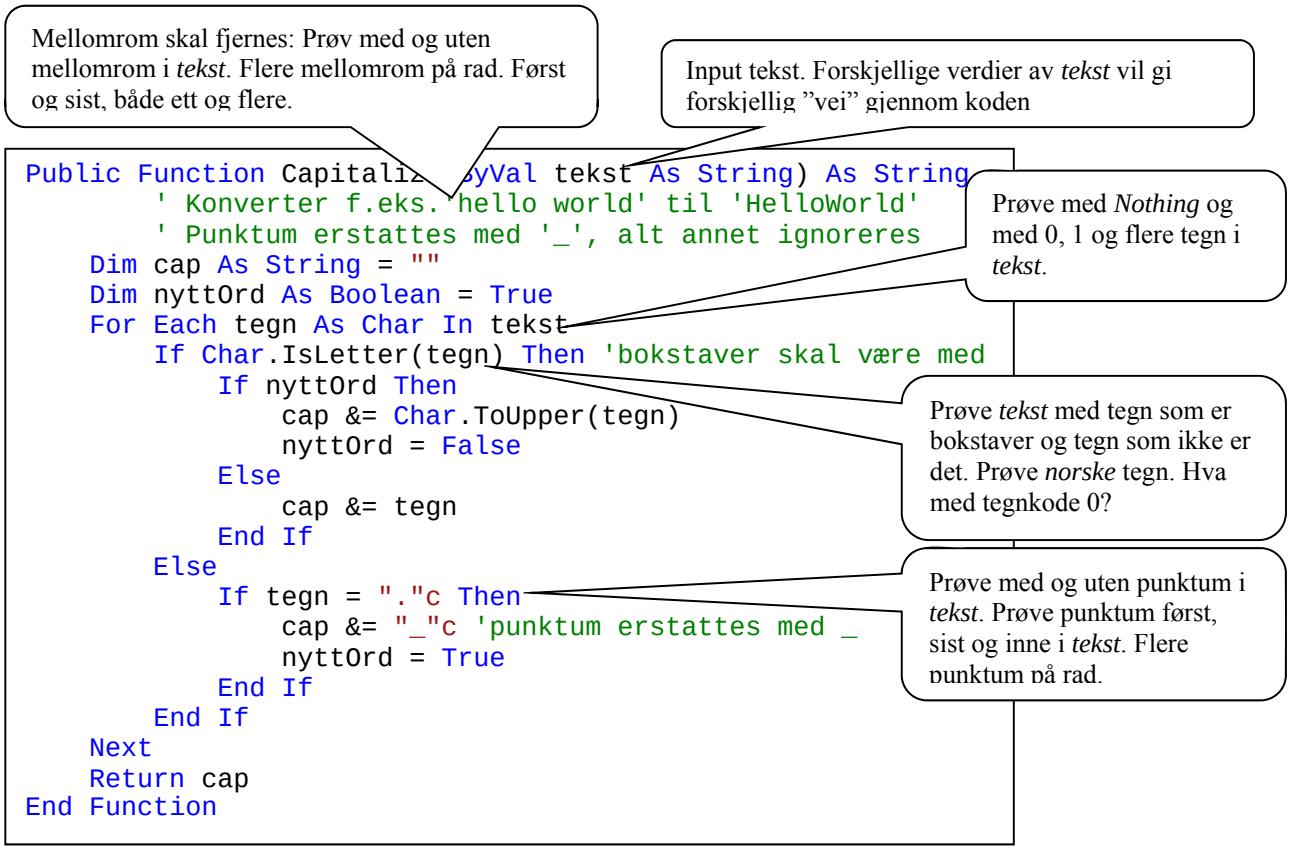
Vi lager input i form av *test cases*, kjører dem gjennom modulen og tar vare på output. Vi sammenlikner så input med output og ser etter output som ikke stemmer med spesifikasjonene.

Hovedproblemet er (a) å finne alle fornuftige tilfeller av input og (b) å vite nøyaktig hvilken output denne inputen skal gi. Ideelt sett, skal jo input dekke all kode i modulen og alle kombinasjoner av veier gjennom koden. I praksis er dette ikke mulig, ettersom vi her ikke kjenner koden. Det kan altså godt hende at vi finner *noen* feil, men vi kan ikke på noen måte vite at vi har funnet *alle* feil. Det er grunnen til at vi tester våre programmer ved å gi input, men allikevel blir det svært ofte feil igjen som testingen ikke finner.

En mulighet er å lage et stort antall input tilfeldig. Igjen er problemet jobben med å gå igjennom input/output manuelt.

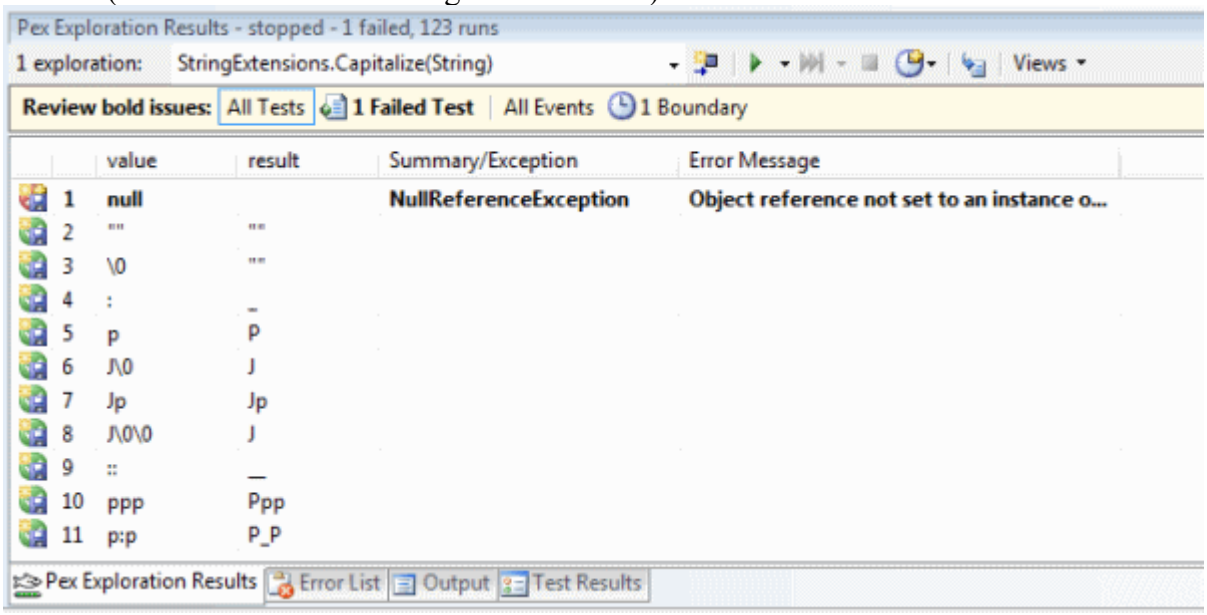
Whitebox

I whitebox testing, kjenner vi koden og utnytter det til å lage input som best mulig tester koden. Spesielt må vi se etter alle grenseverdier og valg for å sjekke at løkker og valg går riktig. Det er ikke enkelt. I en kode med f.eks. bare 10 if-setninger, er det 2^{10}=1024 veier gjennom koden, så det er vanskelig å sikre at alle kommer med.



Som man ser, tenker jeg meg her test cases som er laget for å dekke koden. Det er i seg selv problematisk, for egentlig skal koden være tilpasset spesifikasjonen og da må den testes etter den. Her er nok spesifikasjonen (i form av kommentaren i koden) litt for vag. For hva med andre tegn som bindestrek, komma, parenteser, anførselstegn o.a.? Spesifikasjonen er uklar om det. I koden her blir de ignorert, men kunne enkelt vært tatt med, f.eks. ved å teste på `Char.IsPunctuation(tegn)`.

Det finnes verktøy som analyserer koden og genererer tilfeldig input for alle veier gjennom koden. Pex er et slikt verktøy for NET-miljøet[50]. Her er noen av test cases som Pex produserte for koden ovenfor (som da var skrevet i C# og litt annerledes):



[50] Det lar seg dessverre ikke kjøre med VB Express, så vi får ikke demonstrert det, men det foreligger en video.

Den viktigste feilen Pex fant, var at argumentet kan være *Nothing* (*null* i C#) – det hadde ikke programmereren tenkt på.

Flere versjoner av samme funksjon

Dette er en metode som bl.a. NASA har brukt, når de skriver programmer av ekstrem viktighet, f.eks. for romfergen. Da lar man flere, *uavhengige* teams skrive hver sin versjon av funksjonen. Alle versjonene legges inn i koden. En modul som skal benytte funksjonen, ber alle versjonene finne svaret og sammenlikner. Hvis alle versjonene returnerer samme svar, regnes det som åpenbart korrekt. Hvis bare én avviker, noterer systemet seg at den ikke er til å stole på. Denne funksjonens svar gis mindre vekt senere. Ellers avgjør flertallet hvilken verdi som skal brukes, men mennesker kan varsles hvis avvikene er store og eller et klart flertall er vanskelig å finne. Etter sigende laget NASA tre versjoner av alle viktige funksjoner.

Manuelle metoder

Det finnes mange, forskjellige, manuelle måter å kontrollere programkode på, og de forutsetter alle at koden er kjent. Den vanligste er *gjennomgang* (*walk through*). Da presenteres koden av forfatteren. Deltakere kan være kolleger, revisor, ledere eller andre. Gjennomgangen skjer trinn for trinn, og deltakerne ser kritisk på koden, kommer med spørsmål osv. Alle feil og ubesvarte spørsmål noteres ned. Hvis det ikke er noen, kan koden godkjennes. En formell og godt kjent form av gjennomgang er *Fagin's Inspection*, som spesielt interesserte kan lese mer om på http://en.wikipedia.org/wiki/Fagan_inspection.

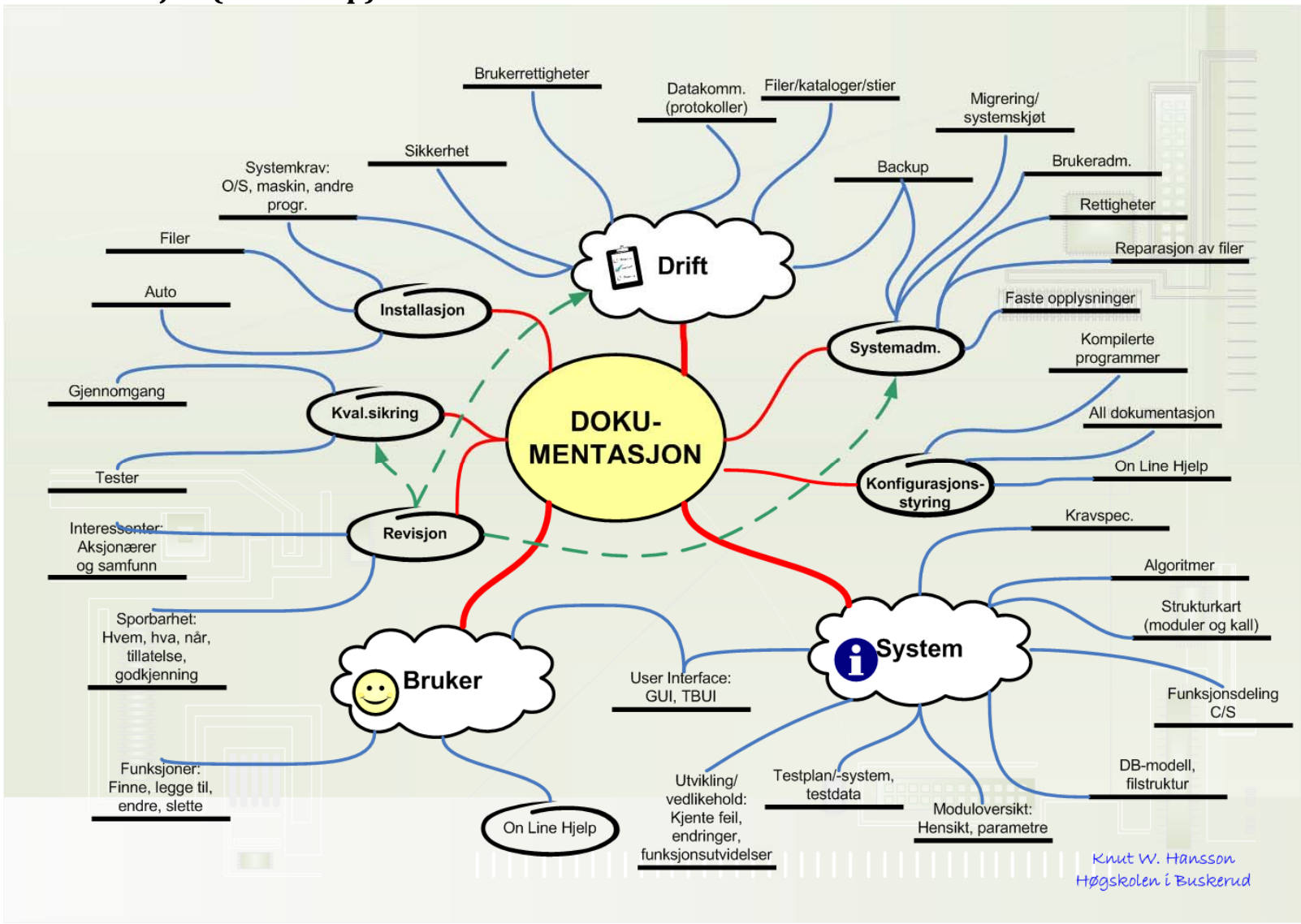
Software metrics

Det kan være aktuelt å se ekstra nøye på visse programdeler. Ved å samle mål om programkoden, kan man velge ut de delene som er mest interessante å teste, f.eks. fordi de kalles oftest (antas å være kritiske), eller er mest komplekse (det er gjerne der feilene kommer). Det finnes flere verktøy som genererer slik mål for koden. I .NET-miljøet finnes f.eks. *Metrics Power Tool* som gir

- ✓ Maintainability Index (vedlikeholdsvennlighet)
- ✓ Cyclomatic Complexity (antall uavhengige iterasjoner i koden)
- ✓ Depth of Inheritance (arvedybde)
- ✓ Class Coupling (argumenter som sendes ved kall)
- ✓ Lines Of Code (LOC) (antall kodelinjer)

Målene gjelder for hver modul og totalt for programmet.

Tema L2 – Dokumentasjon (mind map)



Tema L3 – Hjelp til brukeren

Microsoft fremholder at den aller viktigste hjelpen man kan gi brukeren, er å gjøre sitt ytterste for at programmet er feilfritt. Man må

1. **skrive ryddig og god kode**
2. **teste grundig**

På den annen side, kan vi (fortsatt ifølge Microsoft) ikke regne med at vi klarer å lage feilfrie programmer av noen størrelse. Dessuten vil *brukeren* selv ofte gjøre feil som får alvorlige konsekvenser. Programmer vil normalt stoppe med en feilmelding hvis run-time systemet oppdager feil[51]. Det må vi i det lengste unngå. (Hvis det ikke er til å unngå, bør vi i alle fall loggføre feilen til en fil, og oppgi filnavn osv. til brukeren før programmet avslutter.) Vi må altså

3. **fange feil med "feilfeller"**

Microsoft anbefaler å sentralisere feilhåndteringen, slik at den blir konsistent. Man bør oppgi

4. **melde hva som har skjedd, og**
5. **hva brukeren kan gjøre med det**

Selv vil jeg også nevne betydningen av

6. **"ryddig" og oversiktlig brukergrensesnitt (GUI)**

Et godt brukergrensesnitt leder brukeren gjennom arbeidsoppgavene på en naturlig måte, og det blir lettere å se hva som skal gjøres og fylles inn, hvis det er satt inn i riktig sammenheng.

Brukeren kan også ønske bistand uten at det har oppstått noen feil. Slik formulerer Microsoft det i VB hjelp:

> *No matter how well crafted the application, at some point most users are going to have questions about how to use it. Unless you're going to be there to answer their questions in person, the best way to handle this is by providing a Help file for the application.*

Altså må vi legge opp til at

7. **programmet kan tilby brukeren hjelp**

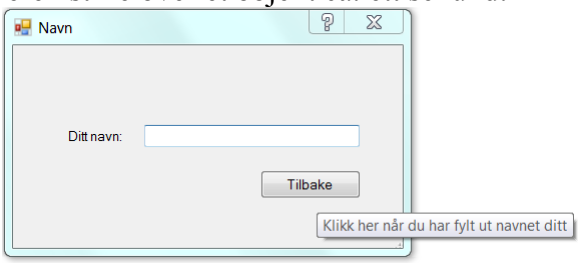
Vi kan tilby brukeren flere typer av hjelp:

1) "ToolTipText"
2) "WhatsThisHelp"
3) Vanlig hjelp (F1)

[51] I nye versjoner av Windows kan brukeren først få en feilmelding med mulighet til å fortsette. Da vil det svært ofte være nytteløst å fortsette, enten fordi feilmeldingen umiddelbart dukker opp igjen eller fordi variabel ikke har fått verdien de skal ha, så nye feil oppstår senere.

1) ToolTipText

Den aller enkleste hjelpen vi kan gi brukeren, er ToolTips. Det er en liten hjelpetekst, som vises når brukeren har holdt musepekeren stille over et objekt ca. ett sekund:



Man kan lage ToolTip-tekst til praktisk talt *alle* kontroller, også kontroller som brukeren ikke kan endre. Det kan jo f.eks. være en etikett som man vil forklare. DataGrid virker imidlertid dårlig.

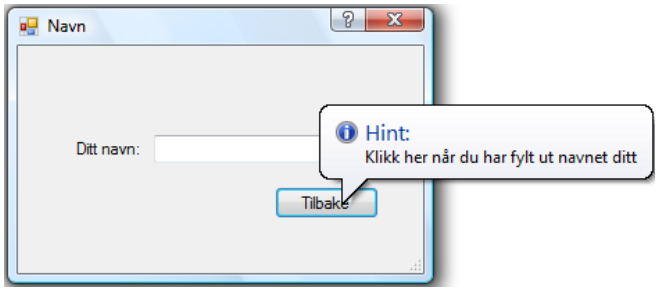
For å vise ToolTipText kreves det en ToolTipText kontroll på skjemaet (verktøykassen under *Common Controls* eller skap en mens programmet går). For hver kontroll setter du egenskapen ToolTipText. Det kan du gjøre i designmodus (egenskapen *ToolTip*), eller i programmet:

```
ToolTip1.SetToolTip(butTilbake, "Klikk her når du har fylt ut navnet ditt")
ToolTip1.SetToolTip(txtNavn, "Skriv navnet ditt her")
```

ToolTipText kontrollen har også andre egenskaper som kan være interessante (og som er satt i figuren), bl.a.

```
ToolTip1.IsBalloon = True
ToolTip1.ToolTipTitle = "Hint:"
ToolTip1.ToolTipIcon = ToolTipIcon.Info
```

Da blir det slik:



2) "What'sThisHelp" (="PopUp Help")

Skjemaegenskaper

Øverst til høyre i vinduet, kan man sette et ikon med som ser ut som et spørsmålstegn. Når brukeren klikker på det, forandres musepekeren til et spørsmålstegn. Når brukeren deretter klikker på en kontroll, kommer det frem et lite vindu med hjelp om den kontrollen. Hjelpen likner svært på den han får i en ToolTipText, og er derfor ikke lenger så mye i bruk. Isteden gir man kontekstavhengig hjelp med tasten F1. Selv om skjemaet har PopUp hjelp, kan du vise vanlig, full hjelp som reaksjon på bruk av F1-tasten.

For hver kontroll som skal gi hjelp, må vi angi hvilken hjelp som skal vises i kontrollens egenskap *HelpString*. Du gir den verdi i designmodus, eller programmatisk gjennom *HelpProvider* for skjemaet:

```
HelpProvider1.SetHelpString(txtNavn, "Skriv navnet ditt her.")
```

Skjemaet må ha følgende egenskaper (gjerne satt i designmodus):

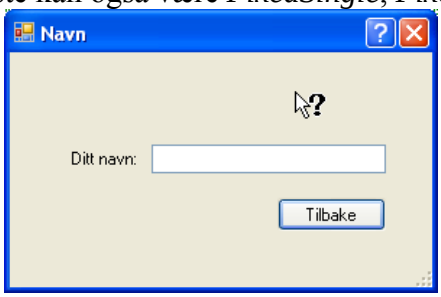
```
Me.MaximizeBox = False
Me.MinimizeBox = False
Me.HelpButton = True
```

Du kan altså ikke vise vindu med både maksimerings-, minimerings- og hjelpknapp samtidig. Hjelpknapp brukes derfor typisk for dialogvinduer og brukes ofte i skjemaer som brukeren skal fylle ut.

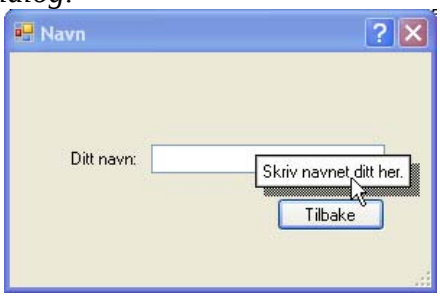
I tillegg må skjemaet ha følgende egenskaper(som også er default):

```
Me.ControlBox = True
Me.FormBorderStyle = Windows.Forms.FormBorderStyle.Sizable
```

Den siste kan også være *FixedSingle*, *Fixed3D* og *FixedDialog*.



Brukeren har klikket spørsmålstegnet



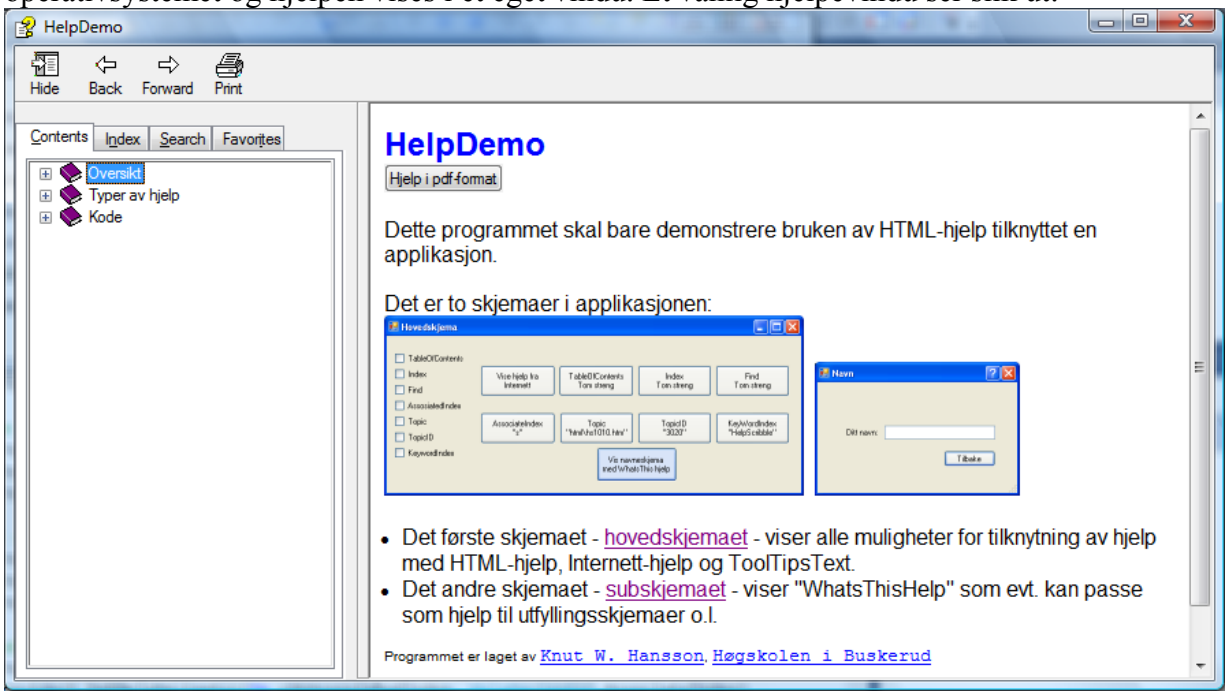
Brukeren har klikket på tekstkontrollen

Meg bekjent er det ikke mulig å få frem en "balloon" med denne formen for hjelp.

3) Interaktiv hjelp (F1)

Brukere er vant til å få hjelp når de bruker tasten F1. Det har vært flere former for hjelp i Windows[52], men i dag er det bare kompilert HTML-hjelp som brukes, i tillegg til pdf-filer og vanlige Internettsider. Kompilerte HTML-filer har etternavn ".chm". Ikonet for slike filer er et "ark med spørsmålstegn":

Hjelpefilen kjøres av et separat hjelpeprogram (HTML Help Control, *hh.exe*) som følger med operativsystemet og hjelpen vises i et eget vindu. Et vanlig hjelpevindu ser slik ut:



[52] Det finnes to hjelpesystemer for Windows, nemlig WinHelp og HTML-help. WinHelp kom med Office og ble standard hjelpesystem for Windows 3.1. Med Windows 95 ble den 32 bits. Med Windows 98 kom HTML-help som nå er det *eneste* som kan brukes sammen med VB 2010. Nedenfor bruker jeg derfor kun HTML-help.

Legg merke til at det er flere faner (*Contents* osv.) og til høyre vises hjelpen som er valgt.

I tillegg kan man alltid vise websider i en standard nettleser. Det kan være det beste hvis applikasjonene skal kjøres over Internett eller hvis du bevisst vil "lokke" brukeren til ditt eget nettsted. Mange leverandører velger å gjøre det slik. Ulempen er jo at brukeren må være tilkoblet Internett for å få hjelp. (Også idag er det brukere som ikke er tilkoblet Internett, f.eks. fordi de jobber på et fly.) I websiden, kan du inkludere en ActiveX eller en Java Applet (Java med GUI) som gjør HTML-help viewer tilgjengelig også for websider. Hvis man har hjelp både kompilert og som Internettside, må man selvsagt passe på at de alltid er like.

Uansett system må man
a) lage hjelpefiler
b) knytte hjelpefilene til applikasjonen
c) knytte spesifikk hjelp til kontrollene

a) Lage hjelpefiler

Det finnes minst fire måter å lage hjelpefiler på:

1. Microsofts egen HTML-Help Workshop (HTML-Help 1.4 SDK), som er gratis tilgjengelig fra Microsoft – se f.eks. http://msdn.microsoft.com/en-us/library/ms670169(v=vs.85).aspx. HTML Help Workshop genererer de filene som er nødvendige og kompilerer dem[53].
2. Skriv hjelpesidene i **MS Word** (jeg har ikke forsøkt med **OpenOffice Writer**, men jeg antar at det også vil virke greit). Hvert hjelpetema må ha sin egen side og nederst på samme side må det stå en fotnote med tegnet # som fotnotemerke etterfulgt av et nøkkelord (unik identifikator). Dokumentet lagres som rtf-fil og kompileres med Help Compiler (hhc.exe) som følger med HTML Help Workshop. Du kan lese mer om dette på http://support.microsoft.com/default.aspx?scid=kb;en-us;175491. Dette var måten det ble gjort på da HTML Help først ble introdusert. Selvom Word er grei å bruke, blir hele prosessen tungvint, gir lett feil og anbefales ikke. (Det finnes noen hjelpeprogrammer som knyttes til Word og bistår med prosessen. Disse har jeg ikke prøvd.)
3. Bruk en **HTML-editor** og lag din egen html-fil. Du må inkludere Microsofts HTML Help ActiveX Control og sette parametrene riktig. Dette gir full kontroll/fleksibilitet, men krever betydelig kompetanse og anbefales derfor ikke.
4. Bruk en dedikert, **tredjeparts Help Editor**. ***Dette alternativet anbefaler jeg.*** Jeg anbefaler da gratisprogrammet HelpScribble Demo, som er en av mange slike editorer. Les nedenfor om hvordan du installerer og bruker HelpScribble.

b) Knytte hjelpefilene til applikasjonen

Microsoft anbefaler å legge hjelpefilen sammen med applikasjonen. Hvis du legger den et annet sted, anbefaler de å bruke full sti, men det blir jo dumt når applikasjonen flyttes, så bør du i alle fall bruke relativ sti og *My.Application.Info.DirectoryPath*.

For å gi hjelp utover ToolTipText og WhatsThisHelp, kreves det alltid at skjemaet har en ***HelpProvider*** kontroll (verktøykassen under *Components*). Den må ha egenskapen *HelpNameSpace* satt. Den må vise til en chm-fil. Programmatisk:

```
HelpProvider1.HelpNamespace = My.Application.Info.DirectoryPath _
     & "\HelpDemo.chm"
```

Du kan også alternativt henvise til en webside ved å angi full URL:

```
HelpProvider1.HelpNamespace = "http://www.knut.hansson.name"
```

[53] Det er en stund siden jeg prøvde den sist, og jeg fant den da både forvirrende og tungvint. Det *kan* jo ha bedret seg. Den seneste versjonen jeg finner, er fra 2007 og henviser til Windows XP som høyeste versjon. Det tyder på at Microsoft har sluttet å vedlikeholde den – de har gitt opp konkurransen.

c) Knytte spesifikk hjelp til kontrollene

i) Hjelp for hver kontroll

Hjelpen som vises med tasten F1, er avhengig av hvilken kontroll som er i fokus. Vi må altså knytte kontrollene til et bestemt tema i hjelpefilen. Det gjør vi i designmodus ved å sette tre egenskaper, eller programmatisk gjennom HelpProvideren:

```
HelpProvider1.SetHelpKeyword(Me.chkTopicID, "1010")
HelpProvider1.SetHelpNavigator(Me.chkTopicID, HelpNavigator.TopicId)
HelpProvider1.SetShowHelp(Me.chkTopicID, True) 'default=False
```

Her har vi har bestemt at hjelpen for kontrollen *chkTopicID* skal finnes gjennom ID for temaet (*TopicID*), at for denne kontrollen er ID = 1010 i hjelpefilen (*HelpKeyWord*), og vi har satt på hjelpen.

Det finnes seks andre måter å finne riktig hjelpeside på, ved å sette andre *HelpNavigator*-verdier. I et medfølgende eksempel, viser jeg alle variantene, men jeg synes *TopicID* er den enkleste å bruke.

ii) Hjelp generelt for hele skjemaet

Du har kanskje ikke tilknyttet hjelp til *alle* kontrollene på skjemaet. Da bør du knytte en generell hjelp til skjemaet også. Den vil "overta" bare hvis kontrollen som er i fokus *ikke har tilknyttet egen hjelp*. Dette gjør du med egenskapene for *skjemaet*, på samme måte som for hver kontroll (et skjema *er* jo en form for kontroll).

```
'Tilordne hjelp til skjemaet - vis Innhold
HelpProvider1.SetHelpKeyword(Me, Nothing)
HelpProvider1.SetHelpNavigator(Me, HelpNavigator.TableOfContents)
HelpProvider1.SetShowHelp(Me, True) 'default=False
```

Vise hjelp med knapp eller meny

Det er ofte greit for brukeren å ha en egen knapp og/eller meny merket "Hjelp", som viser hjelpefilen. Det er også aktuelt å vise hjelpesider når det oppstår feil.

Til det brukes klassen *Help*. Denne klassen er abstrakt, så det kan ikke lages objekter av den. Den har imidlertid bl.a. metoden *ShowHelp*. Den har fire varianter, gitt ved forskjellige argumenter. Her viser jeg bare én av dem, nemlig den mest omfattende:

```
Help.ShowHelp (kontroll, hjelpefil, hjelpetype, parameter)
```

- ✓ **Kontroll** er den kontrollen som "eier" hjelpedialogboksen. Hjelpeboksen vises modalt på samme måte som med *ShowDialog*, dvs. at når den lukkes så aktiveres "eieren" (*parent*) igjen. Bruk gjerne *Me* her, så blir skjemaet "eier".
- ✓ **Hjelpefil** er en streng som peker til en hjelpefil. Det kan være en lokal fil (enten chm-fil eller html-fil) eller en URL som evt. vil bli åpnet av default nettleser.
- ✓ **Hjelpetype** er en av de syv *HelpNavigator*-verdiene (f.eks. *HelpNavigator. TableOfContents*):
- ✓ **Parameter** er et objekt. Det vil være et strengobjekt, der du angir hvilken hjelpeside som skal vises.

Det synes mest naturlig for en slik hjelp å vise innholdsfortegnelsen, indeks eller søkefanen (og ikke en bestemt side). Da må hjelpetypen være henholdsvis *TableOfContents*, *Index* eller *Find*. Parameteret kan du sette til en tom streng for alle disse.

```
Help.ShowHelp(Me, hjelpefil, HelpNavigator.TableOfContents, "")
```

Hvis det er flere skjemaer med hver sin hjelpeside, anbefaler jeg å bruke *TopicID*.

Hvis du vil åpne en lokal html- eller pdf-fil, skriver du bare

```
Help.ShowHelp (kontroll, hjelpefil)
```

der hjelpefilen henviser til den lokale filen:

```
Help.ShowHelp(Me, "HTML\HelpDemo.html")
```

I det medfølgende programeksemplet viser jeg alle mulighetene.

Klikkbare bilder

En svært vanlig, og effektiv hjelp, er at hjelpefilen inneholder et bilde av applikasjonen som det kan klikkes på. Brukeren får hjelp avhengig av hvor på bildet han klikker.

Du skaffer deg et bilde av den kjørende applikasjonen med *Alt+PrtSc* og lagrer det. Dette bildet må du så redigere, slik at du knytter lenker til områder på bildet. Da lager du et såkalt SHG-bilde (*Segmented HyperGraphics*). Til dette trenger du en egen editor – i HelpScribble er den inkludert.

Oppbygning av HTML-hjelp

Alt som står i hjelpen bør sees fra *brukerens* side. Det kan likevel være interessant å fortelle noe om hvordan programmet gjør ting, men bare hvis det kan interessere brukeren.

Inndeling i sider

HTML-hjelp bør deles inn i *sider*. Du bør ha

1) En **hovedside** (”hjem”) der du forteller litt om programmet, overordnet hva det gjør og hvordan det virker. Hvis det er mye å fortelle, bør den deles i flere sider med et hovedtema for hver. Hovedsiden bør ha en meny med lenke til hvert skjema og/eller hovedtema.
2) En **førsteside for hvert skjema**. Der bør du vise et bilde av skjemaet (klikkbart shg) med lenker til forklaring av hver kontroll, og fortelle hva som gjøres i dette skjemaet.
   Hvis det bare er ett skjema, kan denne slås sammen med hovedsiden.
3) En **side for hver kontroll** som skal få spesifikk (kontekstavhengig) hjelp.
4) Lenker til andre sider av interesse (”Se også”), og andre lenker av interesse. Siden HTML-hjelp vises i en HTML viewer, kan du også lenke til websteder på Internett med en URL.

Innhold på sidene

I tillegg til teksten og bilder, bør hver side ha

- ✓ En **overskrift med tema** (Topic) – det vises som overskrift øverst på siden.
- ✓ En **identifikator** (TopicID) – et heltall som entydig identifiserer siden. Microsoft bruker selv ID 1-512, og anbefaler at du bruker ID fra 1000 for det første skjemaet, fra 2000 for det neste osv. Disse får brukeren ikke se. Det kan være lurt å nummerere med bare hvert tiende tall, f.eks. 1000, 1010, 1020 osv., så er det enklere å legge nye sider innimellom (et gammelt triks fra den gang vi måtte nummerere linjene i Basic-programmer!).
- ✓ Ett eller flere **nøkkelord** – de kommer inn under Index-fanen så de blir søkbare for brukeren.

Innholdsfortegnelse

Endelig bør du lage en **innholdsfortegnelse** der det er henvisning til de viktigste sidene. Brukeren får en egen fane merket ”Content” der din innholdsfortegnelse vises.

Søkeord

Du bør vurdere å generere fulltekst søking. Da bygger kompilatoren selv opp en egen indeks for alle ordene du har brukt på hjelpesidene dine, og gjør dem søkbare. HTML-hjelp viewer vil vise en egen fane merket ”Search” der brukeren kan søke i teksten. Dette er altså noe annet enn nøkkelordene som vises under fanen ”Index”.

How-to

Vurder å lage en ”How-to” seksjon, der du enkelt forteller nye brukere hvordan de skal få til ting. Denne er primært for nybegynnere, og skiller seg klart fra sider med de mer tekniske detaljene.

Sekvenser

Hvis du vil at brukeren skal kunne "bla" gjennom hjelpesidene i en rekkefølge, må du lage det selv. Bruk lenke, evt. en knapp, for "neste side" (brukeren kan alltid komme tilbake – det er egen knapp for det i vieweren).

Feilfeller

Når du fanger feil, kan det også være aktuelt å vise en hjelpeside (og ikke bare gi en feilmelding). Da må du lage hjelpesider for slike feil også.

Installasjon og bruk av HelpScribble

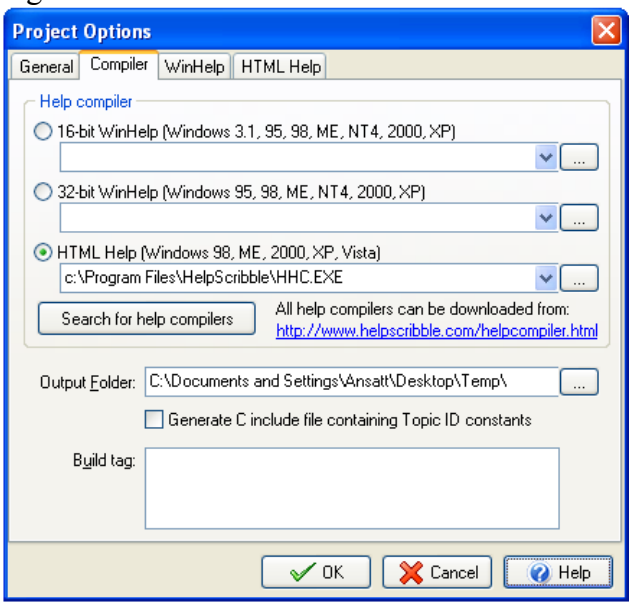
HelpScribble har en gratisversjon som legger på en tekst nederst på hver side. Det gjør den ubrukelig kommersielt, men helt OK for studenter.

Installasjon

HelpScribble laster du ned gratis fra http://www.helpscribble.com/download.html. Installer og start HelpScribble og se om du har kompilatoren for HTML Help (*hhc.exe*) – sjekk *Options|Compiler* og søk etter kompilere. Hvis du ikke har den, kan du laste den også fra http://www.helpscribble.com/helpcompiler.html og installere den.

Vær også sikker på at du kan konvertere fra tekstbehandleren din til pdf-filer. Selv bruker jeg den ”virtuelle skriveren” CutePDF (http://www.cutepdf.com), som er gratis, og den har jeg gode erfaringer med. Det er også fullt mulig å bruke Microsoft Word til dette (”lagre som” pdf) og Open Office Writer (”eksporter” som pdf).

Når du har alt dette på plass, og har sjekket f.eks. med en teksteditor at pdf-konverteringen virker, starter du HelpScribble. Åpne ”Files|Options”, fanen ”Compiler” og sjekk at HelpScribble har funnet HTML kompilatoren. Hvis ikke må du be HelpScribble om å lete etter den, eller selv ”vise” HelpScribble hvor den er lagret.



Gode hjelpesystemer har ofte klikkbare bilder av skjemaer. Når du tar bilde av skjermen (med *ALT+PrintScreen*) legges bildet på utklippstavlen[54]. Derfra kan du lime det inn i en billedbehandler og lagre det.

Oppsummering.

Du trenger altså

- ✓ Hjelpeditor, f.eks. HelpScribble
- ✓ Kompilatoren hhc.exe
- ✓ Mulighet for å lagre pdf, f.eks. Word/Writer eller en virtuell pdf-skriver
- ✓ Mulighet for å ta bilder av skjermen, f.eks. *ALT+PrtScr* eller et helst et program som tar bilde i png-format

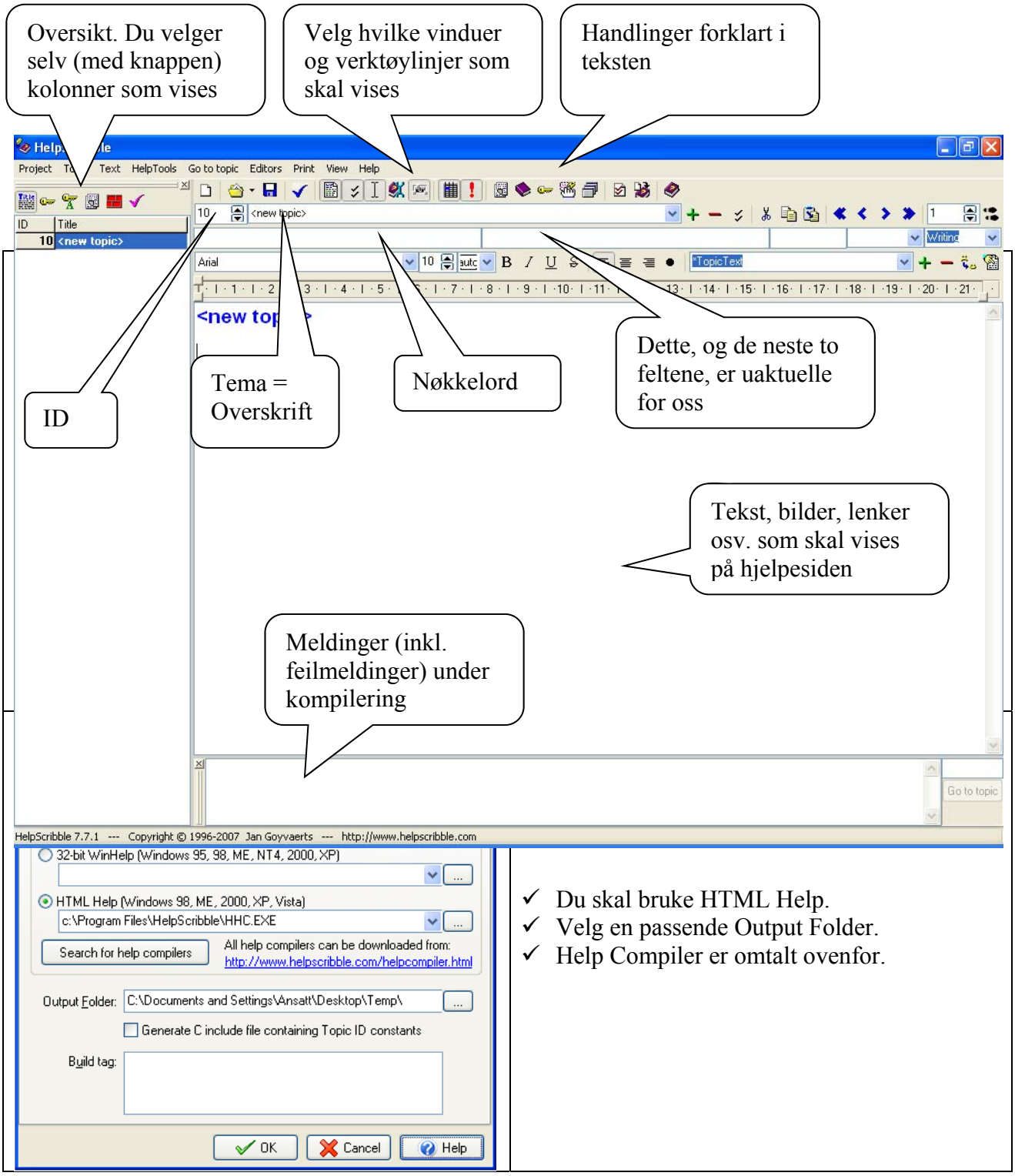
Note: Jeg har også forsøkt programmet HelpNDoc. Den er gratis til privat bruk og legger da til et banner nederst med lenke til HelpNDocs hjemmesider.

Editoren er svært enkel å installere og enkel å bruke. Den produserer chn-, pdf-, html- og Word-filer med ett eneste klikk og uten problemer. Alle formatene får lenker som virker.

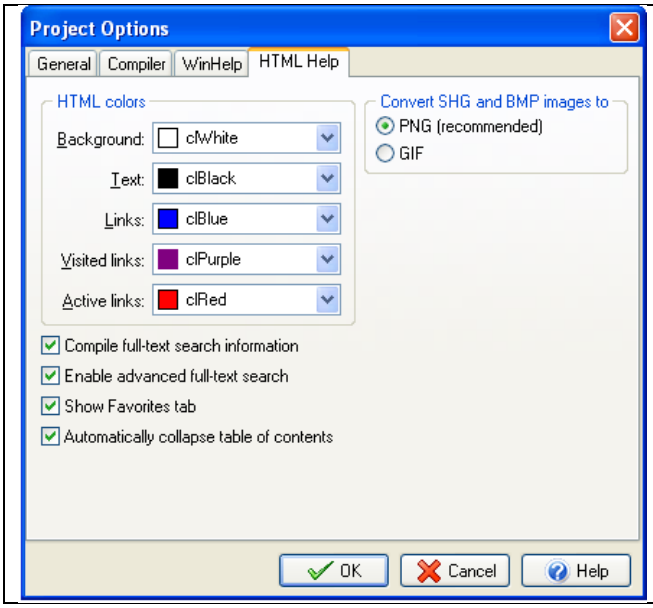
Jeg savner muligheten til å legge inn knapper med lenke, og editere klikkbare SHG-bilder (Segmented HyperGraphics). Ellers er HelpNDoc minst like bra som HelpScribble og har enklere grensesnitt og installasjon.

[54] *ALT+PrtScr* i Vista lagrer bildet med jpg-format. Det har begrensninger og gir tapt informasjonen. Formatet png håndteres greit av de fleste nettlesere og andre programmer, og er komprimert uten tap. For å ta png-bilder av skjemaene, må du følgelig bruke et annet program. Selv bruker jeg Gadwin PrintScreen, gratis fra http://www.gadwin.com/download/index.htm#PrintScreen.

Bruk



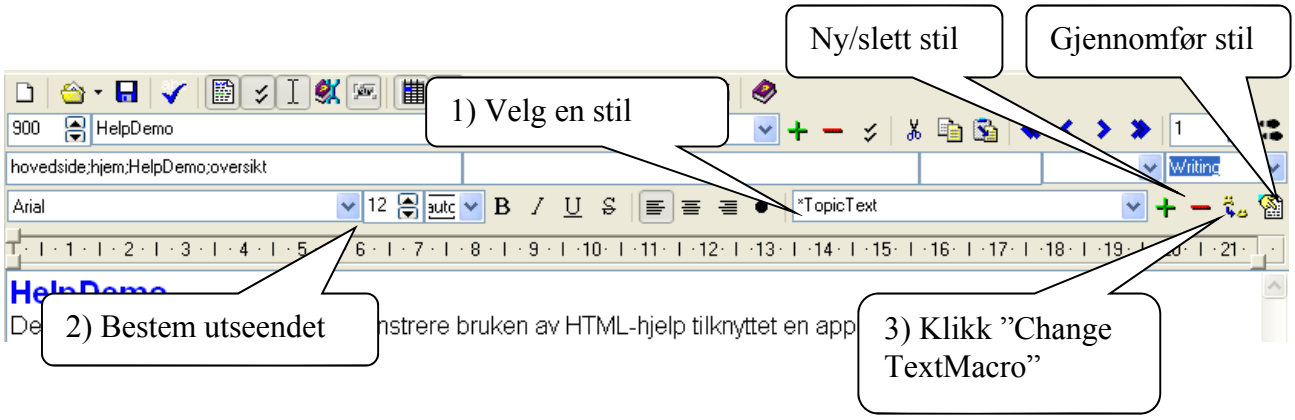
- ✓ Du skal bruke HTML Help.
- ✓ Velg en passende Output Folder.
- ✓ Help Compiler er omtalt ovenfor.



Legg merke til at jeg har satt på alle valgene nederst.

- ✓ De to første gir brukeren en search-fane for fritekst søking.
- ✓ Favorites tab gir en fane for favoritter. ”Collapse table of contents” virker slik at når brukeren klikker for å åpne en overskrift i innholdsfortegnelsen, så lukkes automatisk andre som er åpne.
- ✓ Jeg foretrekker også png-grafikk, siden den er mye bedre enn gif. Både GIF og PNG er tapsfri, men GIF har begrenset antall farger. (WinHelp, som vi ikke lenger bruker, har problemer med png-grafikk.)

Endre og legge til stiler



Skrive inn teksten

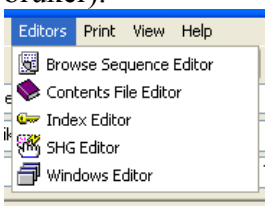
Det skjer omtrent som i en hvilken som helst teksteditor. Sett inn bilder, lenker, knapper og annet her. Du kan knytte lenke til både tekst og bilder. Det er fullt tillatt å linke til en webside, til e-post osv.

Legg til nøkkelord

Nøkkelordene kan du legge til for hver hjelpeside – på linjen over teksten. Du kan også bruke den egne editoren for nøkkelord (Index Editor). Du kan skrive flere nøkkelord for hver side, atskilt med semikolon. Setter du komma mellom to nøkkelord, blir det andre et subtema: ”Program, kode”.

Editorer i HelpScribble

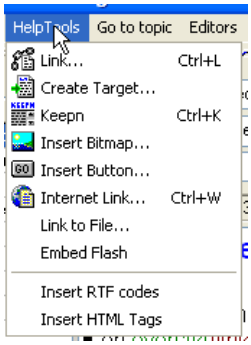
Det er fem editorer, men sekvenser og vinduer er bare aktuelle for den gamle WinHelp (som vi ikke bruker).



- ✓ **Contents** er innholdsfortegnelsen. Du må lage den manuelt og separat fra indeksen.
- ✓ **Index** hjelper med nøkkelordene. Du angir flere nøkkelord for et tema ved å skille dem med semikolon. Hvis du skriver to ord atskilt med komma, vil det siste være et underbegrep.
- ✓ **SHG** bruker du for å gjøre bilder klikkbare.

Editorene er stort sett selvforklarende.

Legge til spesielle elementer



Menyen *HelpTools* gir tilgang til verktøy for forskjellige elementer i hjelpefilen:

- ✓ *Link…* bruker du for å lenke til et annet sted i hjelpfilen. Du må merke tekst/bilde *før* du lager lenken som skal knyttes til teksten/bildet. Pass på at du kan krysse av hvis lenken skal vises i eget vindu, kalt ”Popup link”. (Lenke til annen dokumentasjon, URL o.l. gjør du med *Internet Link…*)
- ✓ *CreateTarget…* skaper et ”anker” som du kan ”hoppe til”, midt i teksten. (Alle andre hopp går til begynnelsen av et tema.) For å hoppe til et anker, bruker du *Link*.
- ✓ *Keepn* er en form for sideskift. Den virker bare i WinHelp og er ikke lenger aktuell.
- ✓ *Insert Bitmap…*setter inn et bilde. Det kan være vanlig grafikk (jpg, gif, png) eller et klikkbart bilde (shg).
- ✓ *Insert Button…* setter inn en knapp, med tekst og lenke til et annet tema. (Du kan her lenke til en WinHelp makro, men det virker bare i WinHelp og er ikke aktuelt.)
- ✓ *Internet Link…* lager en lenke til noe utenfor hjelp, f.eks. en webside (lenk med *http://...*), en e-postadresse (lenk med *mailto:...*), en fil (lenk med filnavnet, evt. sti).
- ✓ *Link to File…* lager en knapp som lenker til en fil. Du velger altså om du vil ha lenken som en tekst (*Internet Link*) eller knapp *(Link to File*).
- ✓ *Embed Flash* setter en Flash-animasjon inn i hjelp.
- ✓ *Insert RTF codes* er bare aktuelt for WinHelp, som bruker rtf-format internt
- ✓ *Insert HTML Tags* setter inn HTML tagger. Det kan være nødvendig, hvis du vil ha formater eller annet som HelpScribble ikke støtter.

Kompilere hjelpen

Du kan når som helst kompilere hjelpen (knapp opp til høyre, eller *Project|Make*). Når du kompilerer, lages en HTML-hjelp fil (chm). Samtidig vises den, så du kan se hvordan den blir.

Lage hjelpen i andre formater

Det er aktuelt å lage hjelpen i andre formater også. Særlig aktuelt er kanskje en "flat" HTML-fil for hjelp, dvs. all hjelpen samlet på en webside. Det kan også være aktuelt å lage en rtf-fil som kan åpnes i en tekstbehandler for overføring til pdf derfra. Bruk menyen *Project|Make Flat Manual*.

Det kan også være interessant med all hjelpen i en pdf-fil som en slags manual. Da kan du åpne den ”flate” HTML-filen (eller rtf-filen) med Word/Writer, sørge for passende sideinndeling og kanskje redigere litt, før du lagrer/eksporterer som pdf eller du skriver ut websiden til en PDF-skriver.

Tema M – Repetisjon av dette kurset

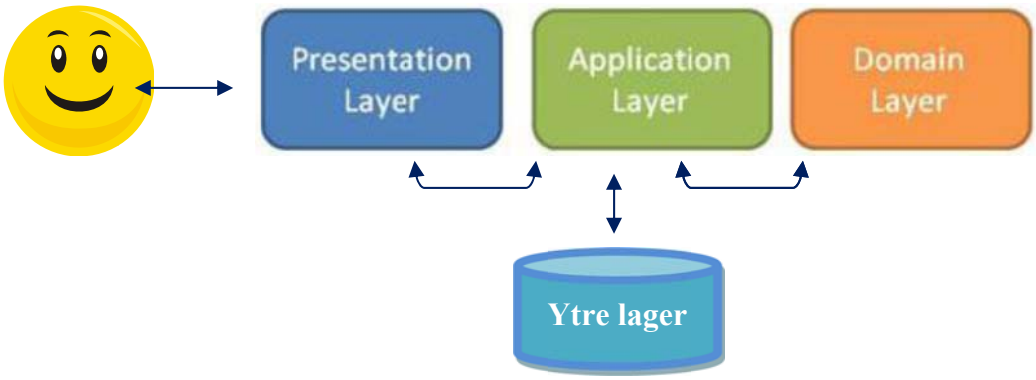
Repetisjonen presenteres som en oppgave, men det er ikke meningen at dere skal forsøke å løse den. Det gjør jeg i timene. Det vil nok være en fordel for dere å ha med en utskrift til gjennomgangen.

I denne tenkte oppgaven, får vi sett på mye av det sentrale i pensum:

1. Analysere problemet og utforme en løsning
2. Bruk av egenskrevet bibliotek med dokumentasjon (XML-tags)
3. Bruk av klasser/objekter
4. Flere skjemaer i samme program
5. Filbehandling
6. Bruk av MySQL database – lagre og hente rader med SQL DML
7. Utstrakt bruk av funksjoner og prosedyrer med argumenter
8. Overskriving av ToString()
9. Lister (samlinger) og gjennomgang av dem
10. Try-Catch og feilhåndtering
11. Posit/admit og akkumulering

Programmet er ikke strukturert *helt* slik jeg ville anbefale. Jeg tillater bl.a. skjemaene å få direkte tilgang til, og endre, objektene – det vil jeg egentlig helst skal skje gjennom et eget kontrollobjekt. Forenklingene gjør allikevel resten tydeligere, og derfor har jeg tillatt meg det her.

Slik bør et objektorientert program helst struktureres (litt forenklet):



1. Domenelaget er de objektene som modellerer omverdenen og har de dataene som vi vil ta vare på – her er det lokomotivobjekter og avlesningsobjekter.
2. Applikasjonslaget er ett, enkelt objekt som styrer all logikken slik som å legge til/fjerne et lokomotiv eller et avlesningsobjekt. Listene med lokomotiver og avlesninger lagres der. Det er også applikasjonslaget som henter og lagrer objekter og håndterer filbehandlingen.
3. Presentasjonslaget er skjemaene. All kommunikasjon med brukeren skal skje gjennom presentasjonslaget.

All kommunikasjon mellom lagene skal skje gjennom applikasjonslaget. Det er derfor ikke tillatt for presentasjonslaget å kommunisere med domenelaget.

Dette er ting vi kommer tilbake til i senere kurs, flere ganger.

Oppgave "Strømforbruk"

Generell beskrivelse

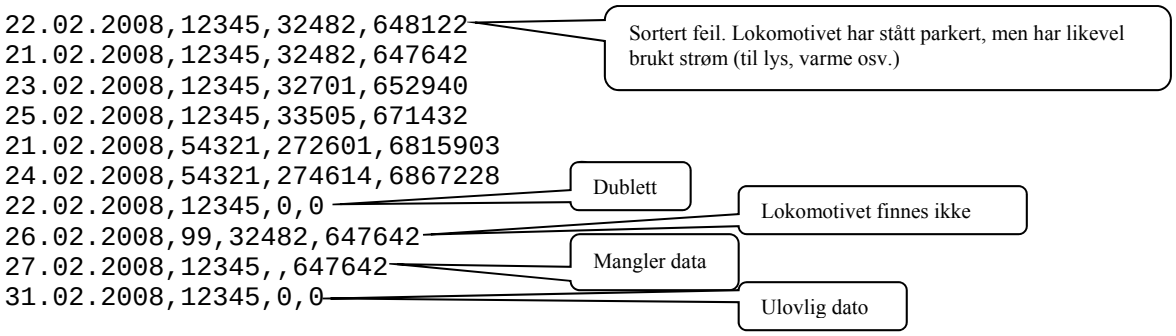
NSB vil undersøke strømforbruket for de elektriske lokomotivene[55]. Man har installert en SMS-sender i hvert lokomotiv som omtrent én gang pr døgn sender en melding til systemet. Meldingen

[55] Vanligvis ca. 20 – 25 kwh/km ifølge http://www.norskbane.no/hk/kvalsikr.doc

inneholder lokomotivets identifikasjonsnummer (heltall), total kilometerstand (heltall km) og totalt strømforbruk (heltall, kwh). I tillegg er meldingen merket med avsenderdato.

Det er to lokomotiver som inngår i undersøkelsen foreløpig, nemlig nummer 12345 og 54321. De ble tatt med i forsøket 21.2.2008, og da ble kilometerstand og målerstand avlest og registret for begge.

Denne SMS-meldingen behandles automatisk og dataene lagres på en kommalimitert fil kalt *avlesninger.dat*, med én post på hver rad, f.eks. 10 poster slik:



Det er ingen sortering av filen – postene legges til etterhvert som de mottas (den første posten i eksempelet er på feil plass). Legg merke til at de fire siste postene er gale – programmet skal oppdage dem og skrive feilmelding filen *feilavlesninger.dat*. Ved kjøring av ovenstående innfil, vil feilfilen vise fire feil:

```
22.02.2008,12345,0,0 - The changes you requested to the table were not
successful because they would create duplicate values in the index, primary key,
or relationship.  Change the data in the field or fields that contain duplicate
data, remove the index, or redefine the index to permit duplicate entries and
try again.
26.02.2008,99,32482,647642 - Finner ikke lokomotivet
27.02.2008,12345,,647642 - Conversion from string "" to type 'Integer' is not
valid.
31.02.2008,12345,0,0 - Conversion from string "31.02.2008" to type 'Date' is not
valid.
```

Det skal nå lages et objektorientert program med Visual Basic, som leser denne filen, lagrer dataene i en relasjonsdatabase (MySQL) og beregner gjennomsnittlig strømforbruk i kwh/km hittil for hvert lokomotiv. Beregningen skjer etter følgende formel for hver avlesning og avrundes til to desimaler:

$$snitt = \frac{(kwh\ nå - kwh\ ved\ start)}{(km\ nå - km\ ved\ start)} = (kwh\ nå - kwh\ ved\ start)/(km\ nå - km\ ved\ start)$$

NSB vil også ha en grafisk fremstilling av dette snittforbruket for hvert lokomotiv over tid, slik at man kan se utviklingen.

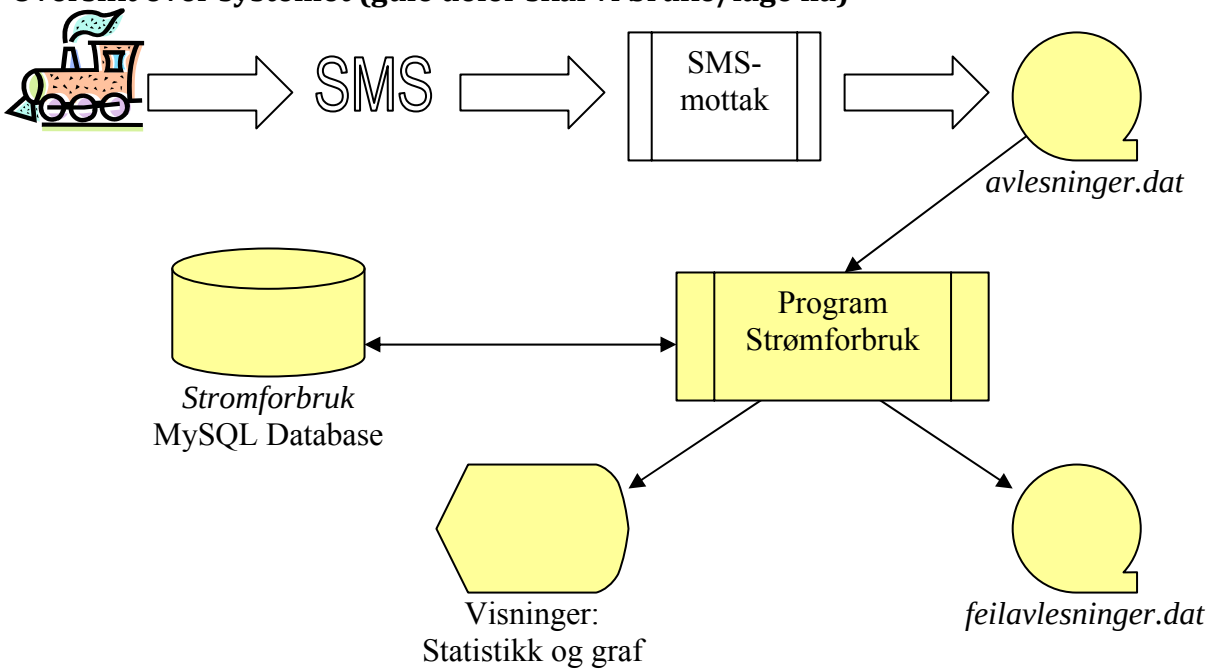
Din oppgave
Lag dette programmet.

Bruk den vedlagte databasen og filen *avlesninger.dat* som eksempel til testing. Unngå å lage datasett med tabeller og adaptere i programmeringsmiljøet (skap isteden de nødvendige objektene selv i programmet ditt).

Løsningsforslag

Analyse & design

Oversikt over systemet (gule deler skal vi bruke/lage nå)



Planlagt funksjonalitet

1) Oppstart
   a) Les databasen og bygg objektene i RAM
   b) Vis menyskjema med tre valg
2) Les fra fil
   a) leser fra avlesningsfilen
   b) kontrollerer – ved feil skriver feilmelding til fil
   c) bygger objektene
   d) lagrer nye avlesninger i databasen
3) Vis statistikk i eget skjema
   a) beregn kjørelengde siden start
   b) beregn strømforbruk siden start
   c) beregn snitt kwh/km siden start
   d) vis resultatet for hver avlesning
4) Vis statistikken grafisk i eget skjema

Filer

Filen *avlesninger.dat* er en ren, sekvensiell tekstfil og har følgende struktur:

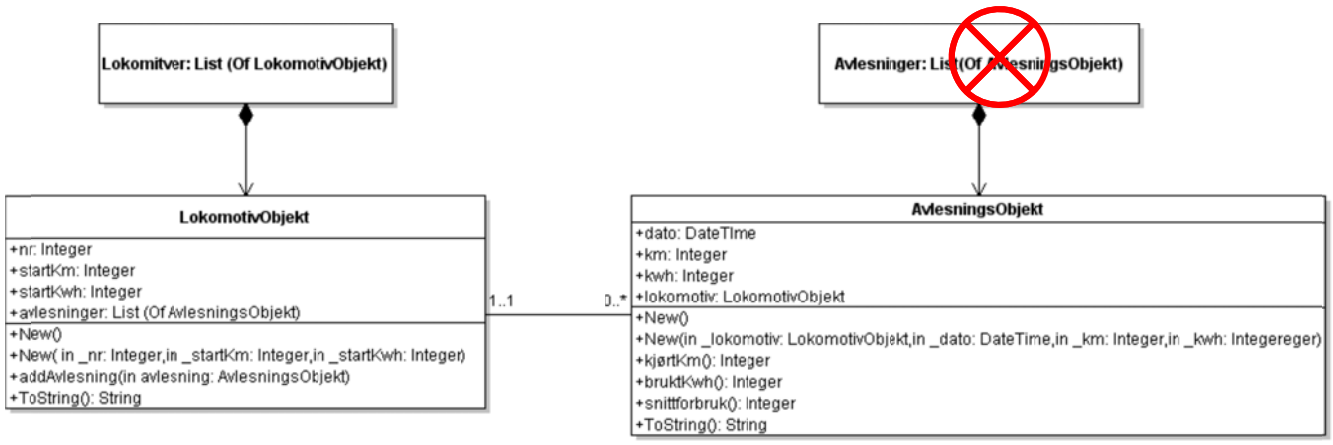
Felt:	**Dato (SMS)**	**Lokomotivets nummer**	**Lokomotivets kilomterstand**	**Lokomotivets målerstand**
Variabel:	**dato**	**loknr**	**km**	**kwh**
Datatype:	***DateTime***	***Integer***	***Integer***	***Integer***
Poster:	22.02.2010	12345	32482	648122
	21.02.2010	12345	32482	647642
	23.02.2010	12345	32701	652940
	25.02.2010	12345	33505	671432
	21.02.2010	54321	272601	6815903
	24.02.2010	54321	274614	6867228

Feltene er adskilt med komma, postene med linjeskift. Det kan være feil i – eller helt mangle – felt i enkelte poster. Det er ingen sortering av postene, de legges inn etterhvert som de mottas.

Filen *feilavlesninger.dat* skal logge feil som oppdages i *avlesninger.dat*. Det skal være en ren tekstfil som inneholder en kopi av posten som det er feil i, og en passende feilmelding. Feilene adskilles med linjeskift, f.eks.:

```
21.02.2010,12345,32482,647642 - Duplicate entry '2021-02-20-12345' for key
'PRIMARY'
26.02.2010,99,32482,647642 - Finner ikke lokomotivet
27.02.2010,12345,,647642 - Conversion from string "" to type 'Integer' is not
valid.
31.02.2010,12345,0,0 - Conversion from string "31.02.2010" to type 'Date' is not
valid.
```

Klasse-/objektmodell

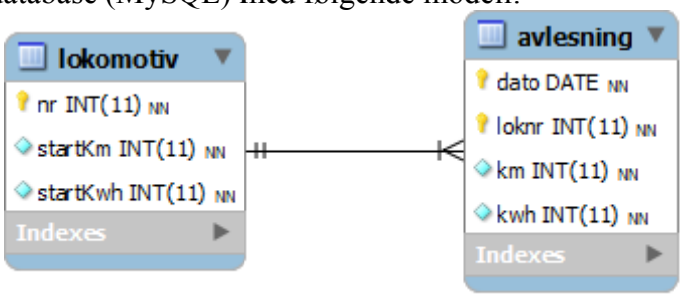


Feltene *startKm* og *startKwh* er avlesninger som ble gjort da undersøkelsen startet. Hvis vi passer på at startverdiene legges inn som første avlesning når et lokomotiv "meldes inn", kan vi slette *startKm* og *startKwh*. Da blir det bare *nr* igjen og det kan vi jo da finne i *Avlesning* (alle lokomotiver vil ha minst én avlesning). Det er altså mulig å fjerne hele *Lokomotiv*. Jeg lar det likevel være som ovenfor, så vi får øvelse med å lese fra to tabeller.

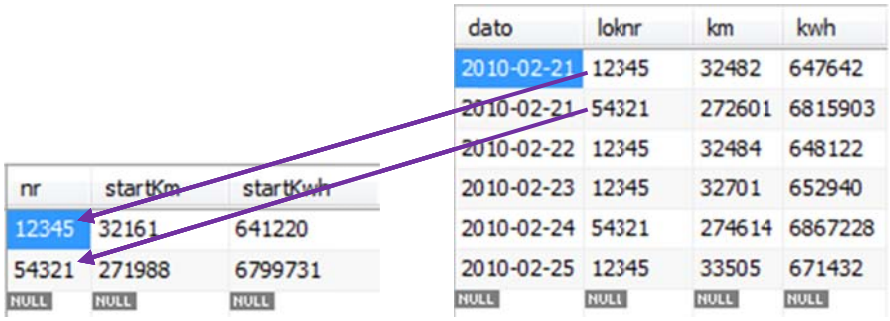
Når vi har en liste over lokomotivene, vil hvert lokomotiv ha en liste over "sine" avlesninger. Da får vi tilgang til alle avlesninger via lokomotivene. Listen over avlesninger er det derfor ikke behov for.

Datamodell for relasjonsdatabasen

Det er laget relasjonsdatabase (MySQL) med følgende modell:



Relasjonen er én til mange med referanseintegritet:



Det er foreløpig bare lagt inn to lokomotiver. Vi legger opp til at vårt program legger inn nye avlesninger (som hentes fra filen *avlesninger.dat*) for registrerte lokomotiver, mens nye lokomotiver legges inn direkte i MySQL.

SQL

Legg inn to lokomotiver:

```
INSERT INTO `stromforbruk`.`lokomotiv`
    VALUES (12345, 32161, 641220);
INSERT INTO `stromforbruk`.`lokomotiv`
    VALUES (54321, 271988, 6799731);
```

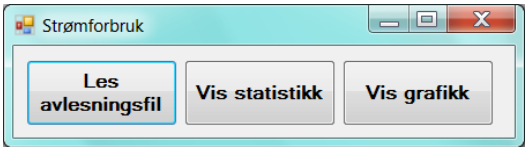
For å hente data og lagre nye avlesninger, trenger vi følgende spørringer:

1) Hente alle lokomotiver:
   `SELECT nr, startKm, startKwh FROM lokomotiv order by nr;`
2) Hente alle avlesninger:
   `SELECT dato, loknr, km, kwh FROM avlesning ORDER BY loknr, dato;`
3) Hente avlesningene til ett bestemt lokomotiv:
   `SELECT dato, loknr, km, kwh FROM avlesning WHERE loknr = <integer> ORDER BY loknr, dato;`
4) Lagre ny avlesning:
   `INSERT INTO avlesning (dato, loknr, km, kwh) VALUES ('<date>', <integer>, <integer>, <integer>);`
   OBS! Datoen kan bare oppgis som f.eks. *2008.02.21*, slik man får med funksjonen Format(<date>,"yyyy.MM.dd"). Merk også anførselstegnene rundt datoen.

Brukergrensesnitt

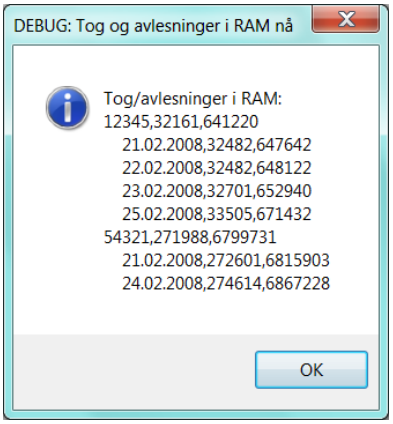
Note: Alle bildene er kjørt med dataene som er oppgitt som eksempel i oppgaven.

1. Hoved-/åpningsskjema:



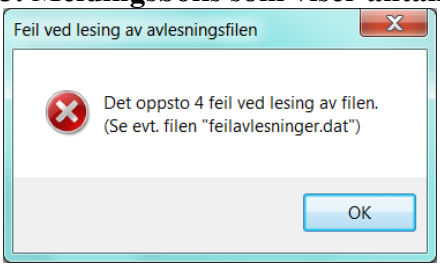
Note: De to siste knappene kunne vært utilgjengelige til filen var lest. Da kunne også lesingen være automatisk ved oppstart, men kanskje vil brukeren *ikke* lese, bare vise statistikk og/eller grafikken. Man kan legge inn knapperad, men det synes unødvendig med så få funksjoner.

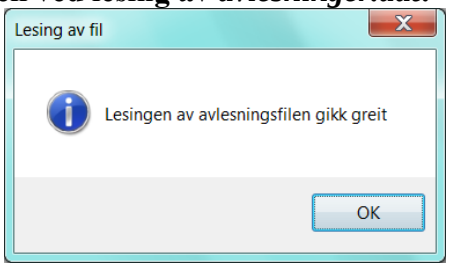
2. Meldingsboks som viser objektene i RAM:



Note: Denne egner seg dårlig når det er mange objekter, men da vil statistikkskjema vise det samme.

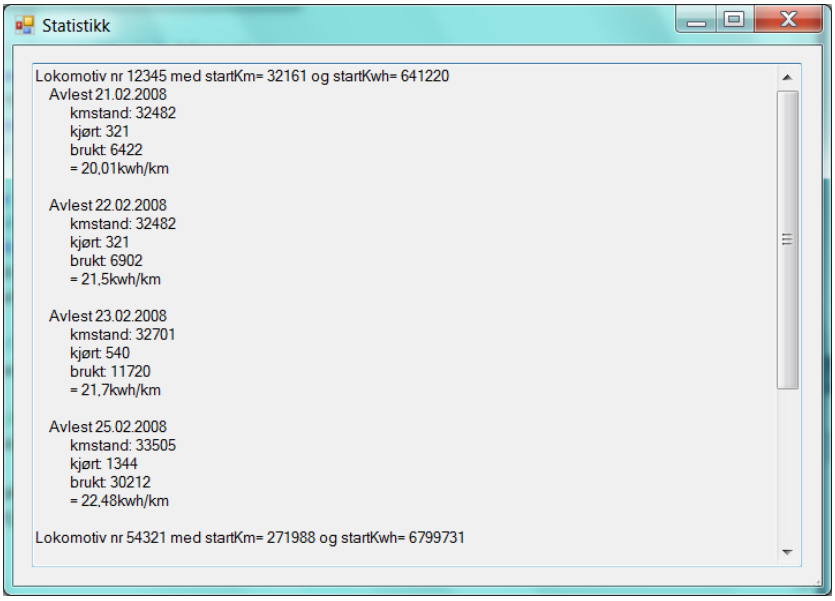
3. Meldingsboks som viser antall feil ved lesing av *avlesninger.dat*:





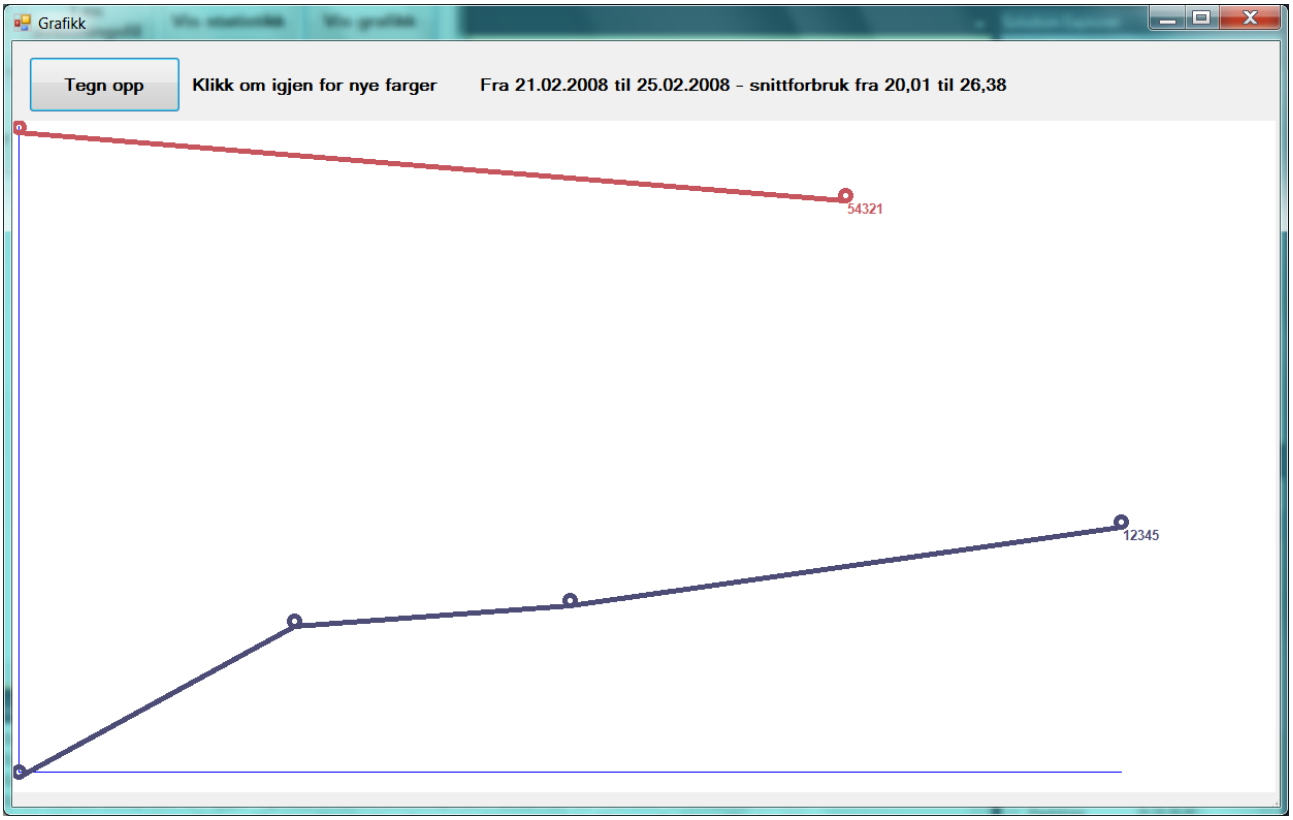
Meldingen til venstre vises hvis det oppsto feil, den til høyre vises hvis alt går greit.

4. Skjema for statistikk:



Tekstboksen er *ReadOnly=True* så brukeren ikke kan endre teksten, bare lese den. Bør unngå *Enabled=False* både fordi skriften blir svak, men også fordi brukeren kanskje vil merke/kopiere til et dokument.

5. Skjema for grafisk fremstilling av dataene:



Note: Maks/min snittforbruk og maks/min dato brukes til å få plass til alle dataene i skjemaet. Siden vi ikke vet hvor mange lokomotiver som skal vises, brukes tilfeldige farger på grafene. Grafikken vises ikke hvis den tegnes under *Load* og den forsvinner hvis skjemaet dekkes over eller minimeres. Derfor er det laget en knapp her, som tegner opp ved klikk. Da kan også brukeren klikke en gang til hvis fargene ikke er bra.

Teknisk ønsker man at alt som er generelt legges i et klassebibliotek, og at de enkelte skjemaene bare kaller på klasser, metoder og data i biblioteket.

Biblioteket StrømLib.vb

Lagret og kompilert som separat, dokumentert prosjekt.

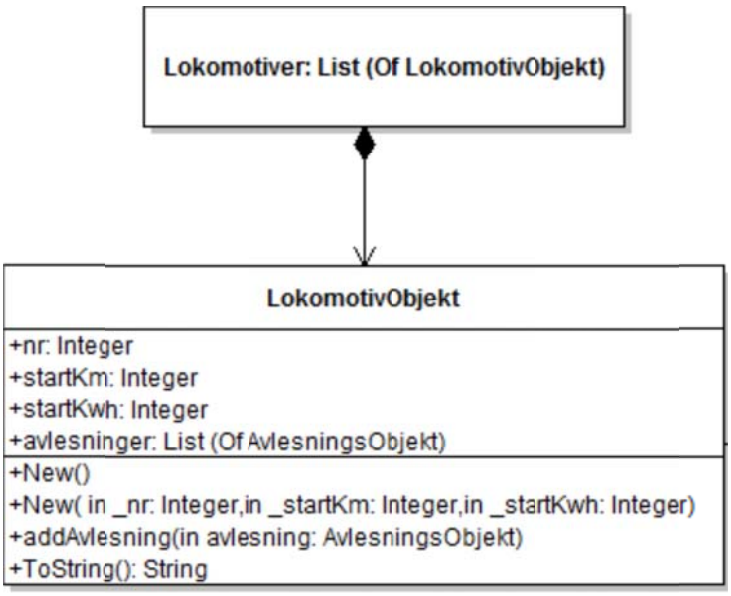
Generelt

Felles variable, funksjoner og prosedyrer deklareres i en *Module*. Da blir de direkte tilgjengelige i hovedprogrammet.

Alt som skal brukes i det andre programmet, må deklareres *Public*. Hvis det er *Private* er det bare synlig innenfor biblioteket.

Jeg lager følgende:

1) Klasse for **lokomotiv** (i egen fil i biblioteket), med
   a) attributter (inkludert en liste med referanser til lokomotivets avlesninger)
   b) konstruktøren *New* med parametre tilsvarende alle de enkle attributtene.
   c) overstyring av funksjonen *ToString*
   d) metode for å legge til en ny avlesning
2) Klasse for **avlesninger** (i egen fil i biblioteket) med i alle fall
   a) attributtene (inkludert en referanse til lokomotivet som avlesningen gjelder)
   b) konstruktør behøves ikke - vi kan bruke den som automatisk følger med (uten parametre)
   c) overstyring av funksjonen *ToString*
   d) muligheter for at avlesningen selv beregner gjennomsnittlig strømforbruk pr kilometer – da trengs også antall kjørte kilometer siden undersøkelsen startet og tilsvarende for strømforbruk. Disse kan lages som funksjoner, men jeg bruker egenskaper.
3) **Metoder** (i modulen) som det opplagt vil være bruk for:
   a) Lese alle poster fra databasen og brygge dem opp som objekter i RAM
   b) Lese avlesningsfilen og
      i) lagre avlesningen i databasen
      ii) generere avlesningsobjekter i RAM (dette gjøres bare hvis lagringen i databasen gikk bra, da databasen foretar integritetskontroller)
      iii) legge avlesningsobjektet til på det aktuelle lokomotivet
      iv) skrive evt feil til feilfilen
4) **Andre metoder** (i modulen) som kan være nyttige:
   a) Finne et lokomotiv i lokomotivlisten når lokomotivets nummer er kjent
   b) Generere statistikk
   c) Finne antall avlesninger i databasen for hvert lokomotiv
   d) Vise alle lokomotiver og avlesninger i RAM (til debugging)
   e) Finne minste og største gjennomsnittsforbruk totalt (til å få riktig skala på grafikken)



Liste over lokomotiver (i RAM)

Skaper en liste over lokomotiver og tilkoblingsstreng (skal brukes flere steder):

```
Public lokomotiver As New List(Of LokomotivObjekt)
Private tilkobling As String = _
             "Server=localhost;" & _
             "Database=stromforbruk;" & _
             "Uid=Knut;" & _
             "Pwd=Tunk;" & _
             "Connect Timeout=30;" & _
             "CharSet=utf8;"
```

Klassen lokomotiver

1. Klassen deklareres

```
Public Class LokomotivObjekt
```

2. Det skal være fire attributter (variable). Alle er *Public* og det er ikke helt ”pent”, fordi da må de andre programmene som setter nye verdier, selv passe på at verdien er lovlig. Vnaligvis vil vi deklarere attributter som *Private* og så tilby metoder som henter og endrer dem (sistnevnte foretar da nødvendig kontroll):

```
    Public nr As Integer = 0
    Public startKm As Integer = 0
    Public startKwh As Integer = 0
    Public avlesninger As New List(Of AvlesningsObjekt)
```

3. Vi trenger en konstruktør (som kalles med *New LokomotivObjekt*). Denne konstruktøren har parametre som setter verdi på alle attributtene:

```
    Public Sub New(ByVal nr As Integer, ByVal startKm As Integer, _
            ByVal startKwh As Integer)
        Me.nr = nr
        Me.startKm = startKm
        Me.startKwh = startKwh
    End Sub
```

4. Vi kan ha bruk for å få skrevet ut et lokomotiv, dvs. omgjort det til en streng. Denne finnes automatisk fra før, vi må overstyre den med *Overrides*:

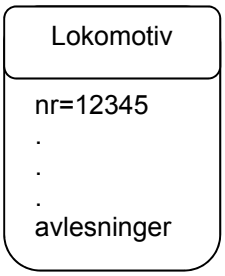
```
Public Overrides Function ToString() As String
    Dim tekst As String
    tekst = nr.ToString & "," & startKm.ToString & "," _
        & startKwh.ToString
    For Each avlesning As AvlesningsObjekt In avlesninger
        tekst &= vbNewLine & "    " & avlesning.ToString
    Next
    Return tekst
End Function
```

5. Lokomotivobjektene har en liste over "sine" avlesninger. Vi må kunne legge til nye. Her foretas det litt primitiv inputkontroll. Listen over avlesninger holdes sortert.

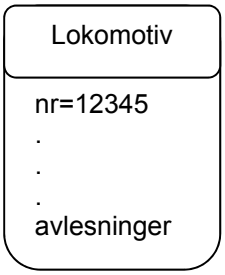
```
Public Sub addAvlesning(ByVal avlesning As AvlesningsObjekt)
    If avlesning.km > Me.startKm And avlesning.kwh > Me.startKwh Then
        'Legg inn sortert - finn først riktig plass
        Dim i As Integer = 0
        Do While i < avlesninger.Count AndAlso _
            avlesning.dato > avlesninger(i).dato
            i += 1
        Loop
        'Plassen er funnet - legg inn her
        avlesninger.Insert(i, avlesning)
    End If
End Sub
```

Forklaring: Legge inn ny avlesning, sortert i listen

While-setningen er i to ledd med *AndAlso*. Det innebærer at evalueringen av uttrykket skal avsluttes så snart som mulig – såkalt "short-circuiting evaluation". Hvis første ledd (*i < avlesninger.Count*) er *False* avbrytes evalueringen, for da er det allerede klart at hele uttrykket er *False* (*False And x* er alltid *False*, uansett hva *x* er). Dermed unngår vi at programmet forsøker å evaluere *avlesninger(i)* når *i >Count* som ville gitt feil (fordi *avlesninger(Count)* ikke eksisterer).

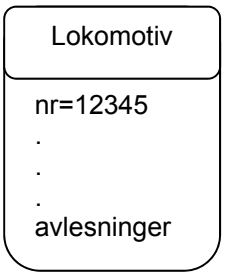


Ingen er lagt inn. *Count=0*



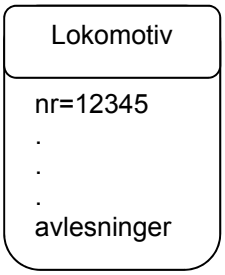
Legg inn 22.2. *Count=0*
i=0 er ikke < Count så *i* blir 0.
insert(0) setter posten forrest i listen.

22.2.2010…



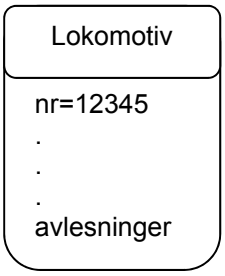
Legg inn 21.2. *Count=1*
i=0 er < *Count* men *21.2* er ikke > 22.2 så *i* blir 0.
Insert(0) setter posten forrest i listen.

21.2.2010… 22.2.20108



Legg inn 23.2. *Count=2*
i=0 er <*Count* og *23.2>21.2* så *i* økes til 1
i=1 er <*Count* og *23.2>22.2* så *i* økes til 2
i=2 er ikke <*Count* så *i* blir 2.
insert(2) setter posten bakerst.

21.2.2010… 22.2.2010… 23.2.2010…



Legg inn 25.2. *Count=3*
Som for forrige post. *i* blir *3* og *insert(3)* setter inn posten bakerst.

21.2.2010… 22.2.2010… 23.2.2010… 25.2.2010…

Klassen AvlesningsObjekt

1. Deklarer klassen

```
Public Class AvlesningsObjekt
```

2. Klassen skal ha fire attributter. Det siste er en referanse til det lokomotivet som avlesning gjelder

```
    Public dato As DateTime = DateTime.FromBinary(0)
    Public km As Integer = 0
    Public kwh As Integer = 0
    Public lokomotiv As LokomotivObjekt = Nothing
```

3. Vi trenger å få vite hvor langt lokomotiver har kjørt siden undersøkelsen startet. Det kan vi gjøre som en *funksjon*, men her viser jeg hvordan man kan legge til en ny *egenskap* (*property*). Når noen vil lese denne egenskapen, brukes metoden *Get*, hvis de vil endre den brukes metoden *Set*. F.eks. vil et program som skriver

```
a = avlesning.kjørtKm
```

hente verdien (med *Get*), mens et program som skriver

```
avlesning.kjørtKm = 150
```

vil *endre* verdien (med *Set*).
Her skal det ikke være mulig å endre verdien, og derfor merkes den *ReadOnly*. Da er det bare tillatt å definere metoden *Get*. Legg merke til at jeg beregner verdien med eget attributt (*km*) og ved å hente attributt fra et annet objekt (*lokomitv.startKm*). Dette kalles et *avledet* attributt, fordi det beregnes ved behov og lagres ikke som attributt i objektet. Da kan den heller ikke gis verdi og derfor bare *get*.

```
    Public ReadOnly Property kjørtKm As Integer
        Get
            Return km - lokomotiv.startKm
        End Get
    End Property
```

4. Helt tisvarende trengs strømforbruket siden start og snittforbruket.

```
    Public ReadOnly Property bruktKwh As Integer
        Get
            Return kwh - lokomotiv.startKwh
        End Get
    End Property

    Public ReadOnly Property snittforbruk() As Double
        Get
            Return Math.Round(bruktKwh / kjørtKm, 2)
        End Get
    End Property
```

5. Konstruktøren må også lages (bare konstruktør uten argumenter lages automatisk hvis det ikke er definert noen annen konstruktør):

```
    Public Sub New(ByVal lokomotiv As LokomotivObjekt, _
                   ByVal dato As DateTime, ByVal km As Integer, _
                   ByVal kwh As Integer)
        Me.lokomotiv = lokomotiv
        Me.dato = dato
        Me.km = km
        Me.kwh = kwh
    End Sub
```

6. Vi trenger å lage en bedre *ToString*:

```
    Public Overrides Function ToString() As String
        Return dato.ToShortDateString & _
            "," & km.ToString() & "," & kwh.ToString
    End Function
```

Dermed er avlesningsklassen på plass.

Prosedyren visdata i hovedmodulen

Denne skal vise en meldingsboks med opplysninger om alle lokomotiver i RAM i øyeblikket. Fin til feilfinning.

```
    Public Sub visdata()
        'DEBUG: Vis hva vi fant
        Dim tekst As String = ""
        For Each lokomotiv As LokomotivObjekt In lokomotiver
            tekst &= lokomotiv.ToString & vbNewLine
        Next
        MsgBox(tekst, MsgBoxStyle.Information, _
               "DEBUG: Tog og avlesninger i RAM nå")
    End Sub
```

Legg merke til at jeg bruker *For-Each* – det passer jo bra når jeg skal gå igjennom en samling.

Funksjonen lesfil i hovedmodulen
Denne skal lese avlesningsfilen (navnet oppgis som parameter). Den bygger opp nye avlesningsobjekter i RAM, legger dem til riktig lokomotiv og lagrer dem i databasen. Hvis det oppstår feil, fanges de og feilmeldingen skrives til egen feilfil (oppgitt som parameter).

Den er laget som en *funksjon* og returnerer antall feil som oppstår. Funksjonen har feilfelle i fire nivåer for å få frem riktig feilmelding.

```
    Public Function lesfil(ByVal avlesningsfil As String, _
                           ByVal feilfil As String) As Integer
        Dim antFeil As Integer = 0
        Try 'Åpne filer
            Dim leser As New IO.StreamReader _
                (My.Application.Info.DirectoryPath & "\" & avlesningsfil)
            Dim feilskriver As New IO.StreamWriter _
                (My.Application.Info.DirectoryPath & "\" & feilfil, False)
            Dim post As String = ""
            Dim data() As String
            Try 'lese filen
                While Not leser.EndOfStream
                    post = leser.ReadLine()
                    data = post.Split(","c)
                    Try 'legge inn og lagre ny avlesning
                        'Finn riktig lokomotiv
                        Dim lokomotiv As LokomotivObjekt
                        lokomotiv = finnLok(CInt(data(1)))
                        If lokomotiv Is Nothing Then 'inner ikke lokomotivet
                            feilskriver.WriteLine(post & _
                                " - Finner ikke lokomotivet")
                            antFeil += 1
                        Else
                            'Skap nytt Avlesningsobjekt
                            Dim nyAvlesning As New AvlesningsObjekt _
                                 (lokomotiv, CDate(data(0)), CInt(data(2)), _
                                  CInt(data(3)))
                            Try 'lagre avlesningen i databasen
                                'Prøv  lagre:
                                lagre(nyAvlesning)
                                'Legg til objektet hvis lagringen gikk bra
                                '(ellers har databasefeilen gitt uthopp):
                                lokomotiv.addAvlesning(nyAvlesning)
                            Catch ex As Exception 'klarte ikke å lagre
                                feilskriver.WriteLine(post & " - " & ex.Message)
                                antFeil += 1
                            End Try
                        End If
                    Catch ex As Exception 'klarte ikke å legge inn/lagre
                        feilskriver.WriteLine(post & " - " & ex.Message)
                        antFeil += 1
                    End Try
                End While
            Catch ex As Exception 'feil i lesingen
                feilskriver.WriteLine(post & " - " & ex.Message)
                antFeil += 1
            End Try
            leser.Close() 'OK, selvom den ikke er åpen
            feilskriver.Close() 'OK selvom den ikke er åpen
        Catch ex As Exception
            Throw New Exception("Klarer ikke å åpne filene" & vbNewLine & ex.Message)
        End Try
        Return antFeil
    End Function
```

Prosedyren lagre i hovedmodulen

Denne forsøker å lagre en oppgitt avlesning (parameter) i databasen. Det kan gå galt, og da kastes en feil som den kallende metoden må håndtere. Alternativt kunne prosedyren lagre håndtert feilen selv (*Try-Catch*) men da måtte den også fått oppgitt navnet på feilfilen.

Prosedyren er litt treg, fordi den åpner og lukker databasen. Legg merke til buken av *With* som forenkler skrivingen. Legg også merke til at datoen må formatteres annerledes enn VB gjør det insert-setningen. Her skal vi ikke få svar fra databasen (men kommandoobjektet kan motta feil som den da kaster), derfor bruker jeg *ExecuteNonQuery*.

```
Private Sub lagre(ByVal avlesning As AvlesningsObjekt)
    'Feil som kastes blir håndtert av kallende prosedyre lesfil og skrives
    'til egen feilfil.
    '0: Deklarasjoner
    Dim minConnection As New MySqlConnection()
    Dim minCommand As MySqlCommand

    '1 Sett connection
    minConnection.ConnectionString = tilkobling
    Try
        '2 Åpne connection
        minConnection.Open()

        '3 Sett command
        minCommand = minConnection.CreateCommand()
        With avlesning
            minCommand.CommandText = _
                "INSERT INTO Avlesning (dato, loknr, km, kwh) VALUES (" _
                & "'" & Format(.dato, "yyyy.MM.dd") & "'," _
                & .lokomotiv.nr.ToString _
                & "," & .km.ToString & "," & .kwh & ");"
        End With

        '4 Utfør command
        minCommand.ExecuteNonQuery()
    Catch ex As Exception
        Throw New Exception("Klarer ikke å lagre" & vbNewLine & ex.Message)
    Finally
        '5 Lukk connection
        minConnection.Close() 'går bra selvom cofnnection ikke ble åpnet
    End Try
End Sub
```

Prosedyren lesbasen i hovedmodulen

Prosedyren leser alle postene i databasen (to tabeller – databasen er oppgitt som parameter) og generer alle objektene i RAM. Databasen åpnes og lukkes igjen.

Det lages én leser for lokomotiver og én for avlesninger. Først leses et lokomotiv, deretter leses alle avlesninger som gjelder dette lokomotivet. De to leserne må knyttes til hver sin *connection*, for en *connection* kan bare være knyttet til én leser/adapter av gangen.

Merk rekkefølgen her:

1) Skap to connections, sett deres connectionstring og åpne dem
2) Skap to kommandoer knyttet til disse connections og sett commandtext.
3) Skap to lesere knyttet til hver sin kommando
4) Les ett og ett lokomotiv med leserens *read*. For hvert lokomotiv: Les alle avlesninger knyttet til dette lokomotivet
5) Lukk leserne og connections

```
Public Function lesbasen() As String
    '0: Deklarasjoner
    Dim lokConnection As New MySqlConnection()
    Dim avlesConnection As New MySqlConnection()
    Dim lokCommand As MySqlCommand
    Dim avlesCommand As MySqlCommand
    Dim lokLeser As MySqlDataReader
    Dim avlesLeser As MySqlDataReader
    Dim nyttLokomotiv As LokomotivObjekt
    Dim nyAvlesning As AvlesningsObjekt

    Try
        '1 Sett connection
        lokConnection.ConnectionString = tilkobling
        avlesConnection.ConnectionString = tilkobling

        '2 Åpne connection
        lokConnection.Open()
        avlesConnection.Open()

        '3 Sett lokCommand
        lokCommand = lokConnection.CreateCommand()
        lokCommand.CommandText = _
            "SELECT nr, startKm, startKwh FROM Lokomotiv order by nr;"
        avlesCommand = avlesConnection.CreateCommand()

        '4 Skap lokLeser
        lokLeser = lokCommand.ExecuteReader()

        Do While lokLeser.Read
            '5 Les neste lokomotiv
            nyttLokomotiv = New LokomotivObjekt( _
                CInt(lokLeser.Item("nr")), CInt(lokLeser.Item("startKm")), _
                CInt(lokLeser.Item("startKwh")))
            lokomotiver.Add(nyttLokomotiv)
            'Skap avlesCommand
            avlesCommand.CommandText = _
                "SELECT dato, loknr, km, kwh FROM Avlesning " _
                & "WHERE loknr = " & nyttLokomotiv.nr.ToString & _
                " ORDER BY loknr, dato;"
            'Skap avlesLeser
            avlesLeser = avlesCommand.ExecuteReader()

            '6 Les tilhørende avlesninger
            Do While avlesLeser.Read()
                nyAvlesning = New AvlesningsObjekt(nyttLokomotiv, _
                    CDate(avlesLeser.Item("dato")), _
                    CInt(avlesLeser.Item("km")), _
                    CInt(avlesLeser.Item("kwh")))
                nyttLokomotiv.addAvlesning(nyAvlesning)

            Loop
            'Lukk avlesLeser
            avlesLeser.Close()
        Loop

        '7 Lukk lokLeser og connection
        lokLeser.Close()
        lokConnection.Close()
        avlesConnection.Close()
    Catch ex As Exception
        Throw New Exception("Feil ved lesing av databasen" & vbNewLine & _
            ex.Message)
    End Try
```

```
        'Hvis hvor mange avlesninger det er i databasen for hvert lokomotiv
        Dim tekst As String = "Antall avlesninger pr lokomotiv i databasen" _
            & vbNewLine
        For Each lok As LokomotivObjekt In lokomotiver
            tekst &= lok.nr.ToString & ": " & _
            antAvlesninger(lok.nr).ToString & vbNewLine
        Next
        Return tekst
    End Sub
```

While-løkkene avslutter når *read* blir *False* (da finnes det ikke flere poster å lese).

Funksjonen antAvlesninger i hovedmodulen

Funksjonen beregner (henter fra databasen) antall avlesninger i databasen for et oppgitt lokomotiv.

```
    Private Function antAvlesninger(ByVal loknr As Integer) As Integer
        '0 Deklarasjoner
        Dim minConnection As New MySqlConnection()
        Dim antallCommand As MySqlCommand
        Dim antall As Integer

        Try
            '1 Sett connection
            minConnection.ConnectionString = tilkobling
            Try
                '2 Åpne connection
                minConnection.Open()

                '3 Sett lokCommand
                antallCommand = minConnection.CreateCommand()
                antallCommand.CommandText = _
                    "SELECT count(*) FROM Avlesning where loknr=" _
                    & loknr.ToString & ";"

                '4 Utfør spørringen som skalar,
                'dvs returnerer bare første kolonne i første post
                antall = CInt(antallCommand.ExecuteScalar)
            Catch ex As Exception
                Throw (ex)
            Finally
                '5 Lukk  connection
                minConnection.Close() 'OK selvom den ikke ble åpnet
            End Try
            '6 Returner resultatet
            Return antall
        Catch ex As Exception
            Throw New Exception("Feil ved lesing av databasen" & vbNewLine & _
                ex.Message)
        End Try
    End Function
```

Her skal det returneres bare én verdi, og derfor bruker jeg *ExecuteScalar*. Funksjonen burde returnert et heltall selv om den feiler og kommer til *Catch*, men vi må velge om den skal hoppe ut med feil, eller med en verdi f.eks. -1. (Dette godtar kompilatoren, som ellers nøye kontrollerer at funksjonen returnerer en verdi uansett.)

Funksjonen finnLok i hovedmodulen

Denne funksjonen skal finne riktig lokomotiv i lokomotivlisten, utfra det oppgitte nummeret. Hvis lokomotivet ikke fins, så returneres *Nothing*: Alternativt kunne funksjonen ha kastet en feil.

```
Private Function finnLok(ByVal nr As Integer) As LokomotivObjekt
    For Each lok As LokomotivObjekt In lokomotiver
        If lok.nr = nr Then Return lok
    Next
    Return Nothing
End Function
```

Funksjonen hentStatistikk I hovedmodulen

Funksjonen henter statistikk for alle lokomotiver i RAM. Den returnerer en streng med data om lokomotivet, data om lokomotiver og alle lokomotivets avlesninger, med dato, kilometerstand, avlest målerstand, kjørte km siden undersøkelsen startet, avlest målerstand, forbruk siden undersøkelsen startet og snittforbruket siden undersøkelsen startet i kwh/km. Strengen kan vises i en meldingsboks ller på et skjema.

```
    Public Function hentStatistikk() As String
        'Viser statistikk for hvert tog
        Dim tekst As String = ""
        For Each lokomotiv As LokomotivObjekt In lokomotiver
            'Overskrift:
            tekst &= "Lokomotiv nr " & lokomotiv.nr & _
            " med startKm= " & lokomotiv.startKm & _
            " og startKwh= " & lokomotiv.startKwh & vbNewLine
            'Avlesninger:
            For Each avlesning As AvlesningsObjekt In lokomotiv.avlesninger
                With avlesning
                    tekst &= "    Avlest " & .dato & vbNewLine _
                    & "          kmstand: " & .km & vbNewLine _
                    & "          kjørt: " & .kjørtKm & vbNewLine _
                    & "          brukt: " & .bruktKwh & vbNewLine _
                    & "          = " & .snittforbruk & "kwh/km" & vbNewLine _
                    & vbNewLine
                End With
            Next
        Next
        Return tekst
    End Function
```

Hovedprogrammet, altså skjemaene

Hovedprogrammet programmeres for seg selv. Det har tre skjemaer: Hovedskjema med knapper, et skjema for statistikk og et for grafikk. Prosjektet er knyttet til biblioteket *Strømlib* (*references*) og alle tre importer *Strømlib*, så slipper man å skrive *Strømlib* foran alle henvisninger til biblioteket.

1. Hovedskjemaet frmStrømforbruk

Hovedskjemaet blir relativt enkelt, da det benytter biblioteket til det meste.

Ved oppstart leses postene fra databasen og en enkel melding av hva som ble funnet vises for debugging. *Lesbasen()* kan kaste feil.

```
    Private Sub Strømforbruk_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        Try
            MsgBox(lesbasen(), MsgBoxStyle.Information, "Resultat av lesing av datasen")
        Catch ex As Exception
            MsgBox(ex.Message, MsgBoxStyle.Critical, "Feil ved lesing av databasen")
        End Try
        visdata() 'DEBUG
    End Sub
```

Knappene for statistikk og grafikk skal bare vise et annet skjema. De vises som dialoger, slik at brukeren må tilbake til hovedskjemaet for å avslutte. De to subskjemaene håndterer det nødvendige når de lastes.

```
    Private Sub butStatistikk_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butStatistikk.Click
        frmStatistikk.ShowDialog()
    End Sub

    Private Sub butGrafikk_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butGrafikk.Click
        frmGrafikk.ShowDialog()
    End Sub
```

Knappen ”les avlesningsfil” skal lese avlesningsfilen. Det overlates til biblioteket som også vil lagre avlesningen i databasen og bygge opp objektene i RAM. Biblioteksfunksjonen *lesfil* returnerer antall feil ved lesingen, som vi kan oppgi til brukeren i meldingsboks.

```
        Dim antfeil As Integer
        Try
            antfeil = lesfil("avlesninger.dat", "feilavlesninger.dat")
            If antfeil > 0 Then
                Throw New Exception("Det oppsto " & antfeil.ToString & _
                       " feil ved lesing av filen." & vbNewLine & _
                       "(Se evt. filen ""feilavlesninger.dat"")")
            Else
                MsgBox("Lesingen av avlesningsfilen gikk greit", _
                    MsgBoxStyle.Information, "Lesing av fil")
            End If
            visdata() 'DEBUG
        Catch ex As Exception
            MsgBox(ex.Message, MsgBoxStyle.Critical, "Feil ved lesing av avlesningsfilen")
        End Try
```

2. Statistikkskjema frmStatistikk.vb

Dette skjemaet har en stor tekstboks med flere linjer. Under lasting av skjemaet, hentes all statistikk med bibliotekets *hentStatistikk* som returner all statistikken i én streng. Den vises i tekstboksen.

Tekstboksen settes *ReadOnly* så ikke brukeren kan endre teksten.

```
    Private Sub frmStatistikk_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        txtStatistikk.Text = hentStatistikk()
        txtStatistikk.ReadOnly = True 'så ikke brukeren kan gjøre endringer
    End Sub
```

Når teksten vises i tekstboksen, blir den også markert. Det kan man ikke gjøre noe med før skjemaet er ferdig lastet, dvs. hendelsen *Shown*:

```
    Private Sub frmStatistikk_Shown(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Shown
        txtStatistikk.DeselectAll() 'txtStatistikk er merket ved start
    End Sub
```

3. Grafskjema frmGrafikk.vb

Her skal det grafes, og det er noe mer komplisert.

Skjemaet har en stor picturebox til å tegne i. Det må gjøres grafisk, og det trengs litt innvendig kant så grafikken kommer inne i pictureboxen:

```
    Dim grafikk As Graphics
    'Rammer for grafikken
    Dim venstre As Integer
    Dim høyre As Integer
    Dim topp As Integer
    Dim bunn As Integer
```

Hvis noe legger seg over vinduet, eller det minimeres, blir grafikken borte. Grafikken kan heller ikke tegnes før skjemat er vist. Derfor har jeg laget en knapp *butTegn* som tegner opp grafikken.

```
    Private Sub butTegn_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles butTegn.Click
```

Litt houskeeping: Grafikken skapes og indre rammer får verdi (fremkommet ved eksperimentering og hardkodet her). Aksene tegnes.

```
grafikk = picGrafikk.CreateGraphics()
venstre = 5
høyre = picGrafikk.Width - 150
topp = 5
bunn = picGrafikk.Height - 20
'Tegn akser
grafikk.DrawLine(Pens.Blue, venstre, topp, venstre, bunn)
grafikk.DrawLine(Pens.Blue, venstre, bunn, høyre, bunn)
```

For å kunne tegne grafene innenfor aksene, må vi vite hva som er minste/største dato og minste/største snittverdi. Jeg bruker vanlig posit-admit-teknikk. Dette har bare mening hvis det finnes registrerte lokomotiver med avlesning (jeg sjekker – for enkelt – at bare første lokomotiv ha avlesning):

```
'A Finn minst/størst dato og minst/størst snittforbruk
'Posit:
If lokomotiver.Count > 0 AndAlso lokomotiver(0).avlesninger.Count > 0 _
        Then 'det finnes et lokomotiv og det har avlesninger
    Dim minDato As DateTime = lokomotiver(0).avlesninger(0).dato
    Dim minSnitt As Double = lokomotiver(0).avlesninger(0).snittforbruk
    Dim maksDato As DateTime = lokomotiver(0).avlesninger(0).dato
    Dim maksSnitt As Double = lokomotiver(0).avlesninger(0).snittforbruk
    For Each lokomotiv As LokomotivObjekt In lokomotiver
        For Each avlesning As AvlesningsObjekt In lokomotiv.avlesninger
            'Admit:
            If avlesning.dato < minDato Then minDato = avlesning.dato
            If avlesning.dato > maksDato Then maksDato = avlesning.dato
            If avlesning.snittforbruk < minSnitt Then _
                minSnitt = avlesning.snittforbruk
            If avlesning.snittforbruk > maksSnitt Then _
                maksSnitt = avlesning.snittforbruk
        Next
    Next
    'B Vis minste/største
    lblData.Text = "Fra " & minDato & " til " & maksDato & _
        " - snittforbruk fra " & minSnitt & " til " & maksSnitt
```

Beregner så skalaen horisontalt (avstanden minste til største dato i dager skal tilsvare det som er tilgjengelig), og vertikalt (tilsvarende for minst/største snitt):

```
    'B Beregn avstander pr enhet
    Dim horisontalt As Integer = (høyre - venstre) _
        \ (maksDato - minDato).Days
    Dim vertikalt As Integer = CInt((bunn - topp) _
        / (maksSnitt - minSnitt))
```

Jeg er nå klar til å plotte. Hver verdi er gitt som et punkt, med x-verdi lik antall dager siden minste dato, og y-verdi lik snittet minus minste snitt. Begge korrigeres for skalaen. Først noen variable for hvert lokomotiv:

```
    For Each lokomotiv As LokomotivObjekt In lokomotiver
        Dim minpenn As New Pen(Color.FromArgb(CInt(Rnd() _
            * &HFFFFFF + &HFF000000)),5) 'tilfeldig farge, bredde 5
        Dim minbrush As New SolidBrush(minpenn.Color)
        Dim minfont As New Font("Arial", 8, FontStyle.Bold)
        Dim x, y As Integer
        Dim punkter As New List(Of Point)
        Dim avlesning As AvlesningsObjekt
```

Fargen blir tilfeldig i intervallet &H000000 til &HFFFFFF. Jeg legger til &HFF000000 for å angi første del (A) som bestemmer transparens.

Jeg går igjennom lokomotivets avlesninger og legger til punkter i listen min. Det tegnes en liten ring rundt hvert punkt:

```
                For Each avlesning In lokomotiv.avlesninger
                    x = (avlesning.dato - minDato).Days * horisontalt + venstre
                    y = bunn - CInt((avlesning.snittforbruk - minSnitt) _
                        * vertikalt)
                    punkter.Add(New Point(x, y + 5))
                    grafikk.DrawEllipse(minpenn, x - 5, y - 5, 10, 10)
                Next
```

Alle plottpunktene for dette lokomotiver er funnet og en sirkel tegnet rundt hver – nå trekkes jeg rette linjer fra punkt til punkt. Her skapes det en array med punkter (x,y) Det trekkes rette linjer mellom alle punktene i arrayen (*DrawLines* trekker mange linjer og krever en array som argument) og lokomotivnummeret skrives ved det siste punktet (siden dette er en grafisk flate, må også tekst ”tegnes”):

```
                'Trekker kontinuerlig linje mellom alle punktene:
                grafikk.DrawLines(minpenn, punkter.ToArray)
                'Skriver tekst
                grafikk.DrawString(lokomotiv.nr.ToString, minfont, _
                    minbrush, punkter(punkter.Count - 1))
            Next
        End If
    End Sub
End Class
```

Demo: Finne største snitt rekursivt og minste snitt iterativt

Det er to funksjoner i biblioteket som ikke er brukt i mitt program. De er tatt med som demonstrasjon av iterativ kontra rekursiv søking etter støreste/minste i en liste. Helt tilsvarende kan gjøres i en array.

A: Iterativt

```
Public Function minstSnitt _
            (ByVal avlesninger As List(Of AvlesningsObjekt)) As Double
    'Finner minste snittforbruk iterativt
    If avlesninger.Count = 0 Then Return -1 'ingen avlesninger
    Dim minst As Double = avlesninger(0).snittforbruk 'posit
    For Each avlesning As AvlesningsObjekt In avlesninger
        If avlesning.snittforbruk < minst Then _
                minst = avlesning.snittforbruk 'admit
    Next
    Return minst
End Function
```

Denne funksjonen får overført en liste med avlesninger. Den bruker den vanlige "posit-admit" teknikken.

1) Hvis listen er tom, kan vi ikke finne minste, og returnerer -1 som et signal.
2) Hvis det er minst én avlesning i liste, gjetter jeg først at den første er minst.
3) Deretter går jeg igjennom alle, og hvis jeg finner en mindre, så innrømmer jeg at jeg tok feil, og tar vare på den nye, minste verdien.
4) Den minste jeg fant returneres.

B: Rekursivt

```
Public Function størstSnitt _
            (ByVal avlesninger As List(Of AvlesningsObjekt)) As Double
    'Finner største snittforbruk rekursivt
    If avlesninger.Count = 0 Then Return -1 'ingen avlesninger
    Dim første As AvlesningsObjekt = avlesninger(0)
    If avlesninger.Count = 1 Then Return første.snittforbruk 'bare én
    Dim resten As List(Of AvlesningsObjekt) _
            = avlesninger.GetRange(1, avlesninger.Count - 1)
    'Returner enten den første, eller den største av resten
    If første.snittforbruk > størstSnitt(resten) Then
        Return første.snittforbruk
    Else
        Return størstSnitt(resten)
    End If
End Function
```

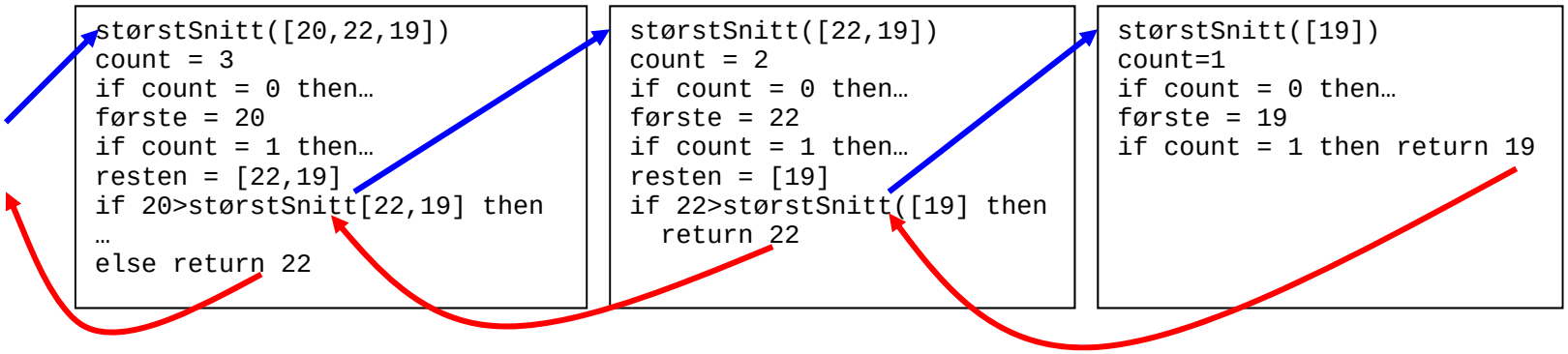
Funksjonen får overført en liste med avlesninger. Den finner største rekursivt.

1) Hvis listen er tom, kan vi ikke finne største og returnerer -1 som et signal.
2) Hvis det bare er én avlesning i listen, må den nødvendigvis være størst og returneres med én gang.
3) Hvis det er mer enn én avlesning i listen, sammenlikner jeg den første med den største i resten, og returnerer den av dem som er størst.

Resten lages ved at jeg kopierer alle avlesningen fra og med nr 1 (det er den *andre* avlesningen!), men da blir det én avlesning mindre (*Count-1*).

Se figur neste side

.


størstSnitt([20,22,19])
count = 3
if count = 0 then…
første = 20
if count = 1 then…
resten = [22,19]
if 20>størstSnitt[22,19] then
…
else return 22
størstSnitt([22,19])
count = 2
if count = 0 then…
første = 22
if count = 1 then…
resten = [19]
if 22>størstSnitt([19] then
return 22
størstSnitt([19])
count=1
if count = 0 then…
første = 19
if count = 1 then return 19

Høgskolen i Buskerud
Postboks 235
3603 Kongsberg
Telefon: 32 86 95 00

www.hibu.no

ISSN 1893-2398 (online)

HØGSKOLEN
i Buskerud