Uke 8 - Mer om objekter og klasser. Tips Oblig3

3. mars 2008

Arild Waaler
Inst. for informatikk, UiO

Stringer er ordentlige objekter

- String er en klasse i Java-biblioteket, men har en egen spesiell syntaks (skrivemåte) så det ser ut som den er en av de basale typene (som int, double,..).
- Når vi har en string, har vi alså både en peker (den vi deklarerer navnet på) og et stringobjekt.
- String-objekter kan ikke endres (trenger du endrbare tekster, bruk klassen StringBuffer)
- Egen skrivemåte for stringkonstanter:

```
String s = "En fin dag i mai";
```

Er det samme som:

```
String s = new String("En fin dag i mai");
```

- Klassen String inneholder mer enn 50 metoder for konvertering mellom ulike datatyper og tekst, samt tekstsøking.

Konstruktører – startmetoder i klasser

- Når vi lager et objekt av en klasse med **new**, kaller vi egentlig en metode som heter det samme som klassen (derfor parentesen bak klassenavnet).
- Vi får automatisk med en slik konstruktør-metode fra oversetteren dersom vi ikke skriver en slik konstruktør selv. Den vi får automatisk er uten parametere og gjør ingen ting.
- Konstruktører nyttes i all hovedsak til å gi fornuftige startverdier for variable i objektet som dannes.
- De konstruktørene vi skriver kan ha parametere.
- Konstruktørene skal ikke ha noen type foran seg, heller ikke void.
- Vi kan ha flere konstruktører i en klasse, men da må parameterne være ulike i antall eller typen av parametrene

Eksempel Student med en konstruktør

```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

   Student(String navn, Kurs [] k){
     this.navn = navn;
     for (int i = 0; i<k.length; i++ ){
         mineKurs[i] = k[i];
         mineKurs[i].antStudenter++;
     }
   }
```

Eksempel Student med 2 konstruktører

```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

   Student() {
     mineKurs = new Kurs[0];
   }

   Student(String navn, Kurs [] k){
     this.navn = navn;
     for (int i = 0; i<k.length; i++ ){
         mineKurs[i] = k[i];
         mineKurs[i].antStudenter++;
     }
   }
```


this

- Av og til trenger vi en peker til det objektet metoden vi utfører er inne i. Java-ordet **`this`** gir oss alltid det.
- Brukes i to situasjoner:
  - Vi har en konstruktor, og parametrene til denne heter det samme som objekt-variable i objektet. Eks:

```
class A {
   int antall;
   A (int antall ){
       this.antall = antall;
   }
} // end A
.. A apek = new A(12);
```

  - Vi skal kalle en metode i et annet objekt (gjerne av en annen klasse). Da kan vi bruke this for å overføre en parameter til denne metoden om hvilket objekt kallet kom fra.

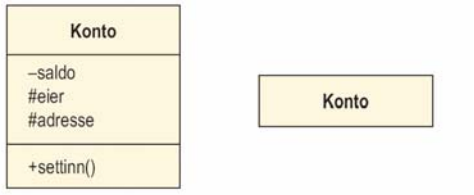
UML-diagrammer av programmene våre

- Hvorfor tegne diagrammer over programmene
  - Oversikt
  - Samarbeid med andre programmererer / systemutviklere
  - Arkitekter, ingeniører tegner først, så bygger de !
  - Enklere å endre en tegning enn programmet
- Objektdiagrammer
- Klassediagrammer
- UML – diagrammene er litt annerledes enn det vi har tegnet hittil (men mye av det samme)

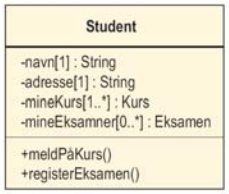
  (i UML er det ca 10 andre diagramtyper vi ikke skal lære)

Klassediagrammer

- En mer kompakt måte enn objektdiagrammer å tegne sammenhengen i programmet
- Skiller seg fra objektdiagrammer ved at vi ikke direkte tegner datastrukturen (pekere og pekerarrayer), men bare forhold (assosiasjoner, forbindelser) mellom klassene.
- I klassediagrammer dokumenterer vi også sentrale metoder.
- Forholdene er linjer med et logisk navn og antall objekter i hver ende
- Anta at vi har laget en class Konto med tre objektvariable: saldo, eier og adresse og en metode: settinn().



Tre (fire) mulig felter i tegning av en klasse



- Navnefeltet (alltid)
  - klassenavnet

Kan utelates:

- Variabelfeltet (attributtene)
  - variabelnavn evt. med type
- Metode-feltet
  - Evt med parametere og returverdi
- (Unntaks-feltet)

Symboler for synlighet
(fra resten av programmet)

\+ public

\- private

\# protected

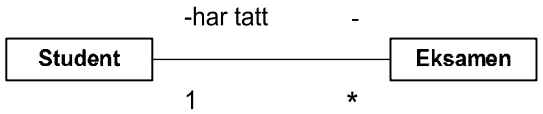
~ package

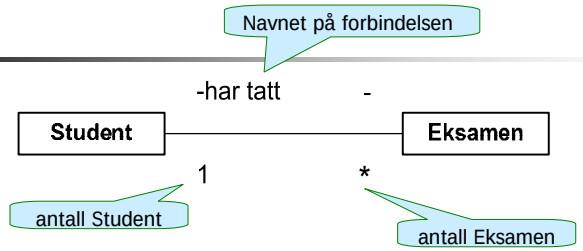
UML Klassediagrammet kan nyttes til

- Modell av problemområdet (domenemodell)
- Modell av klassene i programmet
  (+ modell av databasen,....)
- Men siden vi skal modellere virkeligheten en-til-en i programmet vårt, så blir de like i utgangspunktet

Forhold mellom klasser

- **En student har null eller flere eksamener**
- Vi tegner et forhold mellom to klasser som har med hverandre å gjøre logisk sett, og:
- hvor vi i programmet vil kunne følge pekere for å få adgang til variable eller metoder
- Vi skriver hvor mange objekter det maksimalt på ett tidspunkt kan være på hver side av et slikt forhold
- Siden vi med: Eksamen mener en avlagt enkelt-eksamen vil en Eksamen bare være tilknyttet en bestemt student



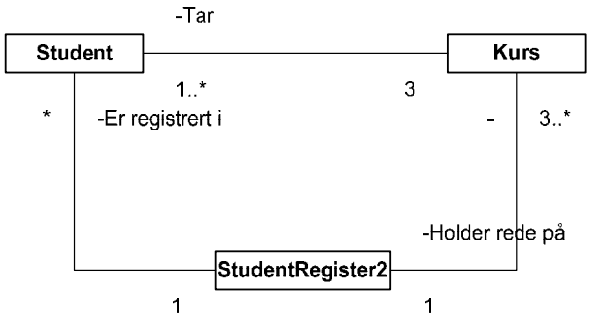


- Forbindelsen leses fra venstre:
  En student har tatt null, en eller flere Eksamener"
- Antallet objekter angis slik:

Skrivemåte	Betydning
1	en
*	null, en eller flere
1..*	minst en
3..*	minst tre
3,4,5	tre, fire eller fem

Studentregister2 – med tillegg: klassen Kurs vet *hvilke* Studenter som tar kurset

- Et studentregister holder orden på studentene og kursene, og en student tar 3 kurs hvert semester



Regler for å plassere riktige antall på et forhold

1. Anta at du står i **ett** objekt av en klasse og ser over til (langs en forbindelse) til en annen klasse:
2. Hvor mange objekter ser du da maksimalt *på et gitt tidspunkt* av den andre klassen
3. Det antallet noteres (jfr. tabellen) på den andre siden
4. Du går så over forbindelsen til den andre klassen og antar at du nå står i **ett** objekt av denne klassen og gjenntar pkt. 1-3

Hvilke forhold skal vi ha med i klassediagrammet ?

- Slike forhold hvor ett objekt av den ene klassen:
  - inneholder
  - består av
  - eier,..

  en eller flere objekter av den andre klassen
- Det vi i programmet vil følge en peker for å få tak i verdien på visse variable i den andre klassen eller kalle en metode.

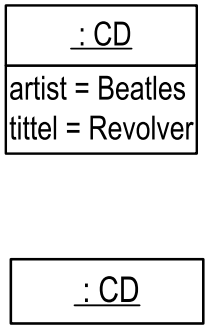
  Det er da ikke 'naturgitt' hvilke forhold vi har i et klassediagram, det avhenger av hvilke spørsmål vi vil være interessert i å svare på.

Objekt-diagrammer

- Vi tegner en typisk situasjon av objekter i systemet vårt, når vi har fått datastrukturen på plass.
- Vi tegner og navngir bare de mest sentrale dataene som:
  - pekere
  - peker-arrayer
  - noen sentrale variable i objektene

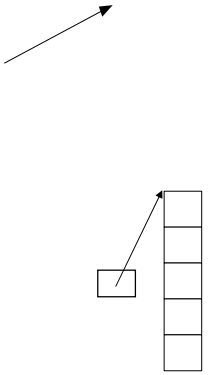
Tegning av et objekt (med mer eller mindre detaljer)

- To eller ett_felt(er) i en boks
- Navnfeltet
  - objektnavn:klassenavn eller bare
  - :klassenavn
- Attributt-feltet (kan være tomt)
  - Navnet på sentrale objektvariable evt. også med verdier



Andre elementer i et objektdiagram

- Pekere
- Peker-arrayer

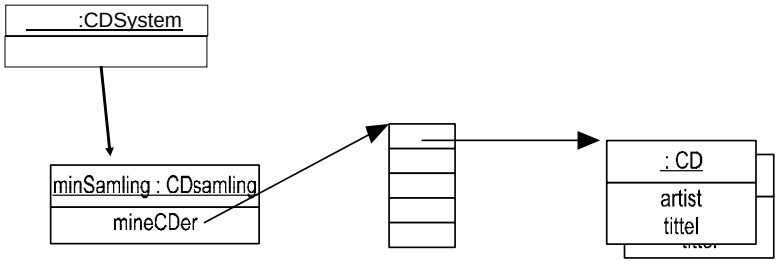


Eksempel: Et CD-samling

- Vi ønsker å lage et lite menystyrt program for å holde orden på CD-samlinga vår med mindre enn 1000 CDer. Vi skal ha funksjoner for å :
  - registrere ny CD
  - Søke etter artist (skriv ut alle platene med denne)
  - Skrive ut hele registeret
- Hvilke klasser har vi i dette problemet
  - Opplagt 'class CD'
  - Noen fler ?

Klassene: CD og CDsamling

- Vi tenker oss følgende datastruktur (er den tilstrekkelig?)
- Vi har her forenklet programmet (klassen CDSystem inneholder main,CDSamling meny-metode og sentral switch i løkke-metoden,... , mens CD inneholder utskriftsrutine .



```
import easyIO.*;

class CDSystem {
   public static void main(String args []) {
      new CDsamling().løkke();
   }
}
class CD{
   String artist, tittel;
   void skrivUt(Out u) {
     u.outln("Artist:" + artist + ", Tittel:" + tittel);
   }
  }

class CDsamling{
  CD [] minSamling = new CD[1000];
  int antCDer = 0;

  void løkke() {
    In tast = new In();
    Out skj = new Out();
    String a;
    CD c;
    int valg;
```

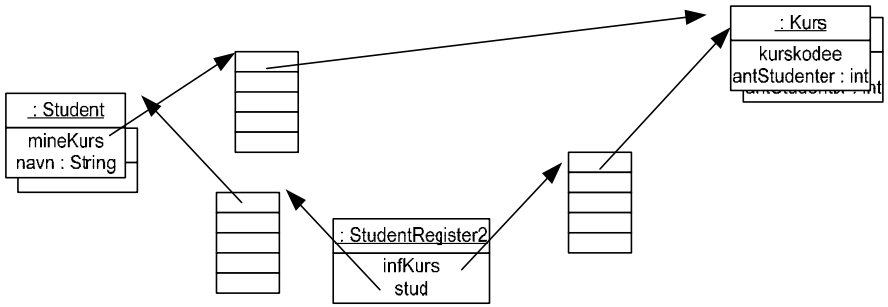
```
do{ skj.outln("Velg:");
    skj.outln(" 1 - les ny plate (skriv artist platetittel");
    skj.outln(" 2 - skriv artist");
    skj.outln(" 3 - avslutt");
    valg = tast.inInt();

    switch(valg) {
       case 1:   // les data
           c = new CD();
           minSamling[antCDer++] = c;
           skj.out("Gi artistnavn:");
           c.artist = tast.inWord();
           skj.out("Gi tittel:");
           c.tittel = tast.inWord();
           break;
       case 2:   // skriv data
           skj.out("Gi artistnavn:");
           a = tast.inWord();
           for(int i = 0; i < antCDer; i++)
            if (minSamling[i].artist.equals(a))
                 minSamling[i].skrivUt(skj);
           break;
       case 3:   // avslutt
            skj.out("Systemet avslutter");
            break;
       default:  //  feil
             skj.out("Bare gi verdier: 1 - 3");
       }
   } while (valg != 3);
}}
```

```
>java CDSystem
Velg:
 1 - les ny plate (skriv artist platetittel
 2 - skriv artist
 3 - avslutt
1
Gi artistnavn:tom
Gi tittel:Fest1
Velg:
 1 - les ny plate (skriv artist platetittel
 2 - skriv artist
 3 - avslutt
1
Gi artistnavn:ola
Gi tittel:FullFres
Velg:
 1 - les ny plate (skriv artist platetittel
 2 - skriv artist
 3 - avslutt
2
Gi artistnavn:tom
Artist:tom, Tittel:Fest1
Velg:
 1 - les ny plate (skriv artist platetittel
 2 - skriv artist
 3 - avslutt
```

Et helt studentregister med kurs, studenter og registeret

- Vi har Studenter på Ifi som første semester tar tre kurs, samtidig som vi har behov for å registrere kurs og hvor mange studenter som tar hvert kurs.
- Vi tegner først en tenkt datastruktur – et UML objektdiagram
- så skriver vi programmet

Objektdiagrammet er en forenkling av programmet. Det tar bare med den essensielle datastrukturen (mest pekere og peker-arrayer) som holder datastrukturen sammen

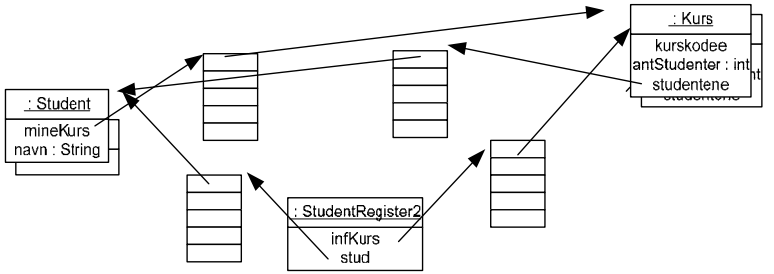


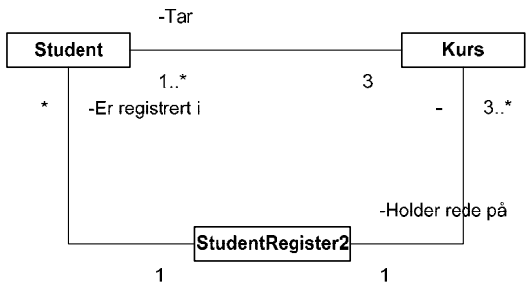

```
class StudentRegister2{
  public static void main(String args []) {
  String [] kurskode = {"INF1000","INF1040","MAT1030"};

  // lag kurs
  Kurs [] infKurs = new Kurs[3];
  for (int i = 0 ; i< infKurs.length; i++)
       infKurs[i] = new Kurs(kurskode[i]);

  //lag studenter på informatikk bachelor
    Student [] stud = new Student[3];
    stud[0] = new Student("Ola N", infKurs);
    stud[1] = new Student("Åsne S",infKurs);
    stud[3] = new Student();

   for (int i = 0 ; i< stud.length; i++)
       stud[i].skrivUt();

}}
```

```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

  Student(){mineKurs = new Kurs[0];}

   Student(String navn, Kurs [] k){
     this.navn = navn;
     for (int i = 0; i<k.length; i++ )
          mineKurs[i] = k[i];
          mineKurs[i].antStudenter++;
   }   }

   void skrivUt() {
     System.out.println("Student med navn:"+ navn+ ",og kurs:");
         for (int i = 0;i < mineKurs.length; i ++)
                System.out.println(mineKurs[i].kurskode);
}}

class Kurs {
  String kurskode ;
  int antStudenter = 0;

  Kurs(String k) {
       kurskode = k;
 }}
```

```
>java StudentRegister2
Student med navn:Ola N, og kurs:
INF1000
INF1040
MAT1030
Student med navn:Åsne S, og kurs:
INF1000
INF1040
MAT1030
Student med navn:null, og kurs:
```

Sammenligning: Objektdiagram og Klassediagram





Generelt om Oblig 3

- Les oppgaveteksten nøye! Ikke gjør mer enn det er spørsmål etter!
- Identifiser objektene og bestem datastruktur for hver klasse som du trenger
- Tegn gjerne en figur over objektene og datastrukturen. Da er det lettere å programmere
- Skaff deg oversikt over spesielle teknikker du trenger å bruke, i dette tilfelle lesing fra og skriving til fil
- Skriv koden gradvis og test ut metoder etter hvert som du skriver dem
- Skriv gjerne først de metodene som er nødvendig for å få en enkel prototype av programmet ditt opp og kjøre! Utsett for eksempel filbehandlingen til slutt!

4 klasser er angitt i oppgaveteksten

- class Oblig3
  - her legger vi main-metoden som sparker det hele i gang.
  - fra denne klassen oppretter vi et Hybelhus-objekt og kaller en metode i dette objektet som styrer interaksjonen med brukeren
- class Hybelhus
  - her ligger den sentrale datastrukturen og metoder for alle funksjonene i menyen
  - vi initialiserer datastrukturen for hybelhuset i konstruktøren
- class Hybel
  - info knyttet til en hybel: leietager og utestående
- class Student
  - info knyttet til student: navn og saldo

class Oblig3 – forslag til kode

```
public class Oblig3 {

   public static void main(String[] args) {
      Oblig3 oblig3kjøring = new Oblig3(args);
   }

   Oblig3(String[] inputfiler){
      String datafil = "HaiHus.data";
      String strømfil = "Lysregning.data";
      if( inputfiler.length > 0 )
         datafil = inputfiler[0];
      if( inputfiler.length > 1 )
         strømfil = inputfiler[1];
      Hybelhus utsyn = new Hybelhus(datafil,strømfil);
      utsyn.kommandoløkke();
   }
}
```

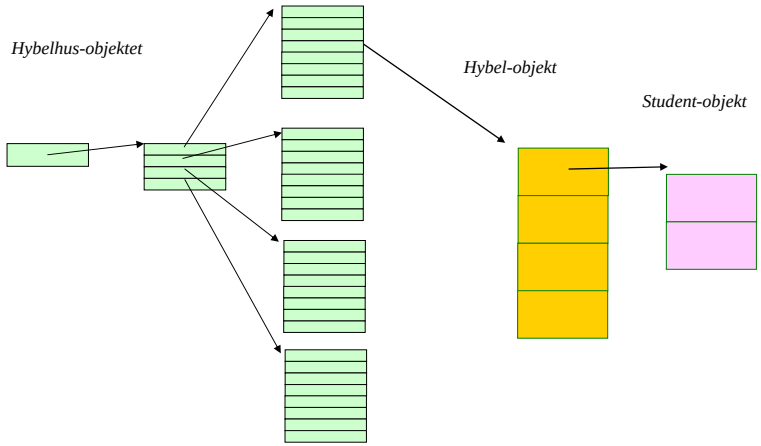
Det eneste main-metoden trenger å gjøre, er å opprette et objekt av klassen Oblig3

Default filnavn som i oppgaveteksten, men med mulighet for endringer fra kommandolinjen. Bra for debugging – du kan lett bruke andre testfiler!

Filnavnene overføres til konstruktøren for Hybelhus-klassen

Så overfører vi kontrollen til metoden i utsyn som styrer brukerinteraksjon

Enkel skisse av datastrukturen



class Hybelhus: forslag til datastruktur

```
public class Hybelhus {
    private String datafil;
    private String strømfil;

    final int ANT_KOR = 4;
    final int ANT_ROM = 8;
    final String TOM_HYBEL = "TOM HYBEL";

    private int totalFortjeneste;
    private int antallHybelmånederMedTommeHybler;
    private int totaltAntallMåneder;

    In tastatur = new In();
    Out skjerm = new Out();

    private Hybel[][] hyblene = new Hybel[ANT_KOR][ANT_ROM];
```

'private' skjuler variable for andre klasser. Dette er god programmeringsskikk!

'final' markerer at verdiene ikke kan endres. Det er vanlig at slike konstanter har store bokstaver.

Variable for informasjonen som leses fra datafilen.

Konstruktøren til Hybelhus-klassen

- Konstruktøren utføres når et objekt opprettes og aldri siden.
- I konstruktøren er det vanlig å gi startverdier til datastrukturen. Konstruktøren mottar gjerne argumenter med informasjon som den trenger for å gi startverdier.
- Siden poenget med denne oppgaven er å trene på å lage en objekt-orientert modell, skal jeg vise i detalj hvordan man kan foreta innlesning fra datafilen i konstruktøren
- Tips om lesing fra strømfilen står i oppgaveteksten. Skriving til filer må dere selv finne ut av (se kap. 3 i boka!).
- Vær nøye med å teste ut koden steg for steg! Når dere kommer til skriving av fil, vær nøyaktig med å teste at filen ser akkurat slik ut som den skal etter skriving!
- Tips ved innlesning: Skriv ut til skjerm samtidig som dere leser inn. Lurt for feilsøking!

class Hybelhus: forslag til konstruktør

```
Hybelhus( String datafil, String strømfil ) {
        this.datafil = datafil;
        this.strømfil = strømfil;
        skjerm.outln("Leser fra " + datafil + ":");
        In hybelfil = new In(datafil);
        for(int i = 0; i < ANT_KOR * (ANT_ROM - 1); i++) {
                int gang = hybelfil.inInt(";");  skjerm.out(gang);
                char bokstav = hybelfil.inChar(";");  skjerm.out(bokstav);
                String studentnavn = hybelfil.inWord(";"); skjerm.out(" " + studentnavn);
                int saldo = hybelfil.inInt(";"); skjerm.outln(" " + saldo);
                hybelfil.readLine(); //for å bli kvitt resten av linja
                hyblene[gang - 1][(int)(bokstav - 'A')] = new Hybel(studentnavn.trim(), saldo);
        }
        totaltAntallMåneder = hybelfil.inInt(";");
        antallHybelmånederMedTommeHybler = hybelfil.inInt(";");
        totalFortjeneste = hybelfil.inInt(";");
        hybelfil.close();
}
```

Nyttig metode i class Hybelhus

```
void listLeietagere(){
    for(int i = 0; i < ANT_KOR; i++){
     for(int j = 1; j < ANT_ROM; j++){  //Tar ikke med fellesarealene her
        skjerm.out(i+1);
        skjerm.out((char)(j+'A') + " ");
        Student student = hyblene[i][j].getLeietaker();
        if(student != null)
            skjerm.outln(student.getNavn() + " " + student.getSaldo());
        else
            skjerm.outln(TOM_HYBEL  + " " + hyblene[i][j].getUtestående());
     }
    }
}
```

Her har jeg brukt en vanlig Java-konvensjon når metodene for å hente dataverdi begynner med get. Metodene for å oppgi verdier skal begynne med set ifølge konvensjonen. NB! Bruk alltid slike metoder for å aksessere variable i en annen klasse og la variablene være deklarert private. Alt annet er dårlig programmeringsskikk og gir i lengden opphav til stygge bugs!

Videre jobbing med class Hybelhus:

- Begynn gjerne med å lage metoden for å styre menyen – jeg har kalt denne ”kommandoløkke()”. Identifiser alle metodene du skal kalle herfra.
- Tenk igjennom hva hver metode skal gjøre og hvordan den skal gjøre det FØR du begynner å skrive kode. Bruk gjerne små skisser og stikkord.
- Spesielt må du tenke over hvilke metoder du trenger fra de andre klassene og hvordan dataverdier skal endres.
- Utfordringen er å bryte oppgaven du skal løse ned i mange småproblemer – som hver for seg er enkle å løse – og så sette sammen løsningene av småproblemene slik at de sammen kan løse hele oppgaven.

... og de andre klassene

- For klassene Hybel og Student gjelder det samme: list opp alle metodene du trenger og tenk over hvilke data disse trenger. Hvordan skal de få tak i informasjonen de trenger?
- Husk at du aldri bør kode overalt samtidig. Når du har fått en viss oversikt, skriv kode for de enkle metodene først og test dem grundig før du går videre.
- Ikke lås deg til en bestemt løsning og datastruktur fra starten. Ofte mangler vi oversikt når vi begynner, og blir nødt til å endre konstruksjonen av programmet underveis fordi det dukker opp ting vi ikke hadde tenkt på da vi startet. Dette er heller regelen enn unntaket!

... og til slutt vil jeg bare ha sagt at:

- Å jobbe med obligene er det du lærer mest av i kurset.
- Bruk boka aktivt som oppslagsverk i denne prosessen. Studer relevante programeksempler i boka! Det er utrolig mye lettere å lære seg noe når du trenger det for å løse en oppgave – enn å lære seg det bare for å lese til eksamen!
- Prøv å bli litt kjent med Java API samtidig. Det er gøy! Java-verdenen er stor og interessant. Prøv deg frem på egen hånd!