Inf1000 (Uke 7)

Objekter, klasser og pekere

Are Magnus Bruaset og Arild Waaler
Institutt for informatikk
Universitetet i Oslo

Objekter

- Hvorfor deler vi verden inn i enheter når vi snakker om den ?
  - En blomst, fjorten trær, ti mennesker, en bil, en vei, mange murstein, en bankkonti,....
- *Svar* : For bedre å kunne tenke om, snakke om og forstå verden.

Verden består av mange objekter, noen ganske like, noen ulike

- Ser vi rundt oss i auditoriet, ser vi
  - Studenter (mange), stoler, murstein, lysarmatur, blyanter,...
- Vi ser at når vi betrakter verden, deler vi den opp i et passe antall enheter/deler/'klumper' som vi har egne navn for.
- Slike enheter/'klumper' vi deler verden inn i, kaller vi ***objekter***
- Mange av objektene i verden er egentlig like, av samme type, men kan skille seg fra hverandre med feks. ulike navn
  - Studentene Kia og Espen
  - to bøker med ulik tittel/forfatter
- Objekter som er egentlig er like sier vi tilhører samme ***klasse***
  - De beskrives av samme sett med variable, men har *ulike* verdier på noen disse (to biler av samme bilmerke men med ulike reg.nummer)

Klasser og objekter i verden

- To objekter kan også være helt vesensforskjellige (f.eks et tre og en lastebil)
  - De er da to objekter av hver sin klasse (klassen Tre og klassen Lastbil)
- Hva vi velger å betrakte som et objekt, og hvilke klasser vi bruker for å beskrive en problemstilling, er ikke bestemt på forhånd. Innenfor vide rammer *bestemmer vi det selv*.

En nominalistisk filosof vil si:

- Klasser er generelle begreper som egentlig ikke eksisterer i verden – det som eksisterer er objektene.
- Generelle begreper (klasser) er menneskenes måte å beskrive og strukturere verden, ikke gitt en gang for alle og vi kan godt lage oss nye begreper.

Hvor mange klasser (og objekter) er det ?

Hvilke klasser vi bruker til å beskrive et problem, varierer ofte etter hvor detaljert vi betrakter en problemstilling og hvilke spørsmål vi ønsker å kunne gi svar på:

- Beskriver vi problemet med *veitrafikk og køer på veiene*, er vi neppe interessert i mer enn å telle antall biler og kanskje skille mellom lastebiler, busser og personbiler.
- Beskriver vi problemet til en *bilfabrikk*, trenger vi en meget detaljert og komplisert beskrivelse av hver bil (bestående av motor, hjul, karosseri, lys,..., hver beskrevet med sin klasse) og mange ulike typer av biler. Vi se da en rekke klasser som hver beskriver sin sine objekter.

Objektorientert Programmering - I

Når vi betrakter et problem vi skal lage et datasystem for, gjør vi to avgrensninger:

1. Vi ser bare på ***en del av verden*** (vårt problemområde)
2. Innenfor problemområdet betrakter og beskriver vi bare det som er der med ***en viss detaljeringsgrad*** - bare så mange detaljer vi trenger for å svare på de spørsmål datasystemet skal kunne gi svar på.

Eks: Hvordan beskrive en student ? Skal vi lage:

a) Et Studentregister, registrerer vi bare navn, personnummer, adresse, tidligere utdannelse og kurs (avlagte og kurs vedkommende tar nå)

b) Et legesystem for studenter, ville vi ta med svært mange opplysninger om hver student (medisiner, sykdommer, resultat fra blodprøver, vekt...) som vi ikke ville drømme om å ha i et vanlig studentregister

Objektorientert Programmering - II

Når vi skal lage et programsystem, så skal det i størst mulig grad være en modell av vårt problemområde – en en-til-en kopi:

Ett objekt i problemområdet medfører at det skal være ett objekt i programmet som representerer dette 'verdens-objektet' når vi kjører programmet.

(i tillegg kommer mange klasser og objekter i programmet for å lese fra tastatur og fil, skrive til skjerm,..)

Eks.: Hver virkelig student skal ha sitt Student-objekt i et studentregister-system.

Hvordan lage klasser og objekter i et program.

- Klasser deklarerer vi med `class`
- Vi lager pekere til objekter ved å deklarere dem med klassenavnet
- Vi lager et objekt med å si `new` foran et klassenavn
- Forholdet mellom et objekt og en peker er som en array-peker og et array-objekt

```
class Student {
   String navn, adresse;
}

class StudentRegister {
  public static void
     main(String args []) {

    Student s1, s2;

    s1 = new Student();
    s2 = new Student();

   }
}
```

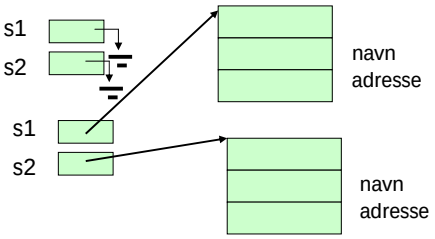
Hva er et objekt i programmet?

- Et objekt er et område i lageret som inneholder *en kopi* av alle de metodene og variable i en klasse det *ikke* står `static` foran
- Klassene er en slags mal/form/oppskrift som vi kan lage objekter med.
- Lager vi to, tre,.. objekter av klassen, får vi to, tre,.. slike kopier
- De variable og metode det ikke står `static` foran, kalles *objekt-variable og objekt-metoder*
- De variable og metoder det står `static` foran, kalles *klasse-metoder og klasse-variable*, og blir ikke med i objektene (men ligger lagret i bare ett eksemplar et annet sted)

Peker og objekter

```
class Student {
   String navn, adresse;
}

class StudentRegister {
  public static void
     main(String args []) {

    Student s1,
            s2;

    s1 = new Student();
    s2 = new Student();

   }
}
```

- En peker inneholder adressen til hvor et objekt ligger i lageret
- eller den inneholder 'null' (= ikke-objekt)
- Vi tegner adressen som en 'pil'



Programmer i Java består av en eller flere klasser

- Vi deler opp programmet vårt i flere klasser
  - Fordi hver programdel (klasse) skal være mulig å holde oversikt over – ikke for stor
  - Fordi en klasse skal være god modell av en del av problemet vårt vi lager program for.
    - Anta at vi hadde et datasystem som omhandlet kurs og studenter. Da ville vi ha en klasse Student og en klasse Kurs i programmet.
- En klasse inneholder
  - Deklarasjon av null eller flere variable
    - som beskriver *ett eksemplar* av det klassen er modell av
  - Null eller flere metoder
- En klasse representer et generelt begrep som:
  Student, Kurs, Person, Dokument, Eiendom, CDRegister, CD, Billett, Bil, Fly, Tre, Ku, Motorsykkel...

Objekter og pekere, og hvordan få adgang til innmaten av et objekt (.)

- Når vi har laget et objekt med `new`, har vi altså fått en kopi av objekt-variablene og –metodene, men hvordan får tak i dem ?
- Vi bruker operatoren . (punktum).
  - Foran punktumet har vi navnet på en peker til et objekt.
  - Etter punktum har vi navnet på en variabel eller metode inne i objektet –
  - Punktumet leses som 'sin' eller 'sitt'
    - eks: La s1 peke på et Student-objekt.

```
s1.navn = "Ola N";
```

```
class Student {
   String navn, adresse;

   void skrivUt() {
     System.out.println("Student med navn:"
          + navn+ ", adr:" + adresse);
   }
}

class StudentRegister {
  public static void main(String args []) {

    Student s1, s2;

    s1 = new Student();
    s1.navn = "Ola N";
    s1.adresse = "Storgt. 12, 1415 Nordby";
    s2 = new Student();
    s2.navn = "Åsne S";
    s2.adresse ="bokhandelen i Kabul";
    s1.skrivUt();
    s2.skrivUt();
   }
}
```

```
>java StudentRegister
Student med navn:Ola N, adr:Storgt. 12, 1415 Nordby
Student med navn:Åsne S, adr:bokhandelen i Kabul
```

Oppsummering om klasser, objekter, pekere og .

- Verden består av **objekter** av ulike typer (**klasser**). Ofte er det mange objekter av en bestemt type.
- Objekter som er av samme klasse, beskrives med de samme variablene, men vil ha forskjellige verdier på noen av disse.
  - Eks: To bankkonti med ulik eier og kontonummer, men samme beløp på saldo (tilfeldigvis)
- Vi lager OO-programmer ved å lage en modell av problemområdet i Javaprogrammet
  - ett objekt i verden gir ett tilsvarende Java-objekt i programmet
  - Objekter kan være av ulik type, og for hver slik type deklarerer vi en klasse i programmet

.. oppsummering forts.

- Et Javaprogram består av en eller flere klasser
- En klasse er en deklarasjon av data og metoder for ***ett objekt*** av klassen.
- Vi deklarerer pekere til objekter av en bestemt klasse – f.eks. class Kurs {..} slik:

```
Kurs kurs14, k2, k;
```

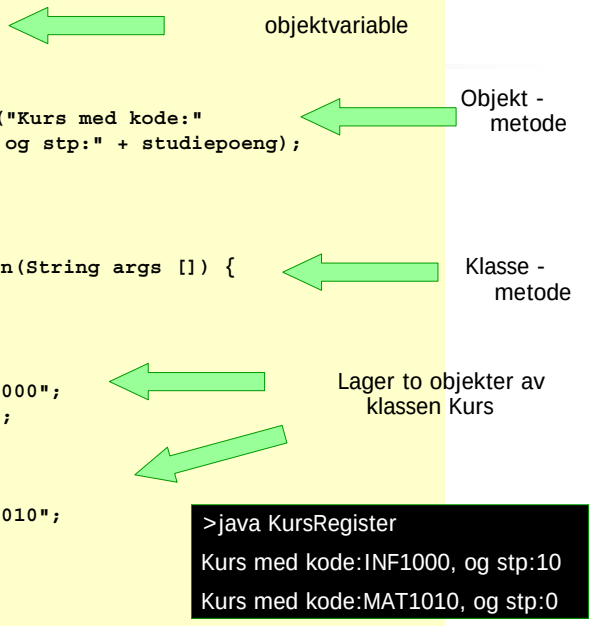
- Vi lager objekter fra klassen med **new**

```
k2 = new Kurs();
```

- Et objekt inneholder en kopi av alle ikke-statiske variable og ikke-statiske metoder i klasse
  - Disse kalles objekt-variable og objekt-metoder
- Vi får adgang (lese, skrive og kalle metoder) til det som er inni et objekt ved . Operatoren :
  - **Vi må ha en peker til et objekt etterfulgt av punktum .**

```
s2.adresse ="bokhandelen i Kabul";
s1.skrivUt();
```

```
class Kurs {
   String kurskode;
   int studiepoeng;

   void skrivUt() {
     System.out.println("Kurs med kode:"
          + kurskode+ ", og stp:" + studiepoeng);
   }
}

class KursRegister {
  public static void main(String args []) {

    Kurs inf, mat;

    inf = new Kurs();
    inf.kurskode = "INF1000";
    inf.studiepoeng = 10;
    inf.skrivUt();

    mat = new Kurs();
    mat.kurskode = "MAT1010";
    mat.skrivUt();

   }
}
```




```
>java KursRegister
Kurs med kode:INF1000, og stp:10
Kurs med kode:MAT1010, og stp:0
```

Klasse-variabel (=statisk variabel)

- Setter vi static foran en variabel, er det er bare **én** felles variabel med det navnet for alle objektene.
- Setter vi **`static`** foran en metode, har den bare utsikt til :
  - sine egne lokale variable og parametere
  - andre statiske variable og metoder
  - klassenavnene
- Statiske metoder og variable kan man få adgang til både
  - via klassenavnet og punktum
  - via peker til et objekt av klassen og punktum

```
class B {
       static int i = 0;
       double x = 0.0;
}
class A
{      int k;

    public static void  main ( String[] args)  {
           B b1 = new B(), b2 = new B();
           // endre klassevariable (det er bare en felles)
           System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
           b1.i = 4;
           System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
           // endre objektvariabel (en kopi i hvert objekt)
           System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
           b1.x = 2;
           System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
      }
}
```

```
>java A
b1.i :0, b2.i:0
b1.i :4, b2.i:4
b1.x :0.0, b2.x:0.0
b1.x :2.0, b2.x:0.0
```

```
class A2
{        int k;  // objektvariabel 'k'
    public static void  main ( String[] args)  {
              k = 1;
 }}
```

```
>javac a2.java
a2.java:6: non-static variable k cannot be referenced from a static context
         k = 1;
         ^
1 error
```

```
class A2
{      int k;

    public static void  main ( String[] args)  {
              A2 aa = new A2();
              aa.k = 1;
      }
}
```

```
>javac A2.java

>
```

Arrayer av pekere til objekter

- Vi kan lage arrayer av pekere til objekter (men ikke av objektene direkte) omlag på samme måte som vi lager arrayer av int, double og String
- Har vi deklarert klassen **`Kurs {...}`** kan vi lage array som følger:

```
Kurs [] ifiKurs;
```

Her deklareres 'bare array-pekeren

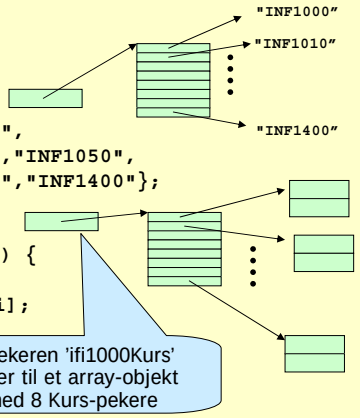
```
ifiKurs = new Kurs[120];
```

Her lages array-objektet med 120 'tomme' pekere

```
ifiKurs[0] = new Kurs();
```

Her settes det første pekeren i array-objektet til å peke på et nytt Kurs-objekt

```
class Kurs {
   String kurskode;
   int studiepoeng=10;

   void skrivUt() {
     System.out.println("Kurs med kode:"
         + kurskode + ", og stp:"
          + studiepoeng);
   }
}

class KursRegister2 {
  public static void main(String args []) {

    String [] kursKoder= {"INF1000","INF1010",
                          "INF1020","INF1040","INF1050",
                           "INF1060","INF1070","INF1400"};
    Kurs [] ifi1000Kurs = new Kurs[8];

    for(int i = 0; i < kursKoder.length; i++) {
       ifi1000Kurs[i] = new Kurs();
       ifi1000Kurs[i].kurskode = kursKoder[i];
       ifi1000Kurs[i].skrivUt();
    }
  }
}
```



Eksekvering av KursRegister2

```
>java KursRegister2
Kurs med kode:INF1000, og stp:10
Kurs med kode:INF1010, og stp:10
Kurs med kode:INF1020, og stp:10
Kurs med kode:INF1040, og stp:10
Kurs med kode:INF1050, og stp:10
Kurs med kode:INF1060, og stp:10
Kurs med kode:INF1070, og stp:10
Kurs med kode:INF1400, og stp:10
```

Et meget enkelt banksystem

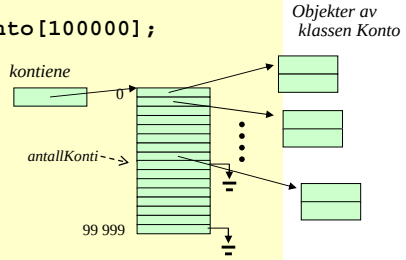
- Vi har klassene:
  - Konto
  - Bank

  Hvilke opplysninger har vi i hver klasse/objekt ?

- Og operasjonene (dvs. metodene):
  - Ny konto
  - Sett inn
  - Ta ut
  - Vis summen av innskudd i banken

Data i Konto og Bank (en bank har mange konti)

```
class Bank{
  Konto  [] kontiene = new Konto[100000];
  int    antallKonti = 0;
  String navn;

  // Metoder mangler

  public static void main
    (String [] args) {..}
}

class Konto {
  String  navn, adresse;
  int     kontoNummer;
  double  saldo =0.0;

  // Metoder mangler
}
```



Metoder i Bank og Konto (+ noen hjelpemetoder)

```
class Bank{
  Konto []  kontiene = new Konto[100000];
  static int  kontoNummer = 500000;
  int  antallKonti = 0;
  In tast = new In();
  String navn;

  double sumInnskudd() {   }

  void nyKonto() {  }

  int menyValg() {  }

  public static void main (String [] args) {
        Bank b = new Bank();
        b.navn = "BB-Bank";
        int valg =0;
        Konto  k;
        double kr ;

        do {
           valg = b.menyValg();
           switch(valg) {
            ............
        } while (valg > 0);
   } // end main

  double spørSvar (String s){}

  Konto riktigKonto() {}
} // end Bank
```

```
class Konto {
   String navn,adresse;
   int kontoNummer;
   double saldo =0.0;

   void settInn (double kr) {}
   boolean taUt(double kr) {}
} // end class Konto
```

```
int menyValg() {
      System.out.println(" \nVelg funksjon i "+ navn+":");
       System.out.println ("1 - ny konto:");
       System.out.println ("2 - innskudd:");
       System.out.println ("3 - uttak:");
       System.out.println ("4 - sum forvaltningskapital\n");
      return tast.inInt();
}

public static void main (String [] args) {
       Bank b = new Bank();
       b.navn="BB-Bank";
       int valg =0;
       Konto k;
       double kr ;

       do {
          valg = b.menyValg();
          switch(valg) {
             case 1: b.nyKonto();  break;
             case 2 :k  = b.riktigKonto();
                     kr = b.spørSvar("Gi innskudd");
                     k.settInn(kr);
                          break;
             case 3 :k  = b.riktigKonto();
                     kr = b.spørSvar("Gi uttaksbeløp");
                     if (! k.taUt(kr))
                     System.out.println("IKKE NOK PENGER");
                              break;
                    case 4:System.out.println(b.navn+
                           " Sum innskudd:" + b.sumInnskudd());
                           break;
                  }

             } while (valg > 0);
             System.out.println("** AVSLUTTER BANKEN ***");
      } // end main
```

```
double spørSvar(String s){
     System.out.print(s+":");
     return tast.inDouble();
}

Konto riktigKonto() {
     System.out.print("Gi navn til eksisterende konto:");
     String s = tast.inWord("\n");
     for ( int i = 0; i < antallKonti; i++)
       if (kontiene[i].navn.equals(s) )return kontiene[i];
     return null;
}

void nyKonto() {
     System.out.print("Gi navn til ny kontoinnehaver:");
     String navn = tast.inWord("\n");
     System.out.print("Gi adresse:");
     String adr = tast.inWord("\n");

     Konto k = new Konto();
     k.adresse = adr;  k.navn = navn;
     k.kontoNummer= kontoNummer++;
     kontiene[antallKonti] = k;
     antallKonti++;
}
```

```
// Mini Banksystem 16.okt. 2006 - am
import easyIO.*;

class Bank{
  Konto [] kontiene = new Konto[100000];
  static int kontoNummer = 500000;
  int antallKonti = 0;
  String navn;

  double sumInnskudd() {
       }

  int menyValg() {
       }

  public static void main (String [] args) {
        Bank b = new Bank();
        b.navn="BB-Bank";
        int valg =0;

        do {
           valg = b.menyValg();
           switch(valg) {
             ..............
         } while (valg > 0);
   } // end main

  double spørSvar(String s){
       }

  Konto riktigKonto() {
       }

  void nyKonto() {
       }

} // end Bank
```

```
class Konto {
  String navn,adresse;
  int kontoNummer;
  double saldo =0.0;
  void settInn (double kr) {
        saldo += kr;
  }

  boolean taUt(double kr) {
       if (saldo >= kr) {
              saldo = saldo - kr;
              return true;
       } else return false;
  }
} // end class Konto
```

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.

Null, && og søppeltømmeren

- Av og til har vi behov for å teste om en peker virkelig peker på et objekt eller ikke:

```
Student s = hyblene[i][j].leietager ;
if (s != null && s.navn.equals("Ola")) {
  // her kommer vi bare hvis s peker på et studentobjekt
  // og navnet i det studentobjektet er lik "Ola"
   ..........
}
```

- Hvis vi vil fjerne et objekt fra en peker og fra hele systemet:

```
hyblene[i][j].leietager = null;
```

- Et objekt som ingen pekere peker på, blir tatt av søppeltømmeren – et program som automatisk startes hvis det er lite plass i hukommelsen:

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)

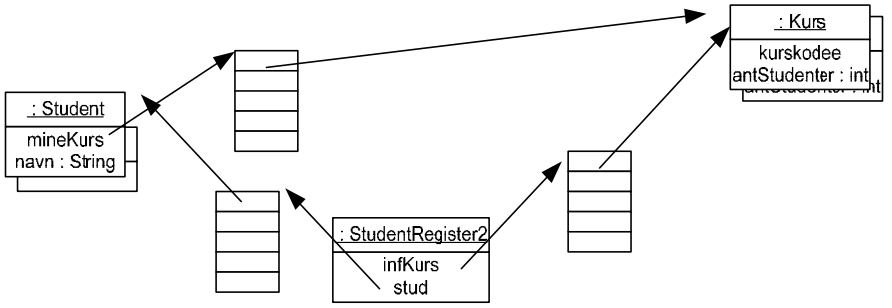
Objekt-diagrammer

- 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

Et helt enkelt studentregister med kurs, studenter og registeret

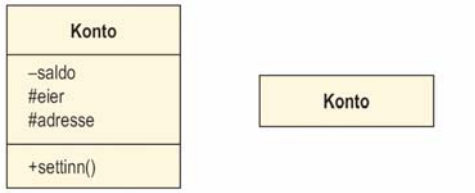
- Vi har Studenter på Ifi som første semester tar tre kurs, samtidig som vi har behov for å registrere kurs og *hvor mange* studenter som tar hvert kurs (men ikke *hvilke* studenter).
- Vi tegner først en tenkt datastruktur – et UML objektdiagram
- så skriver vi programmet

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

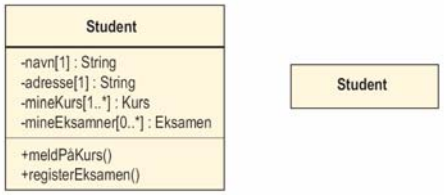


Klassediagrammer

- 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

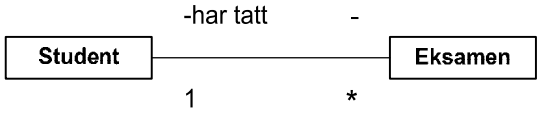
~ package

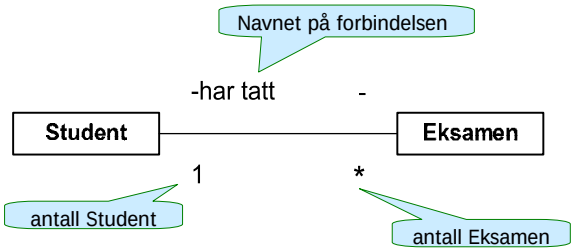
UML Klassediagrammet kan nyttes til

- 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



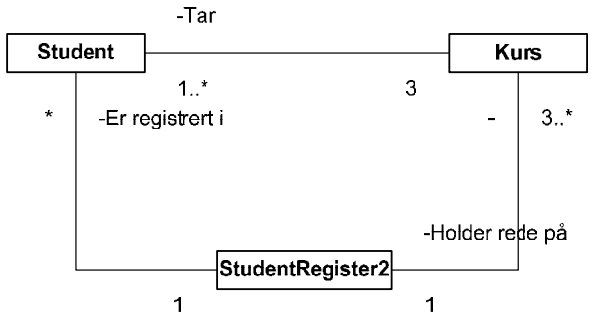


- Forbindelsen leses fra venstre:
  En student har tatt null, en eller flere Eksamener"
- Antallet objekter angis slik:

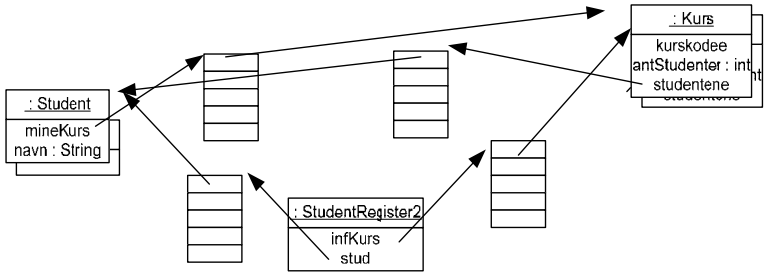
Skrivemåte	Betydning
1	en
*	null, en eller flere
1..*	minst en
3..*	minst tre
3,4,5	tre, fire eller fem

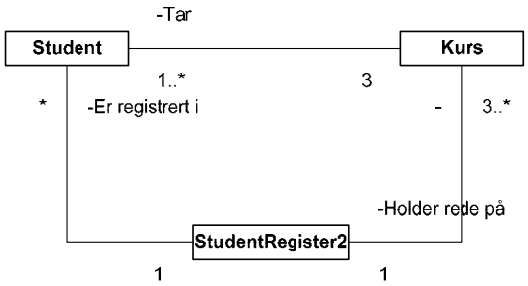
Studentregister2 – med tillegg: klassen Kurs vet *hvilke* Studenter som tar kurset

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



Sammenligning: Objektdiagram og Klassediagram





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.

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.

Oppsummering

- Klasser er oppskrifter for hvordan vi lager objekter med **`new`**
- Vi deklarerer pekere til objekter og bruker punktum .
- Kan ha arrayer av pekere til objekter
- Klasse- og objektvariable og –metoder.
- Konstruktører er 'startmetoder' med samme navn som klassen, Kalles hver gang vi sier **`new`**.
- UML-diagrammer (Objekt- og Klasse-diagram)
  - gir oversikt og forenkling
  - som skikkelige ingeniører lager vi tegninger før vi lager systemet (programmerer)