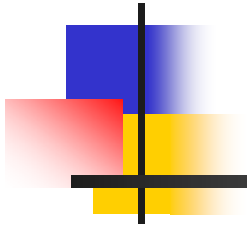
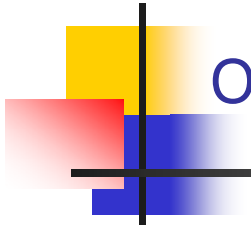


Uke 7 – Mer om Objekter, klasser og pekere; UML

27. Sept og 4. okt. 2011,

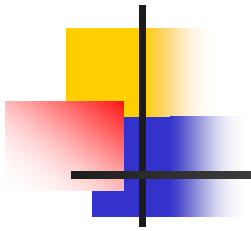
Arne Maus

Inst. for informatikk, UiO



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 kan f.eks ha 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:

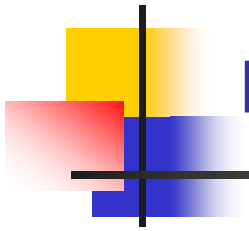
```
Kurs kurs14, k2, k;
```

- 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();
```



Et meget enkelt banksystem

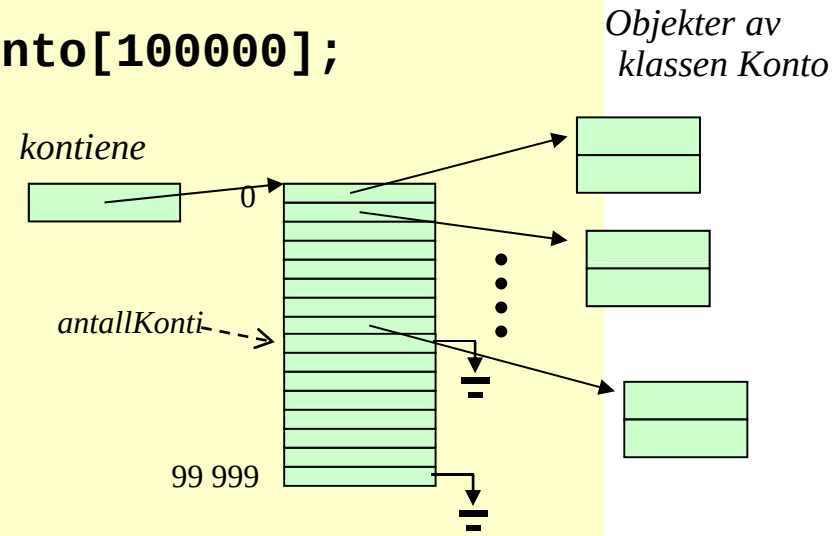
- Vi har klassene:
  - Konto
  - Bank
  - BankSystem (med main)

  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 BankSystem{
    public static void main (String [] args) {.
       Bank b = new Bank();
       b.navn = "BB-Bank";
       b.løkke();
   }
}

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

  // Metoder mangler

  }


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

  // Metoder mangler

}
```



Betyr at pekeren peker på ingenting: **`null`**

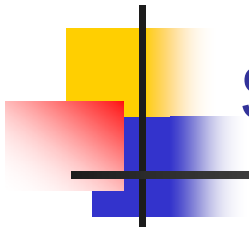
Metoder i Bank og Konto (+ noen hjelpemetoder)

```
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() {  }

  void løkke () {
         int valg =0;
         Konto  k;
         double kr ;

         do {
            valg = 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();
}

void løkke () {
       int valg =0;
       Konto k;
       double kr ;

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

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

```
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();
     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();
     System.out.print("Gi adresse:");
     String adr = tast.inWord();

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



Stringer er ordentlige objekter

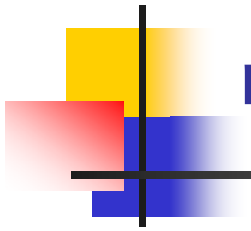
- String er en klasse i Java-biblioteket, men har en egen spesiell syntaks (skrivemåte) så det ser ut som den er en av de basale typene (som int, double,..).
- Når vi har en string, har vi alså både en peker (den vi deklarerer navnet på) og et stringobjekt.
- String-objekter kan ikke endres (trenger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

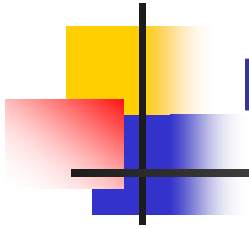
- Av og til har vi behov for å teste om en peker virkelig peker på et objekt eller ikke:

```
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:

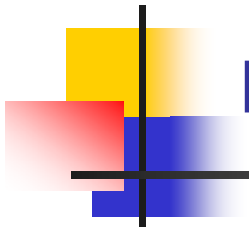
```
hyblene[i][j].leietager = null;
```

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



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

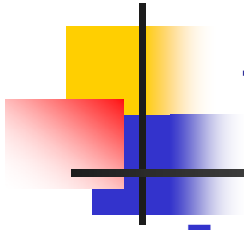


Eksempel Student med én konstruktør

```
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++;
     }
   }
```

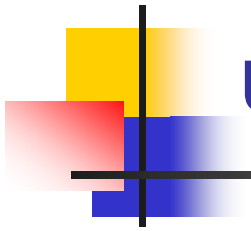


this

- 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.



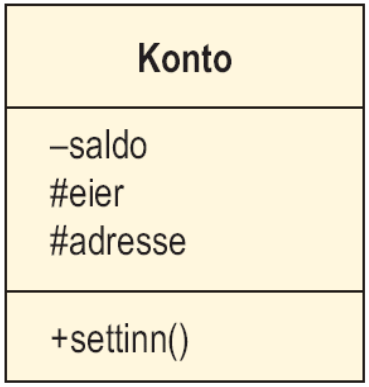
UML-diagrammer av programmene våre

- 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)

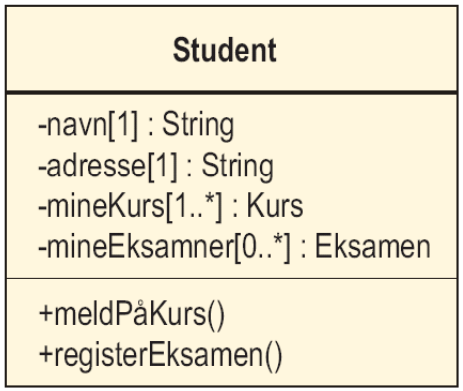
  (i UML er det mer enn 10 andre diagramtyper vi ikke skal lære)

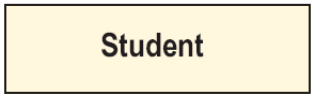
Klassediagrammer

- 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)

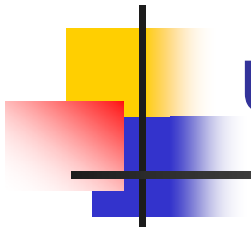
Symboler for synlighet
(fra resten av programmet)

\+ public

\- private

\# protected

~ package



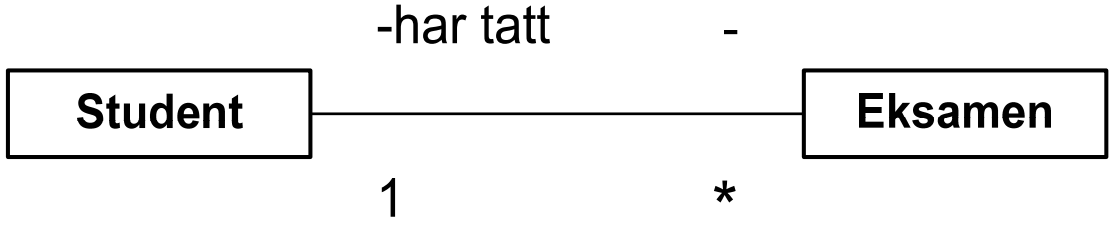
UML Klassediagrammet kan nyttes til

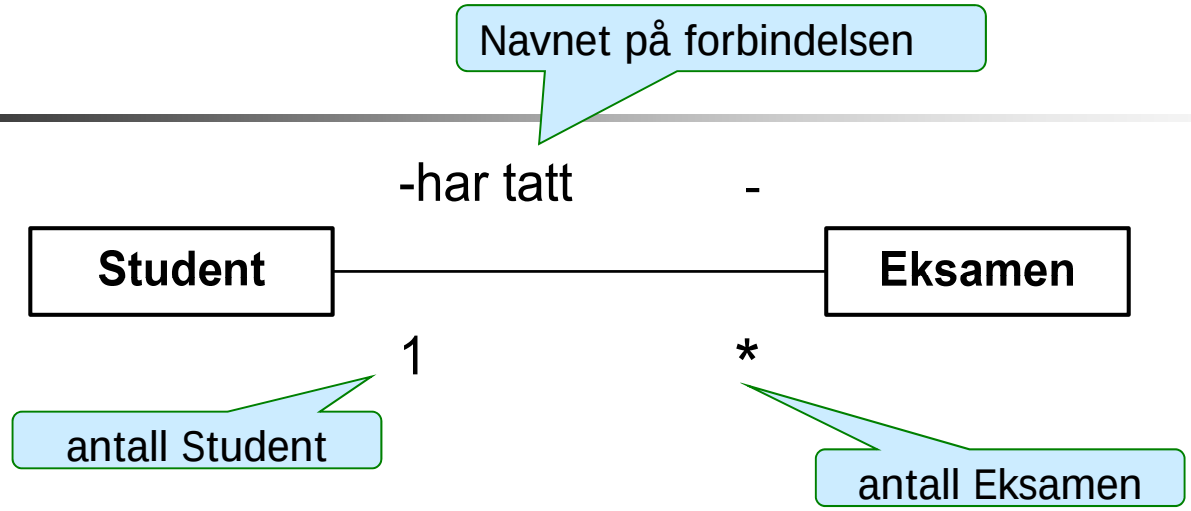
- Modell av problemområdet (domenemodell)
- Modell av klassene i programmet

  (+ 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





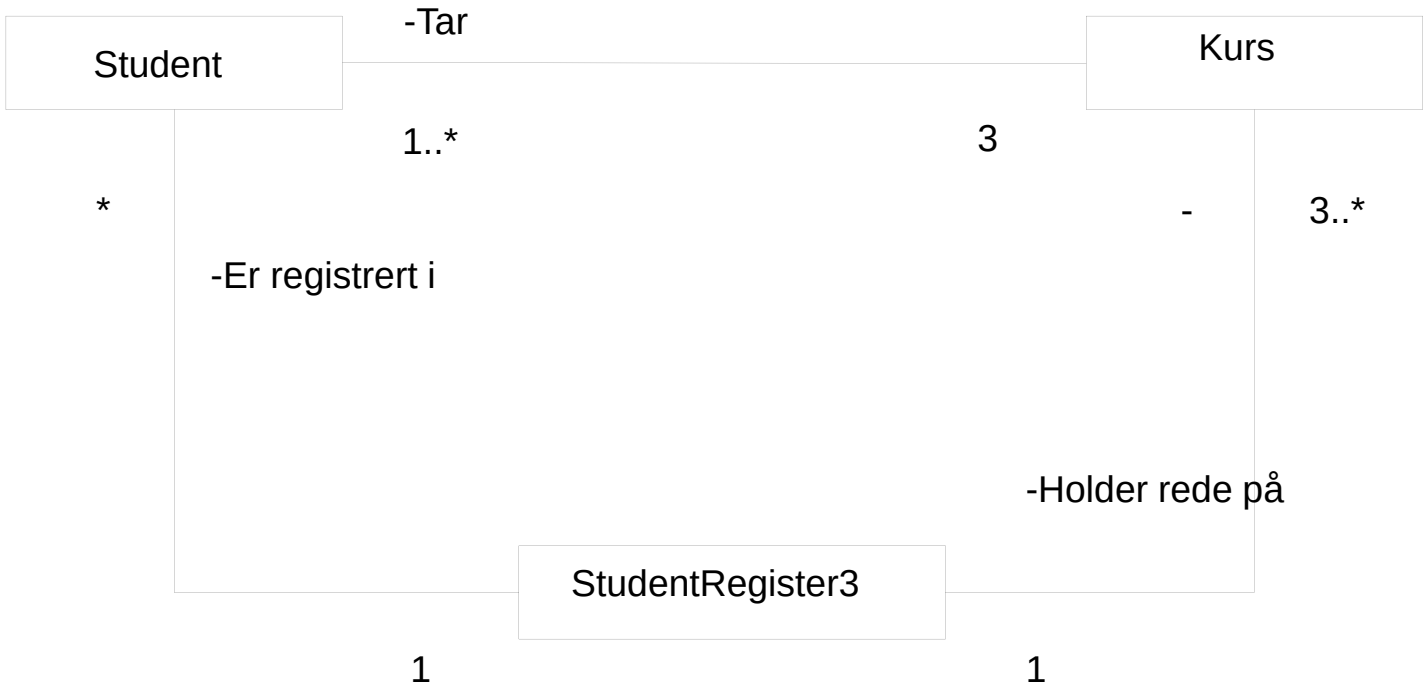
- Forbindelsen leses fra venstre:
  En student har tatt null, en eller flere Eksamener"

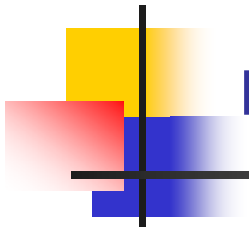
- Antallet objekter angis slik:

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

Studentregister3 – med tillegg: klassen Kurs vet *hvilke* Studenter som tar kurset

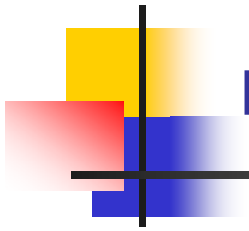
- Et studentregister holder orden på studentene og kursene, og en student tar 3 kurs hvert semester





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

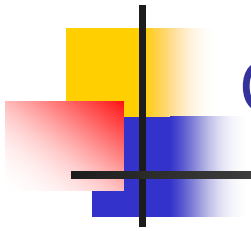


Hvilke forhold skal vi ha med i klassediagrammet ?

- 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.

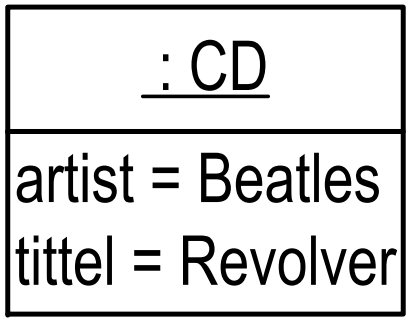


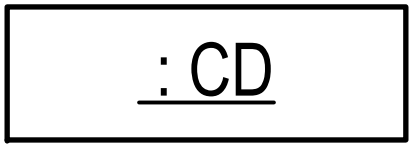
Objekt-diagrammer

- 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

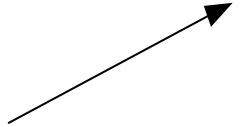
Tegning av et objekt (med mer eller mindre detaljer)

- 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



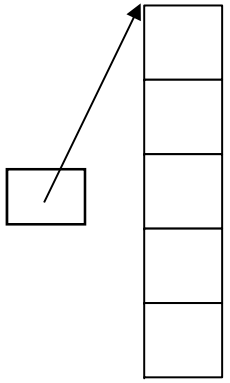


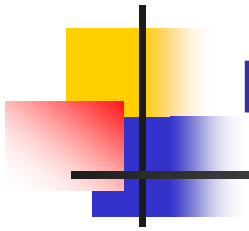
Andre elementer i et objektdiagram



- Pekere

- Peker-arrayer



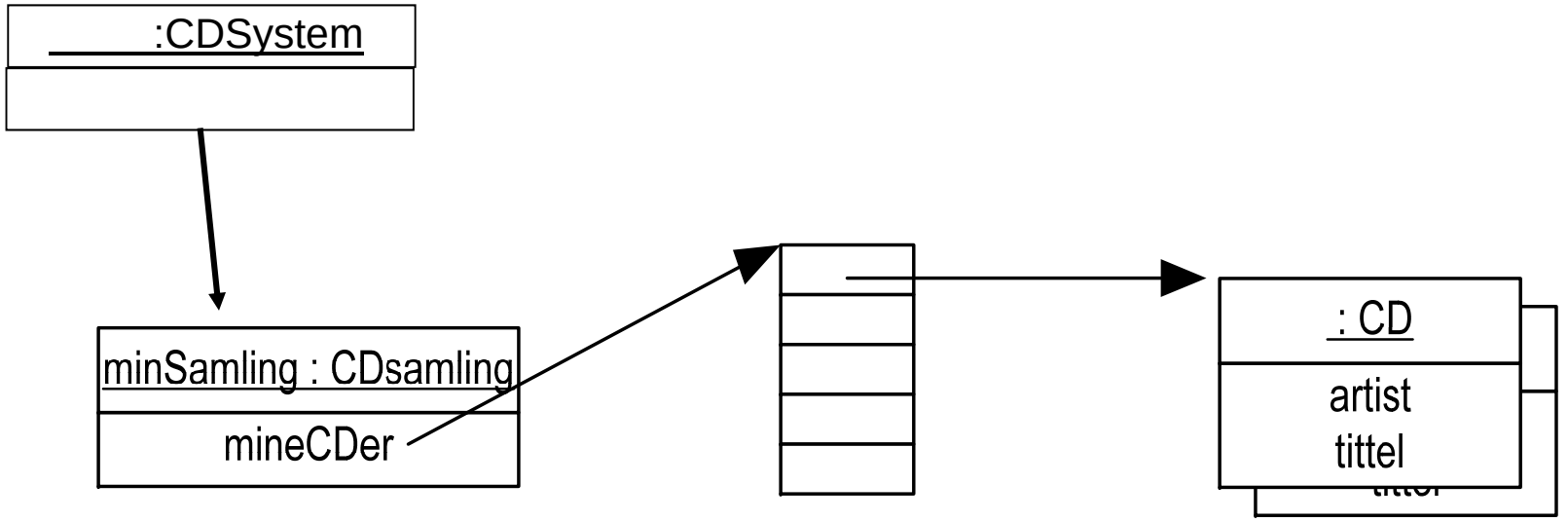


Eksempel: Et CD-samling

- 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

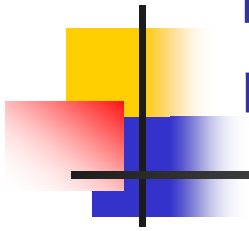
- Vi tenker oss følgende datastruktur
  (er den tilstrekkelig?)
- Vi har her forenklet programmet (klassen CDSystem inneholder main,CDSamling meny-metode og sentral switch i løkke-metoden,... , mens CD inneholder utskriftsrutine .



```
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);
}}
```

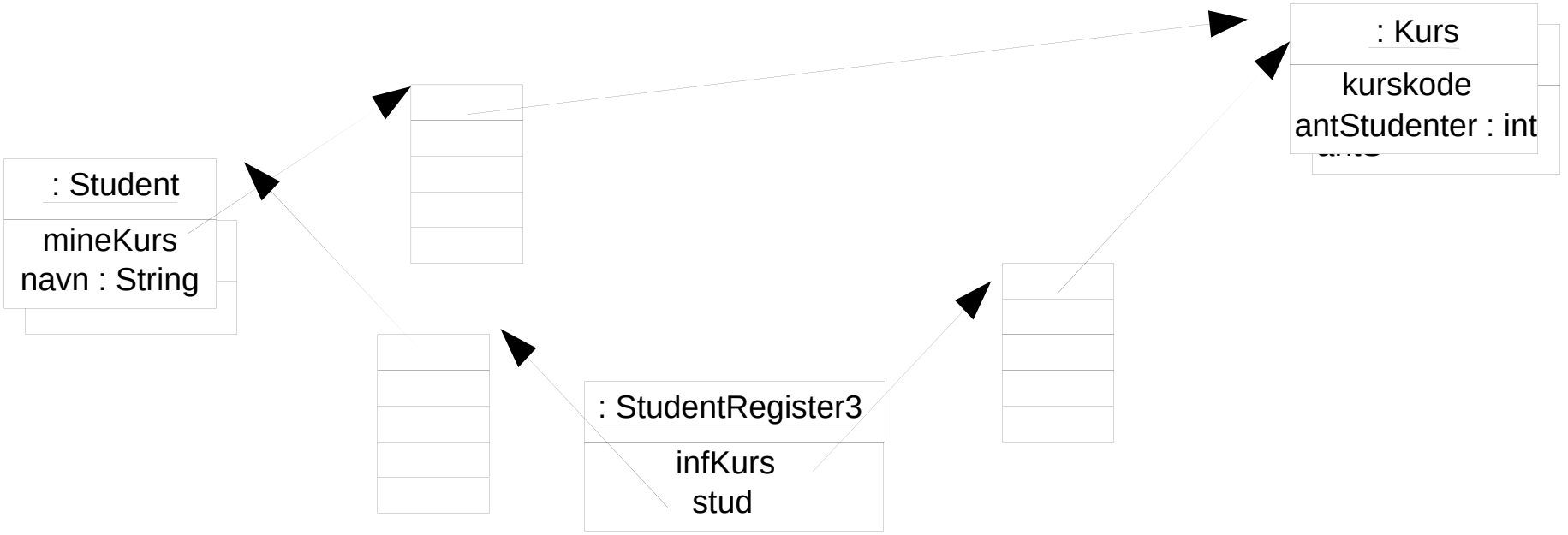
```
>java CDSystem
Velg:
 1 - les ny plate (skriv artist platetittel
 2 - skriv artist
 3 - avslutt
1
Gi artistnavn:tom
Gi tittel:Fest1
Velg:
 1 - les ny plate (skriv artist platetittel
 2 - skriv artist
 3 - avslutt
1
Gi artistnavn:ola
Gi tittel:FullFres
Velg:
 1 - les ny plate (skriv artist platetittel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

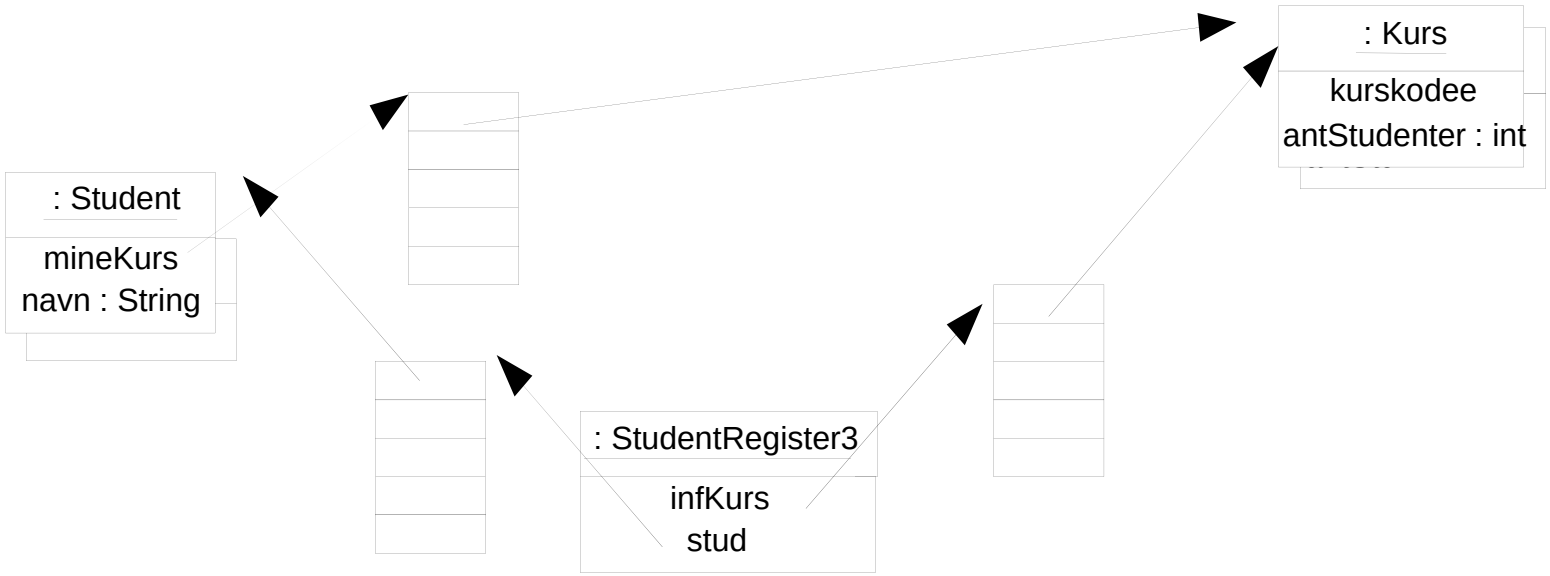


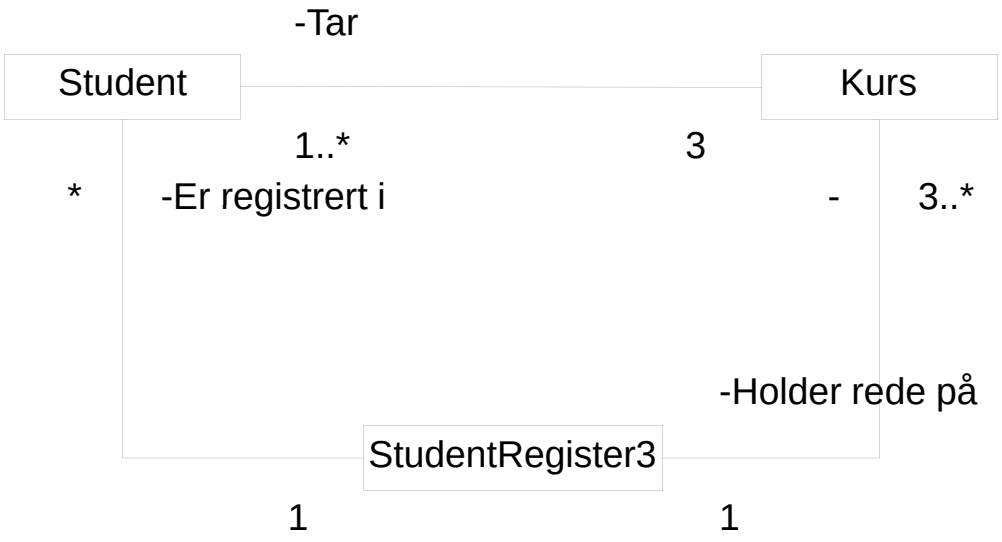
```
class StudentRegister3{
    public static void main(String args []) {
        StudentRegister3 sr = new StudentRegister3();
    }

    StudentRegister3() {
        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();

    }
}
```

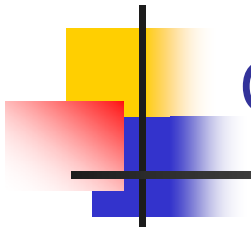
```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

  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 StudentRegister3
Student med navn:Ola N, og kurs:
INF1000
INF1040
MAT1030
Student med navn:┼sne S, og kurs:
INF1000
INF1040
MAT1030
Student med navn:null, og kurs:
```

Sammenligning: Objektdiagram og Klassediagram







Oppsummering

- Klasser er oppskrifter for hvordan vi lager objekter med **new**
- Vi deklarerer pekere til objekter og bruker operatoren: .
- 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)