UNIVERSITETET
I OSLO

# INF1000: Forelesning 7

Klasser og objekter – del 2

- Konstruktører
- Static
- UML

# REPETISJON

# Repetisjon

- 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

UNIVERSITETET I OSLO

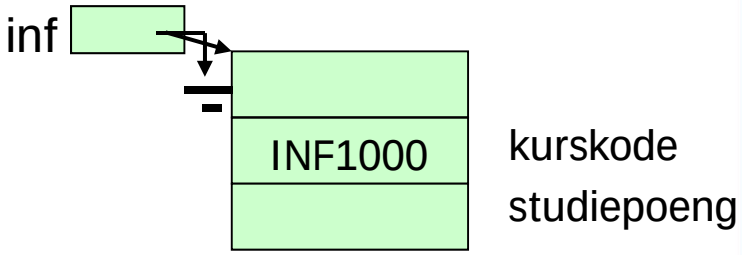
# Repetisjon forts.

- Et Javaprogram består av en eller flere klasser
- En klasse er en deklarasjon av data og metoder for ett objekt av klassen:

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

- For å ta vare på objekter, trenger vi pekere til dem:

```
Kurs inf;
```




- Vi lager objekter med new:

```
inf = new Kurs();
```

- Vi får adgang til det som er inni et objekt med . :

```
inf.kurskode = "INF1000";
```

# Komplett eksempel

UNIVERSITETET I OSLO

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

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

# KONSTRUKTØRER

# Konstruktører – startmetoder i klasser

- Ofte ønsker vi å gi (noen av) variablene i et objekt fornuftige startverdier samtidig som det opprettes.
- Dette kan gjøres ved hjelp av en konstruktør.
- Eksempel på deklarasjon:

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

    Kurs(String kode, int poeng) {
        kurskode = kode;
        studiepoeng = poeng;
    }
}
```

- Eksempel på bruk:

```
Kurs k = new Kurs("INF1000", 10);
```

# 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 konstruktør, og parametrene til denne heter det samme som objekt-variable i objektet:

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

    Kurs(String kurskode, int studiepoeng) {
        this.kurskode = kurskode;
        this.studiepoeng = studiepoeng;
    }
}
```

  - 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.

# Stringer er ordentlige objekter

- String er en klasse i Java-biblioteket, men har en egen spesiell syntaks (skrivemåte)
- Når vi har en string, har vi alså både en peker (den vi deklarerer navnet på) og et stringobjekt.
- String-objekter kan ikke endres
- Egen skrivemåte for stringkonstanter:

```
String s1 = "En fin dag i mai";
String s2 = new String("En fin dag i mai");
```

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

```
if (s1 == s2) { ... }
if (s1.equals(s2)) { ... }
```

UNIVERSITETET
I OSLO

# STATIC

# Klasse-variable (statiske variable)

- Anta at alle kontoene i banken skal ha samme rente. Hvordan lagre denne?
- En mulighet er å la rente være en statisk variabel i klassen Konto:

```
class Konto {
    String navn, adr;
    int kontoNr;
    double saldo;
    static double rente;
}
```

- Denne blir da **felles** for alle Konto-objektene, dvs at det bare finnes **en** kopi av denne variabelen som alle objektene av klassen har tilgang til.
- For å få tilgang til variabelen utenfor et Konto-objekt:

```
System.out.println(Konto.rente);
```

# Klasse-variable – eksempel

- Hver kurs skal ha en unik id, som starter på 1 for det første kurset som opprettes.
- Må i tillegg holde rede på hvor mange kurs vi har laget så langt.
- Dette er en egenskap ved **klassen**, ikke hvert enkelt objekt – vi bruker derfor en statisk variabel til dette:

```
class Kurs {
    String kurskode;
    int studiepoeng,
    int id;
    static int antallKurs = 0;

    Kurs(String kurskode, int studiepoeng) {
        this.kurskode = kurskode;
        this.studiepoeng = studiepoeng;
        id = ++antallKurs;
    }
}
```

# Klasse-metoder (statiske metoder)

- Metoder kan også deklareres som statiske, dersom
  - de logisk hører til klassen som helhet og ikke hvert enkelt objekt
  - de bare benytter seg av de statiske variablene i klassen (og eventuelt andre statiske metoder)
- Eksempel: metode for å oppdatere renten

```
class Konto {
    String navn, adr;
    int kontoNr;
    double saldo;
    static double rente;

    static void nyRente(double ny) {
        rente = ny;
    }
}
```

- Eksempel på bruk:

```
Konto.nyRente(2.75);
```

# Static – vanlig feilmelding

UNIVERSITETET I OSLO

- Hvis vi i en klasse-metode prøver å aksessere en objekt-variabel (f.eks. saldo), gir kompliatoren en feilmelding:

```
class Konto {
    String navn, adr;
    int kontoNr;
    double saldo;
    static double rente;

    static void nyRente(double ny) {
        if (saldo > 0) {
            rente = ny;
        }
    }
}
```

```
>javac Konto.java
Konto.java:8: non-static variable saldo cannot be referenced from a static context
        if (saldo > 0) {
        ^
1 error
```

UNIVERSITETET I OSLO

# main – en statisk metode

- Hva er galt med denne?

```
class Start {
    int k;

    public static void main(String[] args) {

        k = 1;

    }

}
```

- Hvordan rette feilen?

UNIVERSITETET
I OSLO

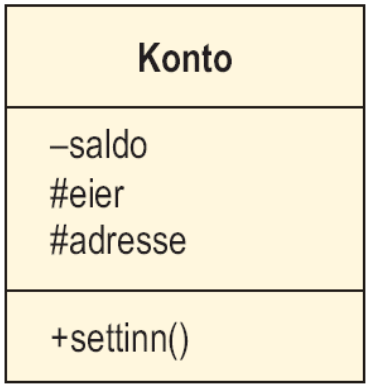
# UML

# UML-diagram

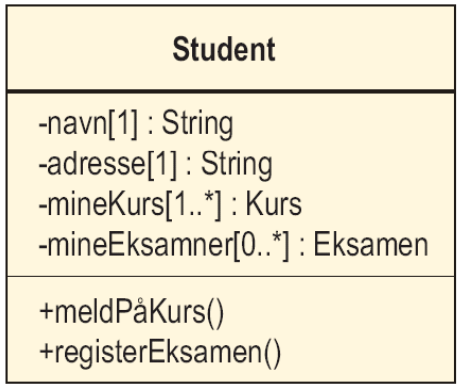
- Hvorfor tegne diagrammer over systemet?
  - Planlegging (arkitekter og ingeniører tegner først – så bygger de!)
  - Oversikt
  - Samarbeid med andre programmererer / systemutviklere
  - Dokumentasjon
- Unified Modeling Language – en industristandard fra OMG (Object Management Group)
- 13 ulike diagramtyper, vi skal se på
  - klassediagram
  - objektdiagram

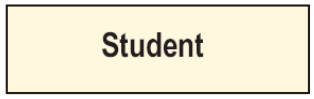
# Klassediagrammer

- En mer **kompakt** måte enn objektdiagram å tegne sammenhengen i programmet.
- Skiller seg fra objektdiagram ved at vi ikke direkte tegner datastrukturen (pekere, arrayer og objekter), men bare forhold (assossiasjoner, 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




Symboler for synlighet
(fra resten av programmet)
\+ public
\- private
\# protected
~ package

Alltid:

- Navnefeltet
  - klassenavnet

Kan utelates:

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

# Bruk av klassediagram

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

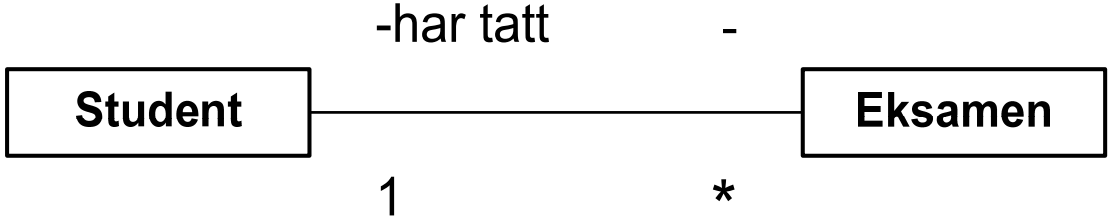
  (+ modell av databasen,....)

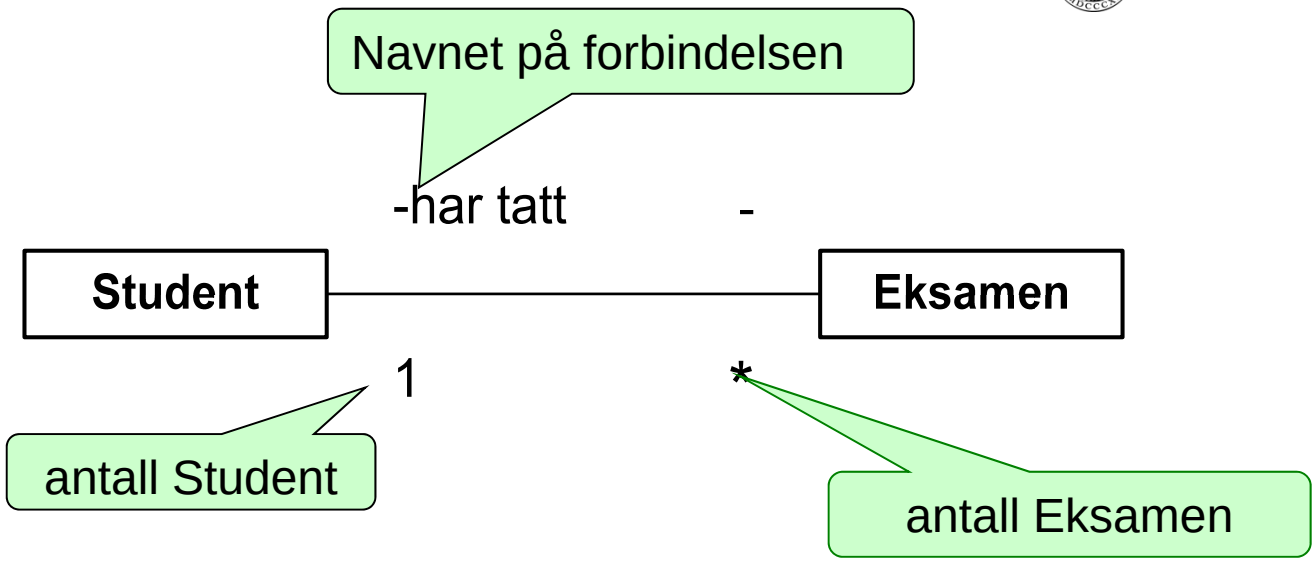
- Siden vi prøver å modellere virkeligheten en-til-en i programmet vårt, så blir disse to modellene like i utgangspunktet

# Forhold mellom klasser

UNIVERSITETET I OSLO

- Vi tegner et forhold (en assossiasjon) mellom to klasser dersom
  - de har med hverandre å gjøre logisk sett, og
  - vi i programmet vil kunne følge pekere for å få tilgang til variable og/eller metoder i de tilhørende objektene
- Vi angir også hvor mange objekter det maksimalt på kan være (på samme tidspunkt) på hver side av et slikt forhold
- Eksempel:
  - En **student** har tatt null eller flere **eksamener**
  - Med eksamen mener vi en avlagt enkelt-eksamen, dvs at en eksamen vil bare være tilknyttet en bestemt student.






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

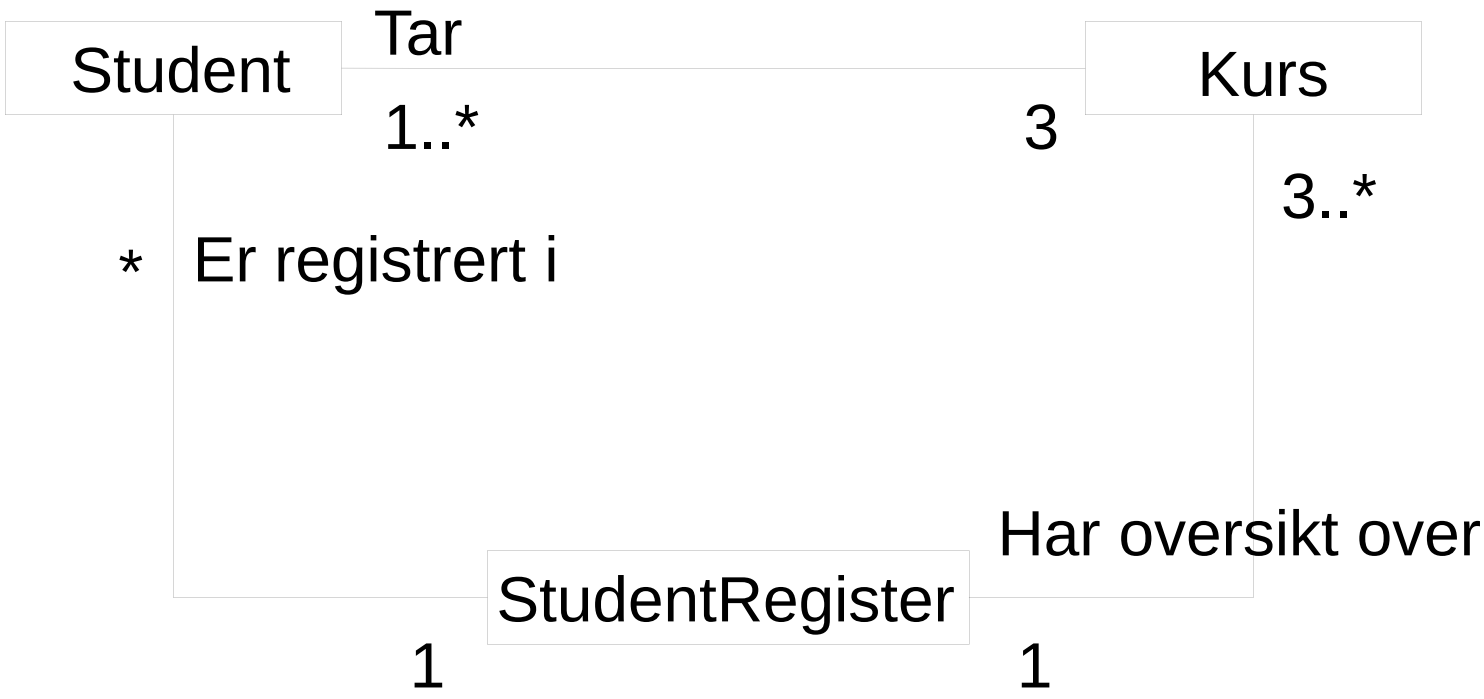
- Antallet objekter angis slik:

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

UNIVERSITETET I OSLO

# Studentregister

- Et studentregister holder orden på studentene og kursene, og en student tar 3 kurs hvert semester
- Hvert kurs har oversikt over studentene som tar kurset



UNIVERSITETET I OSLO

# 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

UNIVERSITETET I OSLO

# 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

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

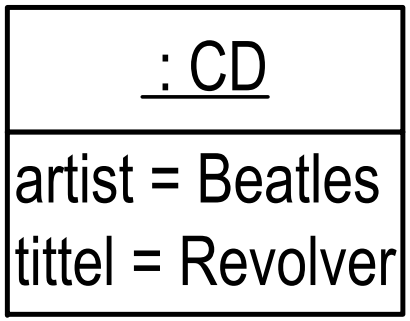
- Det er da ikke ’naturgitt’ hvilke forhold vi har i et klassediagram, det avhenger av hvilke spørsmål vi vil være interessert i å svare på.

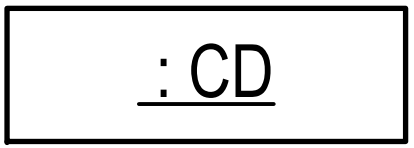
# Objekt-diagram

- 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

- To eller ett felt(er) i en boks
- Navnefeltet
  - objektnavn:klassenavn
    eller bare
  - **:**klassenavn
- Attributtfeltet (kan være tomt)
  - Navnet på sentrale objektvariable,
    evt også med verdier

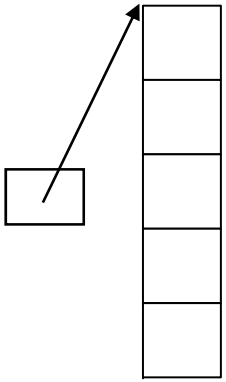


# Andre elementer i et objektdiagram

- Pekere

- Peker-arrayer

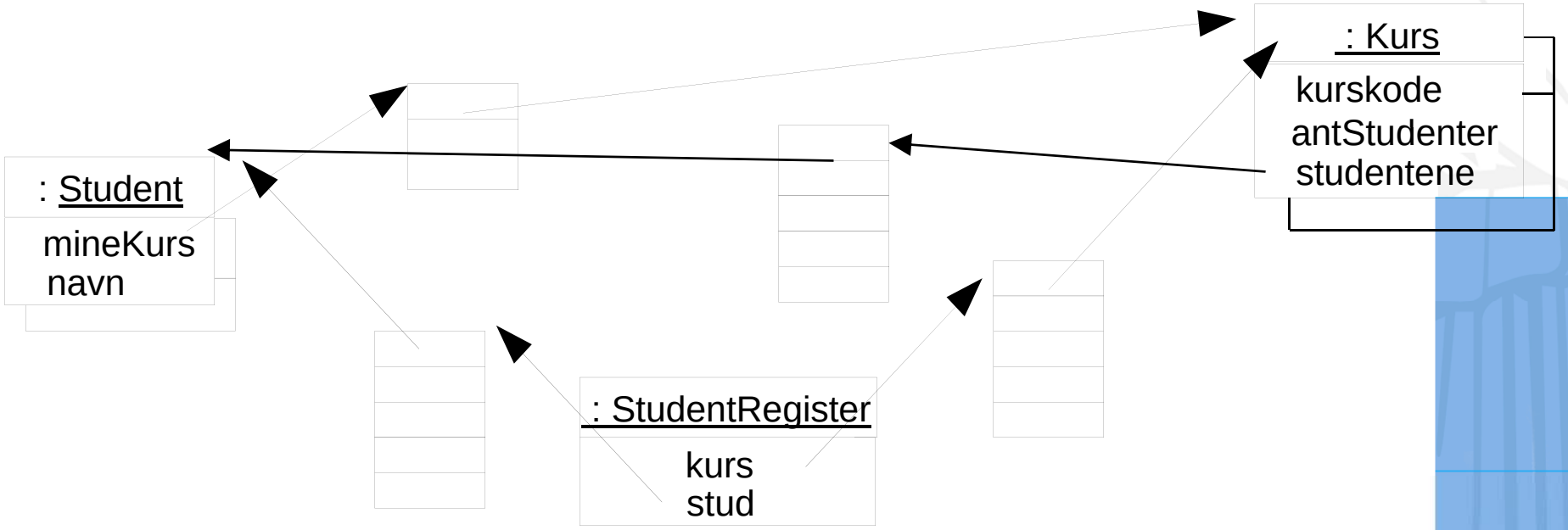


UNIVERSITETET
I OSLO

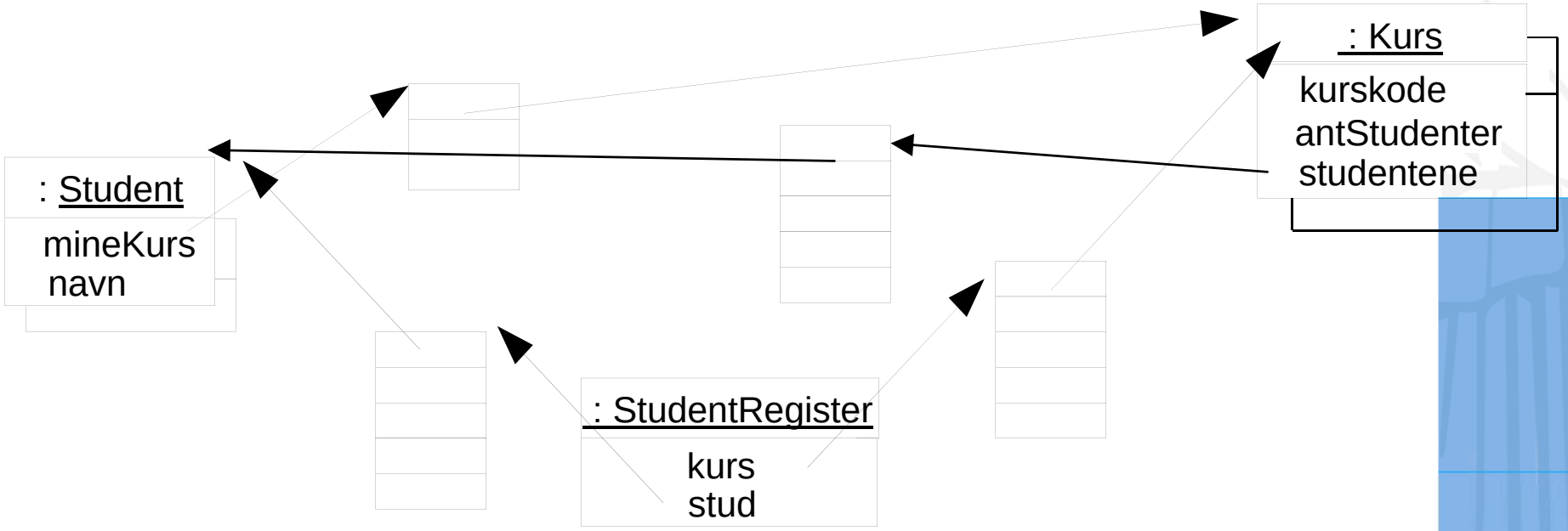
# Et helt studentregister med kurs, studenter og registeret

- 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

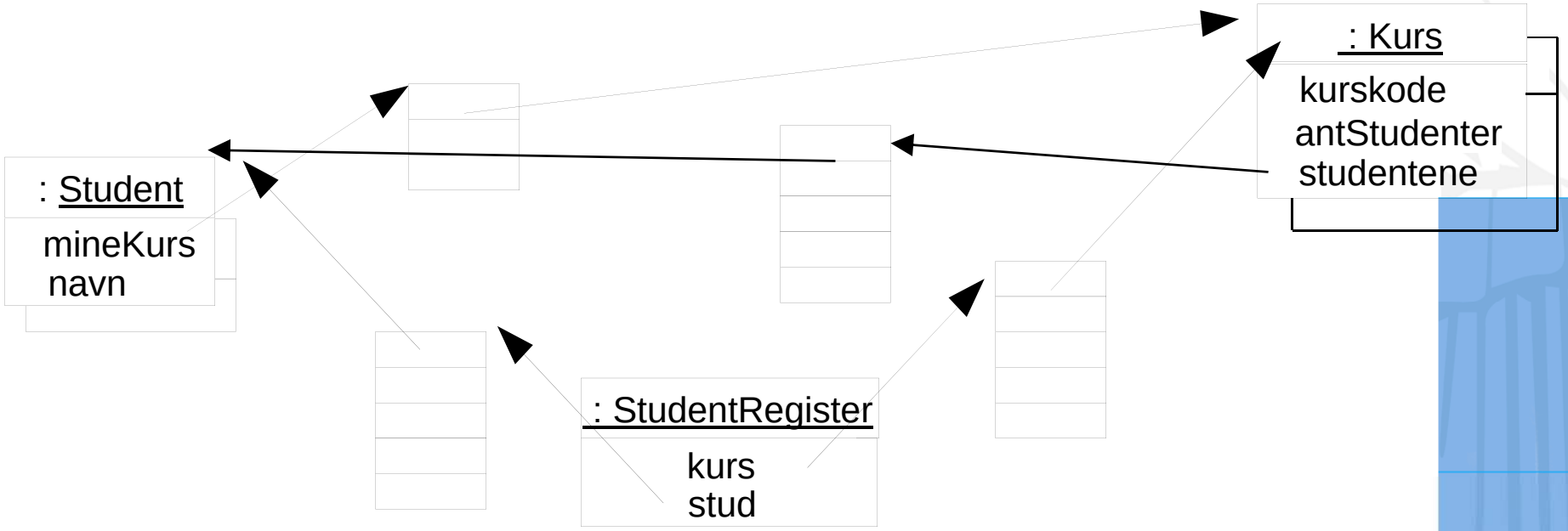


# Fra objektdiagram til Java: Student



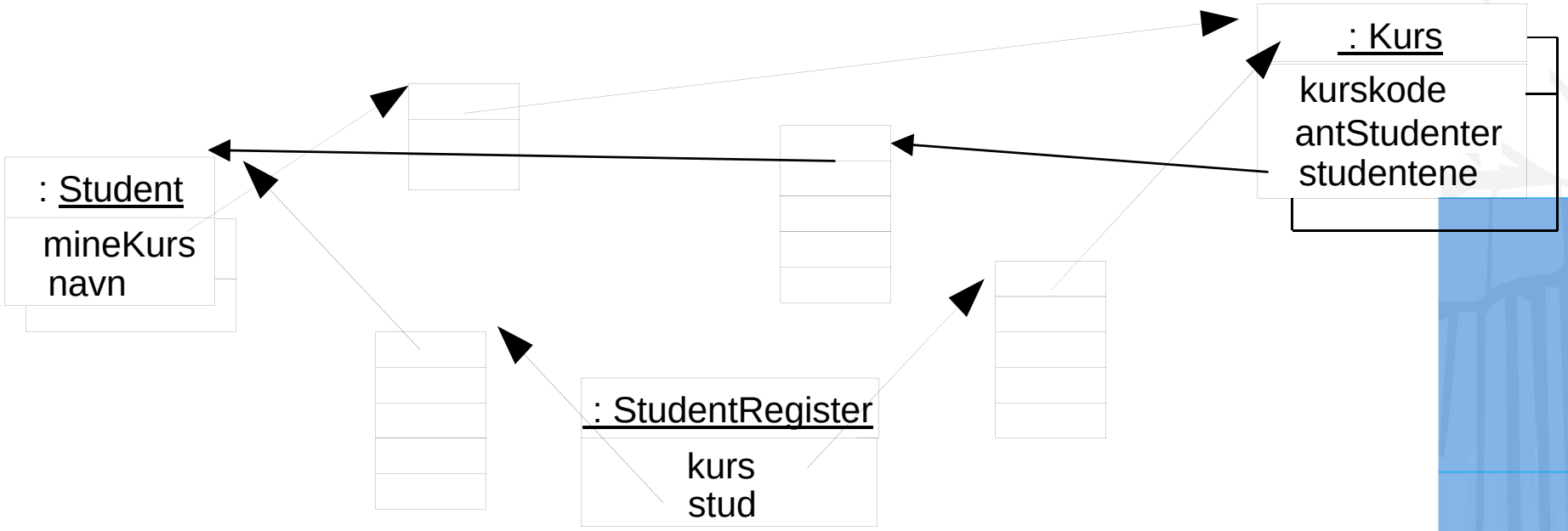

```
class Student {
    String navn;
    Kurs[] mineKurs;
}
```

# Fra objektdiagram til Java: Kurs



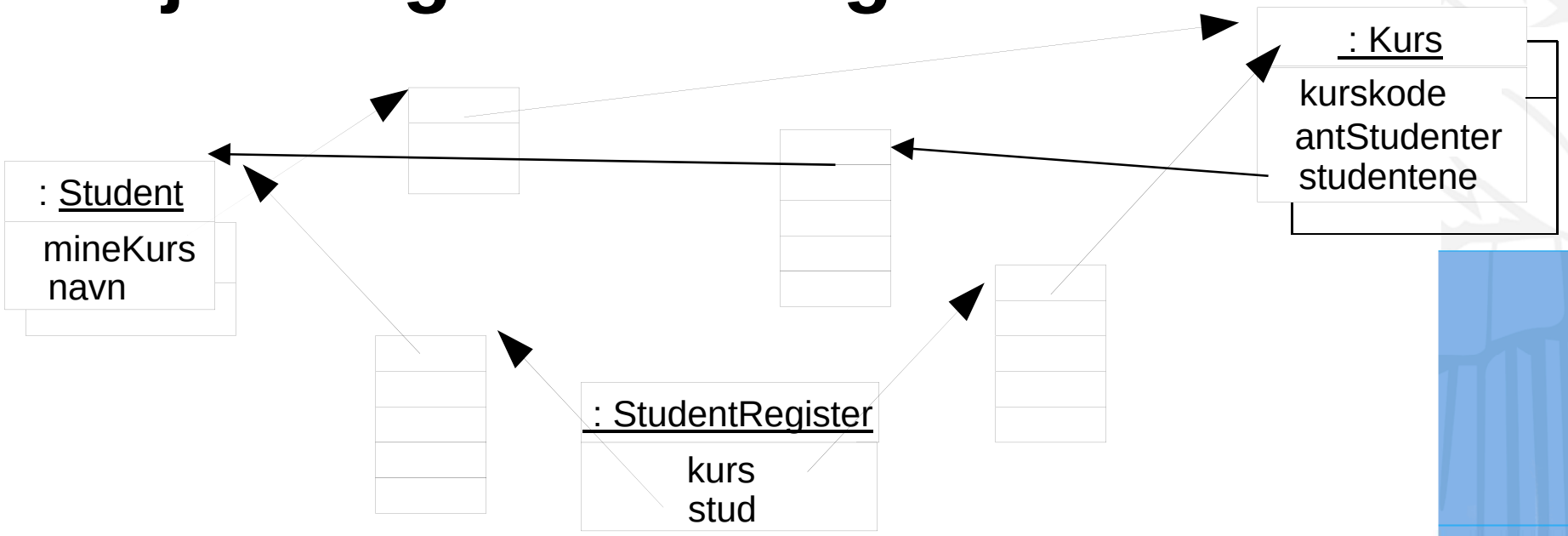

```
class Kurs {
    String kurskode;
    int antStudenter;
    Student[] studentene;
}
```

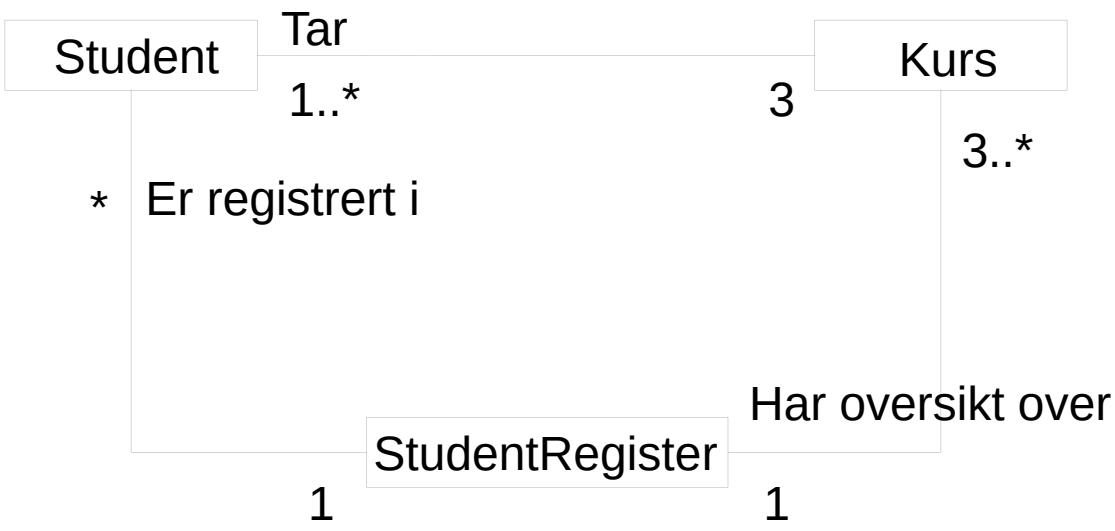
# Fra objektdiagram til Java: StudentRegister




```
class StudentRegister {
    Kurs[] kurs;
    Student[] stud;
}
```

# Sammenligning: Objekt- og klassediagram



# OPPSUMMERING

# 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)