UNIVERSITETET
I OSLO

INF1000 – Forelesning 8

Litt repetisjon: Metoder og klasser

Innkapsling av variable og metoder

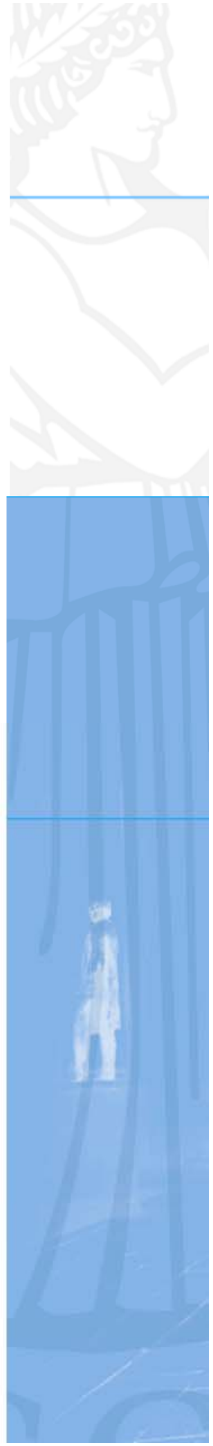
Hvordan gripe an et stort problem? 5 gode råd

Hva er en metode?

- En metode er en valgfritt antall programsetninger vi gir et navn
- All kode i programmet er inne i en metode
  (som igjen er inne i en eller annen klasse)
- Skille mellom
  - å *deklarere* en metode (= skrive Javakode for og kompilere)
  - *Utføre* en metode (det som skjer når vi kaller den)
  - Når vi deklarerer en metode, skjer det 'ingen ting '
- En metode blir utført hver gang den kalles fra koden i en annen metode:
  - da hopper utførelsen av programmet til starten av den kalte metoden
  - har den kalt metoden parametere, kopieres verdiene brukt i kallet til metodens parameter-variable (de er som lokale variable i den kalte metoden)

Hva skjer når vi kaller en metode?

- Når vi kaller en metode, blir det opprettet et **metodeobjekt** og vi kopierer over verdiene brukt i kallet til parameterne
- Dette metodeobjektet
  - inneholder alle lokale variabler og parameterne til metoden
  - når setningene i metoden utføres, brukes disse variablene og parametrene av metoden
  - metodeobjektet fjernes automatisk når metoden er ferdig utført og returnerer
- Merk forskjellen på å deklarere en metode, og at den utføres.



Hvorfor bruke metoder?

- Vi deler opp programmet i metoder fordi:
  - Noen program setninger brukes *flere* steder, eller:
  - Vi vil dele opp programmet i mindre deler
    - Ingen metode bør være lenger enn 30 linjer (og helst mindre)
  - Hver del gjør noe veldefinert som fremgår av navnet:
    - regner ut en bestemt formel
    - skriver ut en meny
    - leser noen data fra terminal eller fil
    - tegner ut opplysninger på skjermen
    - .....

Problemløsning med metoder

- Når vi har laget en metode, har vi laget en *ny operasjon*
- Vi kan i resten av koden tenke at vi nå har en slik operasjon tilgjengelig som om den var innebygd i Java
    - eks: skrive ut en meny, regne ut en bestemt formel,..
- Vi trenger nå ikke tenke på alle detaljene om *hvordan* denne operasjonen blir utført
- Vi har da laget et (lite) verktøy som kan gjenbrukes og lettere løse vårt større problem (hele systemet)
- Denne måte å programmere på heter *bottom-up* programmering og nyttes mye.
    - Eks: Java-biblioteket kan best forstås som en diger verktøykasse med nyttige operasjoner og datastrukturer vi kan (og ofte bør) bruke for å lage vårt program

UNIVERSITETET I OSLO

Metodeoppgave

Skriv en metode som teller opp hvor mange ganger tegnet i char-variabelen **`tegn`** forekommer i tekststrengen **`tekst`**, og returnerer dette antallet.

For eksempel skal kallet **`antallForekomster ("Engelske", 'e')`** returnere verdien 2. Hvis **`tekst == null`** så skal metoden returnere verdien 0.

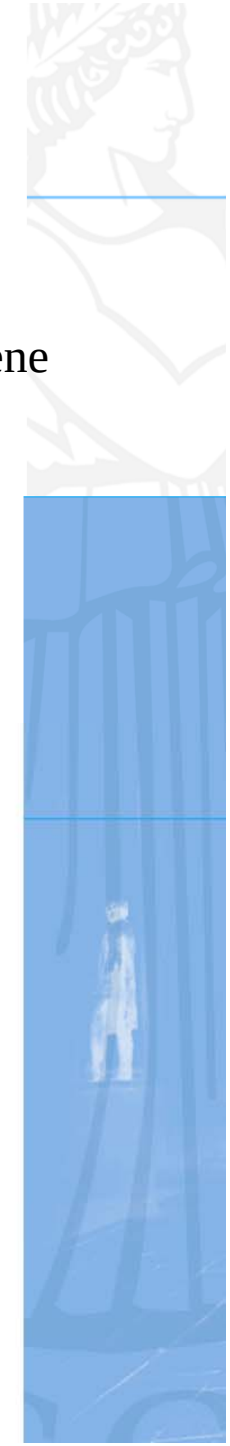
```
int antallForekomster (String tekst, char tegn) {
   if (tekst == null) return 0;

   int antall = 0;
   for (int i=0; i<tekst.length(); i++) {
      if (tekst.charAt(i) == tegn) antall++;
   }

   return antall;
}
```

Ny oppgave

Vi kan lage en sekvens med heltall på følgende måte: la de to første elementene i sekvensen begge være 1 og la deretter hvert element være summen av de to foregående. Den sekvensen vi da får starter slik: 1, 1, 2, 3, 5, 8, 13, ... og kalles Fibonacci-sekvensen.

Vi skal nå lage en metode som beregner de n første elementene i Fibonacci-sekvensen og returnerer disse i en heltallsarray av lengde n+1 (det første elementet i arrayen, svarende til posisjon 0, skal ikke benyttes).

Posisjon k i arrayen som returneres skal inneholde det k'te elementet i Fibonacci-sekvensen for k=1,....,n. Vi kan anta at $n \geq 2$.



```
int[] fibonacci (int n) {

   int[] a = new int[n+1];

   a[1] = 1;
   a[2] = 1;

   for (int i=3; i<n+1; i++) {
      a[i] = a[i-1] + a[i-2];
   }

   return a;

}
```

Hva er en klasse?

- En klasse er en beskrivelse av hvordan *ett* objekt av en bestemt type i vårt problem er.
  - Inneholder variable som beskriver egenskaper for ett slikt objekt – eks:
    - Navn, adresse, studiepoeng, kurs... for klassen Student
    - Registreringsnummer, eier, type, årsmodel for klassen Bil
  - Inneholder metoder som er fornuftig handlinger for ett slikt objekt – eks:
    - skrivUtVitnemål(), meldPåEmne(),.. i klassen Student
    - beregnÅrsavgift(), skiftEier(),.. i klassen Bil

Skille mellom deklarasjon og bruk av en klasse

- Når vi deklarerer en klasse (= skriver Javakode for) skjer det ’ingen ting’ i programmet
- Når vi oversetter og starter opp programmet vårt med javac og java, skjer ’lite’:
  - De variable og metodene det står static foran er tilgjengelig
  - Ingen kode (med unntak av main) utføres
- Først når vi sier **new** på en klasse, får vi laget et objekt av klassen
  - Objektet inneholder alle variable og metoder som ikke har static foran deklarasjonen (objekt-variable og – metoder)
  - Når vi sier new, kaller vi en konstruktør-metode i klassen, og først når den er ferdig, returnerer new det med det nye objektet

Oppgave

Vi skal lage en konstruktør til klassen Student. Konstruktøren skal ha studentens navn som parameter og skal initiere objektvariabelen **navn**.

Vi skal også skrive en objektmetode **void økPoeng(int poeng)** i klassen Student som øker antall studiepoeng med parameterens verdi.

I klassen StudentTest skal vi så sørge for å få initiere arrayen **uioStud** med 32000 Student-objekter. Studentobjektene i arrayen **uioStud** skal ha hvert sitt studentnavn **Stud-1**, **Stud-2**, **Stud-3**, ... **Stud-32000**. Dermed skal f.eks. **uioStud[252]** peke på et Student-objekt hvor studentens navn er satt lik **Stud-253**.

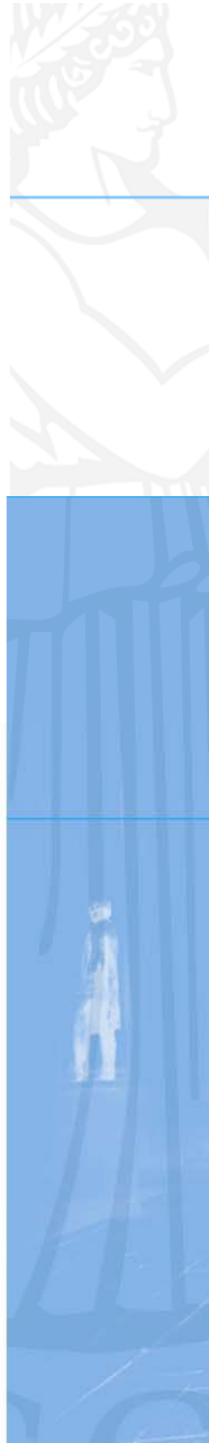
Til sist skal du sette antall studiepoeng til 30 for hver av studentene **Stud-1**, ..., **Stud-25000**.

```
class Student {
      String navn;
      int antallStudiepoeng = 0;
      // Her skal du skrive konstruktøren

      Student (String navn) {
          this.navn = navn;
      }



      // Her skal du skrive objektmetoden økPoeng

      void økPoeng (int poeng) {
          antallStudiepoeng += poeng;
      }

}
```

```
class StudentTest {
   Student[] uioStud = new Student[50000];

   public static void main (String[] args) {
      // Her skal du opprette 32000 Student-objekter med navn Stud-1,
      // Stud-2,...,Stud-32000 og legge dem inn i uioStud-arrayen
      StudentTest st = new StudentTest();
      for (int i=0; i<32000; i++) {
          st.uioStud[i] = new Student(”Stud-” + (i+1));
      }
      // Her skal du øke antall studiepoeng med 30 for studentene
      // med studentnavn Stud-1, Stud-2, ...., Stud-25000.

      for (int i=0; i<25000; i++) {
          st.uioStud[i].økPoeng(30);
      }

   }// end main
```

Forskjeller mellom klasser og metoder

- Begge lager objekter når de kalles, men:
- Et metode-objekt:
  - fjernes når metoden returnerer
  - inneholder ’bare’ variable og parametere som alle er skjult for resten av programmet
- Et objekt laget med **new** fra en klasse:
  - er i hukommelsen etter at det er laget (så lenge det minst er en peker som peker på det)
  - kan inneholde både metoder og variable, som kan nyttes av resten av programmet (med en peker og .)



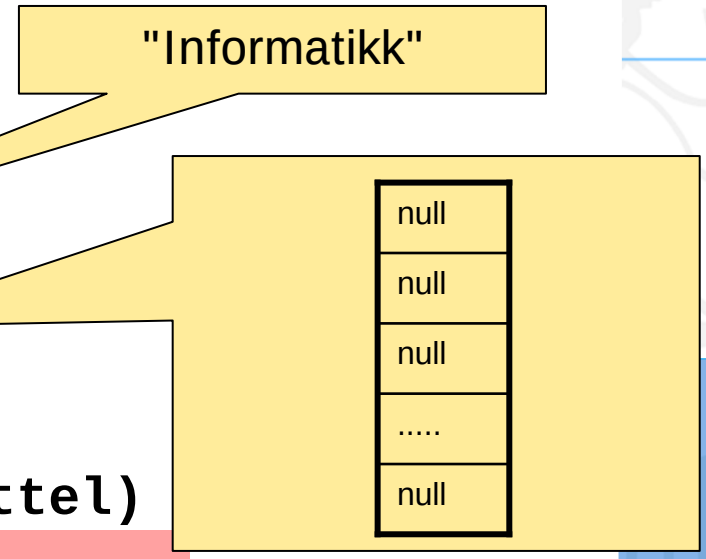
Hva skriver programmet ut?

```
class StudentSystem {
   public static void main (String[] args) {
      Studieprogram inf = new Studieprogram("Informatikk");
      Studieprogram bio = new Studieprogram("Biologi");

      new Student("Kari Olsen", inf);
      new Student("Eva Larsen", inf);
      new Student("Per Jensen", bio);

      bio.liste();
      inf.liste();
   }
}
```

```
class Student {
   String navn;

   Student (String navn, Studieprogram sp) {
      this.navn = navn;
      sp.meldInnStudent(this);
   }

   String somTekst() {
      return navn;
   }
}
```

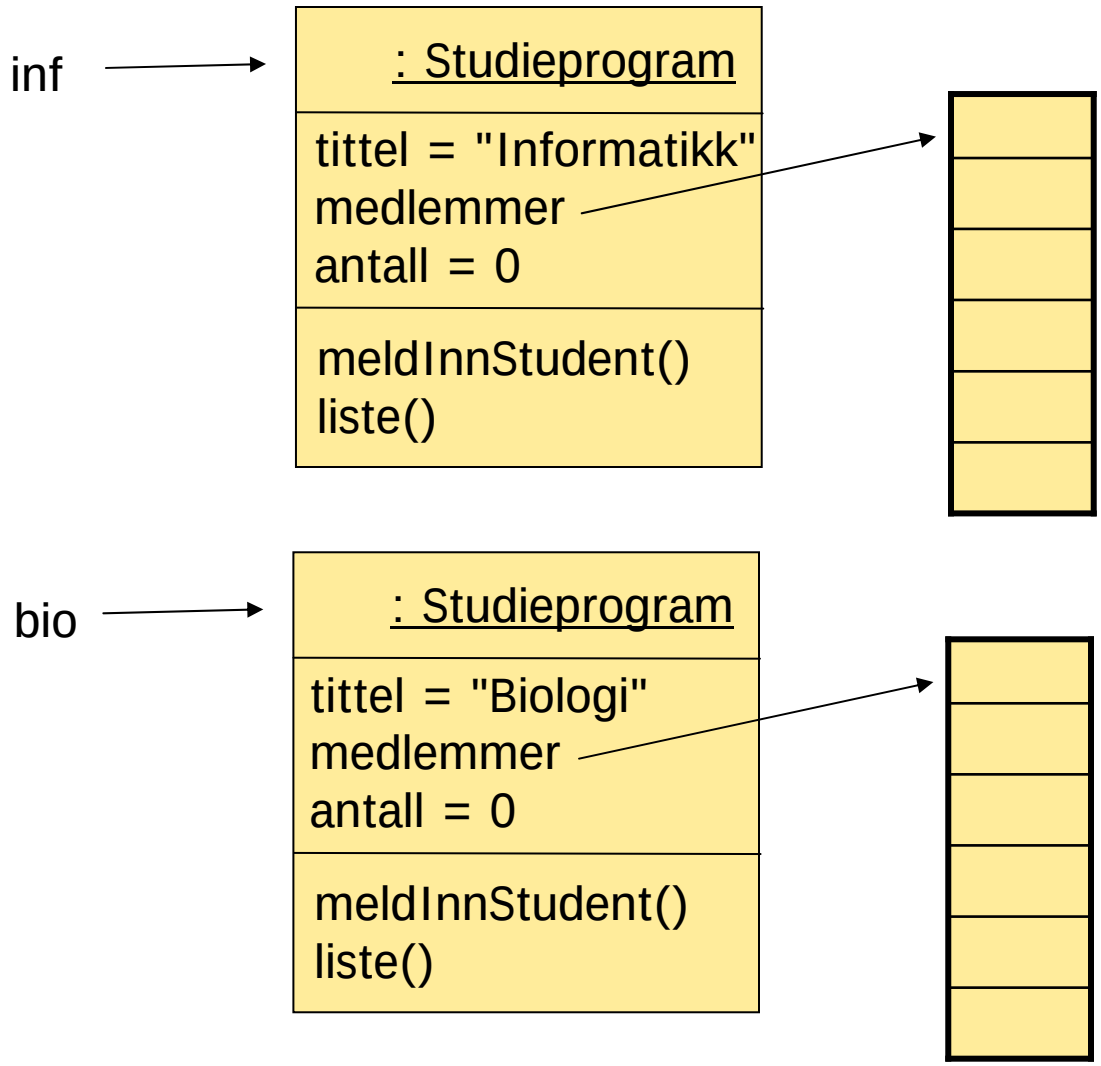
```
class Studieprogram {
   String tittel;
   Student[] medlemmer;
   int antall;

   Studieprogram (String tittel) {
      this.tittel = tittel;
      medlemmer = new Student[100];
      antall = 0;
   }

   void meldInnStudent(Student stud) {
      medlemmer[antall] = stud;
      antall++;
   }

   void liste() {
      for (int i=0; i<antall; i++) {
         System.out.println(medlemmer[i].somTekst());
      }
    }
}
```

new Studieprogram("Informatikk");

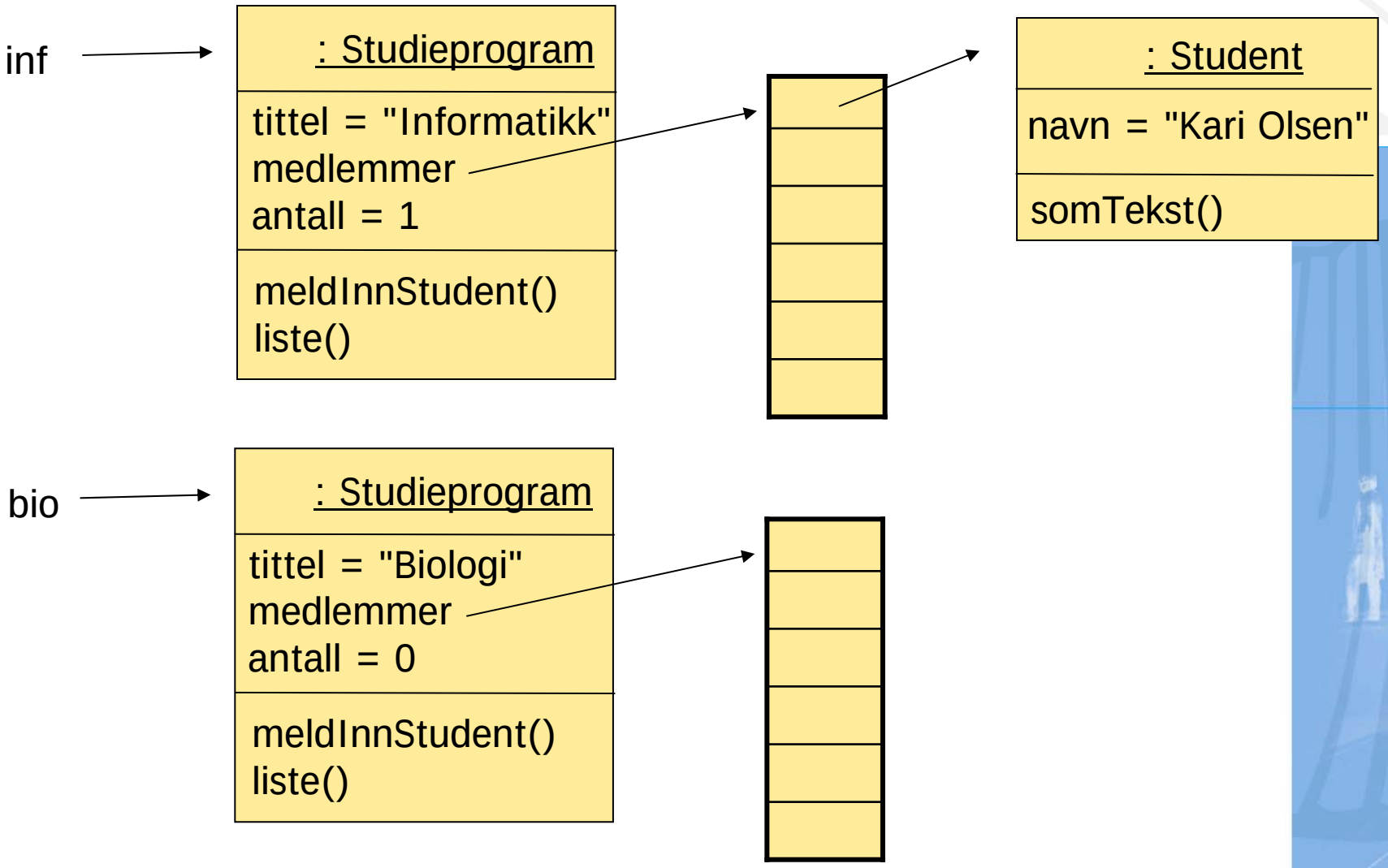



```
class Studieprogram {
   String tittel;
   Student[] medlemmer;
   int antall;

   Studieprogram (String tittel)
     this.tittel = tittel;
     medlemmer = new Student[100];
     antall = 0;
   }

   ......

}
```

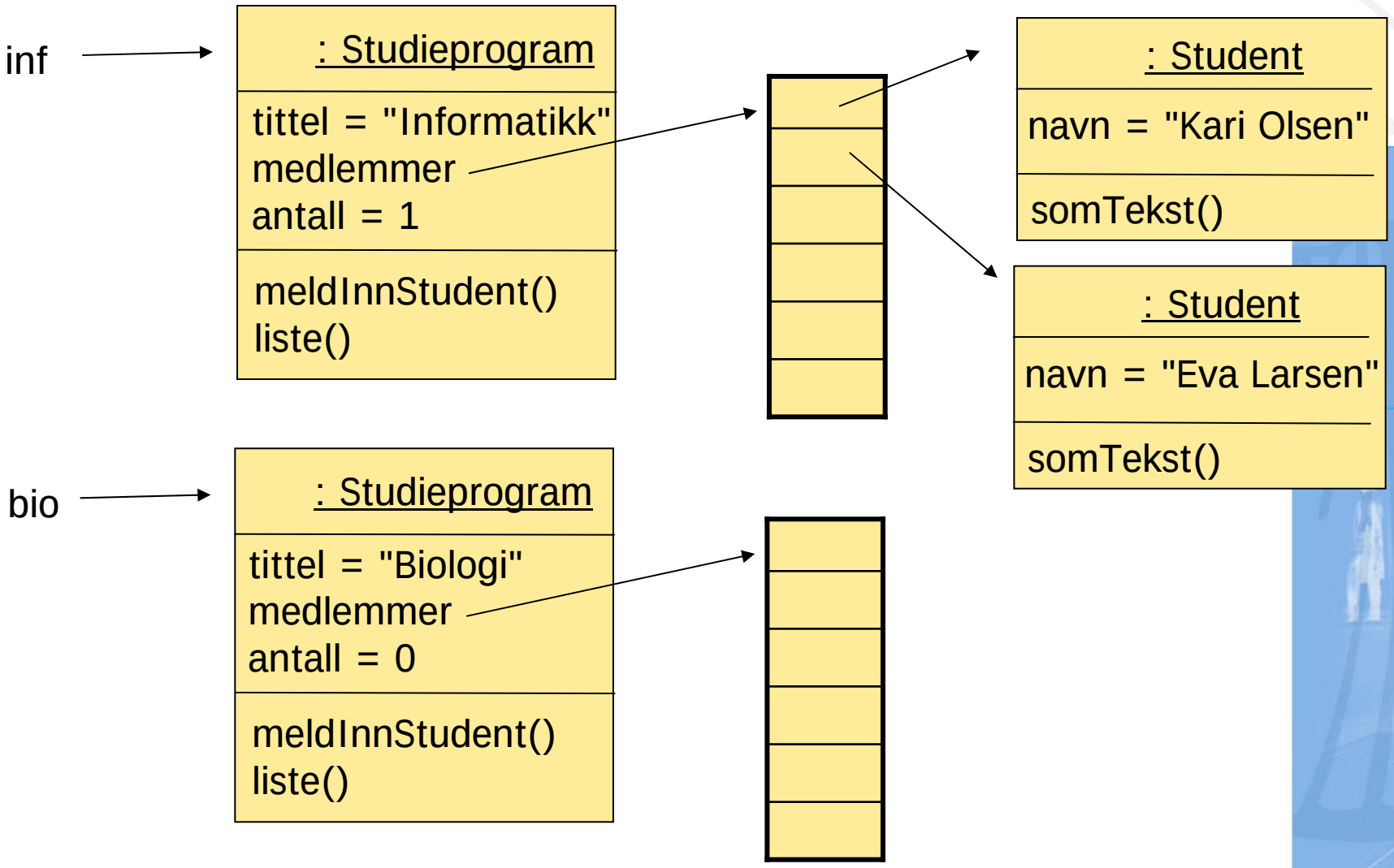
Studieprogram inf = new Studieprogram("Informatikk");
Studieprogram bio = new Studieprogram("Biologi");



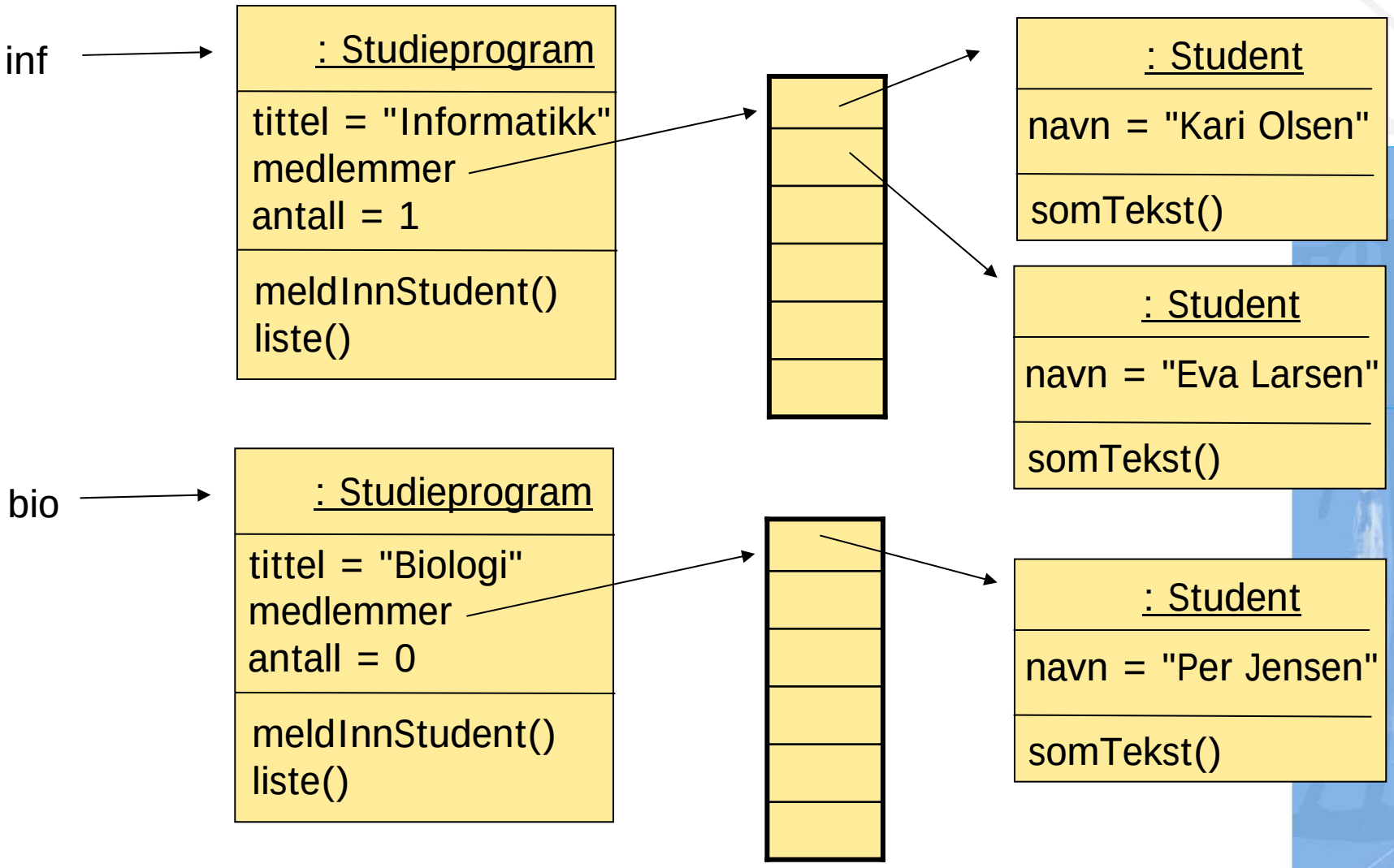
new Student ("Kari Olsen", inf);



new Student ("Eva Larsen", inf);

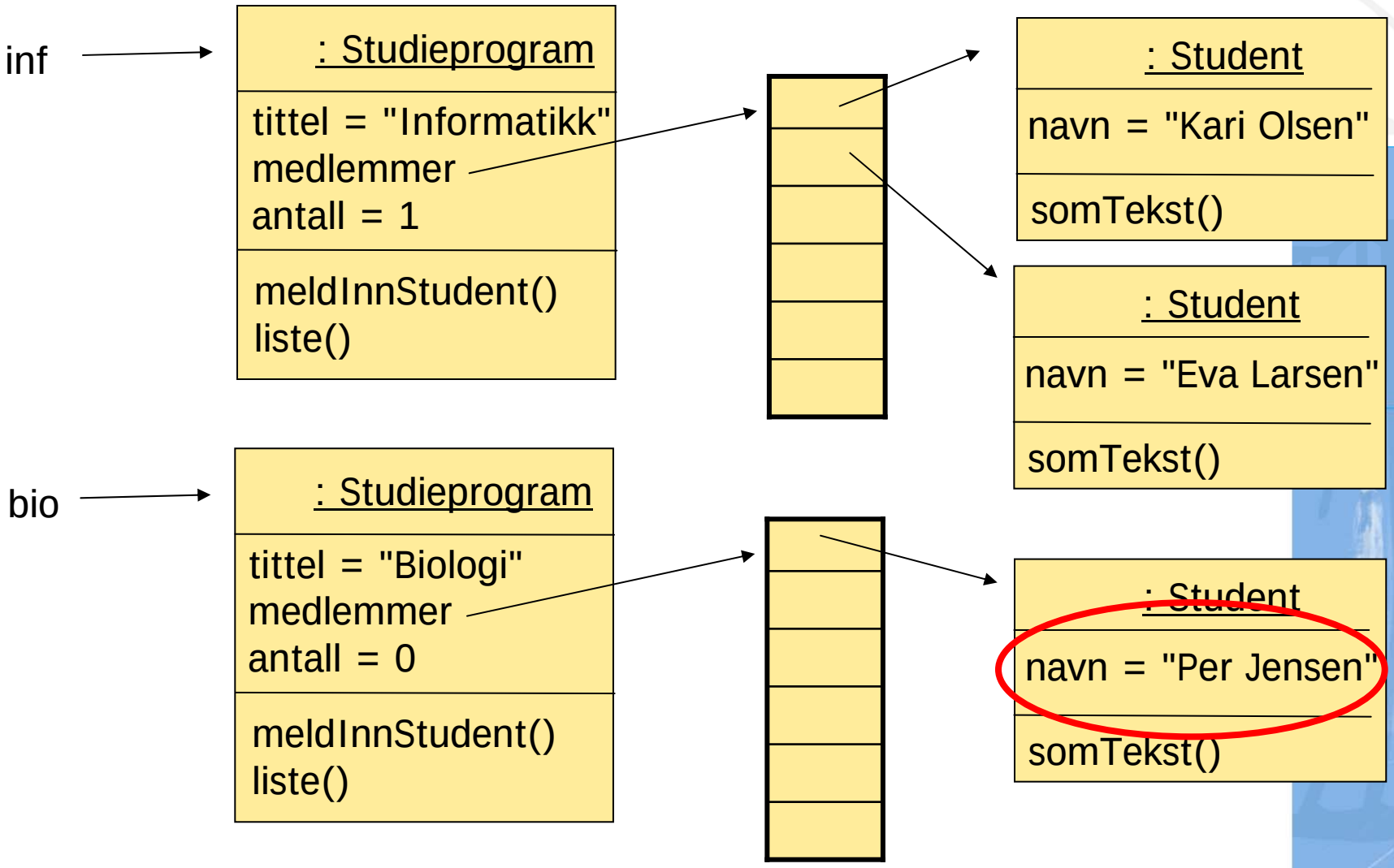


new Student ("Per Jensen", bio);

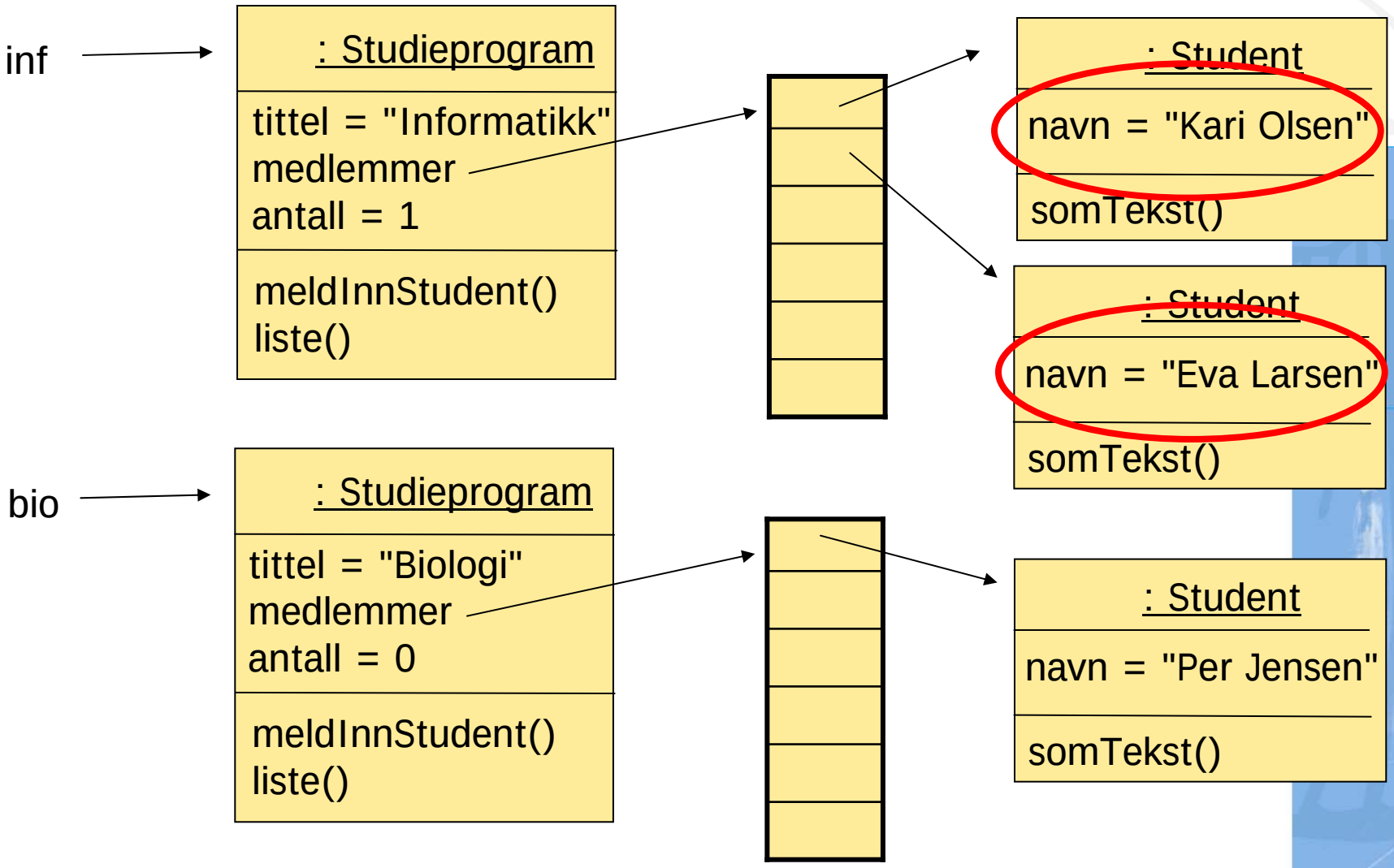


UNIVERSITETET I OSLO

bio.liste();



inf.liste();


inf
: Studieprogram
tittel = "Informatikk"
medlemmer
antall = 1
meldInnStudent()
liste()
: Student
navn = "Kari Olsen"
somTekst()
: Student
navn = "Eva Larsen"
somTekst()
bio
: Studieprogram
tittel = "Biologi"
medlemmer
antall = 0
meldInnStudent()
liste()
: Student
navn = "Per Jensen"
somTekst()

Svar:

Per Jensen
Kari Olsen
Eva Larsen

Ikke alt i et objekt bør være synlig fra resten av programsystemet - innkapsling

- Vi ønsker ofte at resten av systemet bare skal se deler av et objekt
  - eks: **`int saldo`** i et Konto-objekt bør være skjult, resten av programmet skal bare bruke metodene **`settInn()`** og **`taUt()`**.
- Vi kan regulere tilgangen til variable og metoder ved å sette en av følgende foran en variabel- eller metodedeklarasjon:
  - **`private`**
  - **`public`**
  - **`protected`**

Klasse med innkapsling

UNIVERSITETET I OSLO

```
public class Studieprogram {
  private String tittel;
  private Student[] medlemmer;
  private int antall;

  public Studieprogram (String tittel) {
    this.tittel = tittel;
    medlemmer = new Student[100];
    antall = 0;
  }

  public void meldInnStudent(Student stud) {
    medlemmer[antall] = stud;
    antall++;
  }

  public void liste() {
    for (int i=0; i<antall; i++) {
      System.out.println(medlemmer[i].somTekst());
    }
   }
}
```

```
    public hentTittel() {
      return tittel;
    }

    public hentAntall() {
      return antall;
    }

    . . .
}
```

For ”små” systemer hvor alle java-filene ligger på samme filområde, gjelder:

- **ingen modifikator:**
  - deklarasjonen er tilgjengelige for alle klasser i den samme pakken/katalogen, men utilgjengelig for klasser i andre pakker/kataloger.
- **`private:`**
  - deklarasjonen er bare synlig fra kode i metoder deklarert i *samme klasse*, usynlig/sperret for all annen kode
- **`protected:`**
  - deklarasjonen er synlig i samme klasser og subklasser, og fra klasser i samme pakke/katalog, men utilgjengelig fra klasser i andre pakker/kataloger
- **`public:`**
  - deklarasjonen er synlig for all annen kode
- Slik delvis sperring av adgang til variable/metoder, sikrer oss at vi kan beskytte data for tilgang utenfra.

Klassemetoder og objektmetoder

- Klassemetoder (static-metoder)
  - Definert selv om det ikke er laget noen objekter av klassen
  - Kan ”ses” av alle objekter av klassen
  - Kan brukes av andre gjennom dot-notasjon: <klassenavn>.metode(...)
  - Har ikke tilgang til objektvariable eller objektmetoder
- Objektmetoder
  - Bare definert i objekter av klassen
  - Kan ”ses” av objektet som metoden befinner seg i
  - Kan brukes av andre gjennom dot-notasjon: <peker>.metode(...)
  - Har tilgang til alle variable (både klassevariable og objektvariable) og alle metoder (både klassemetoder og objektmetoder)

Klassevariablenes levetid

```
class Person {
  static int ant = 0;
  int alder;
  String navn;

  Person(){ ant ++; }
}
```

Denne variabelen blir deklarert når klassen Person blir referert til for første gang under kjøringen av programmet.

Variabelen lever helt til programmet avsluttes.

Første gang klassen Person blir referert til = første gang programeksekveringen "møter på" Person-klassen, f.eks. :

```
.... new Person() ....

.... i = Person.ant +  ....
```

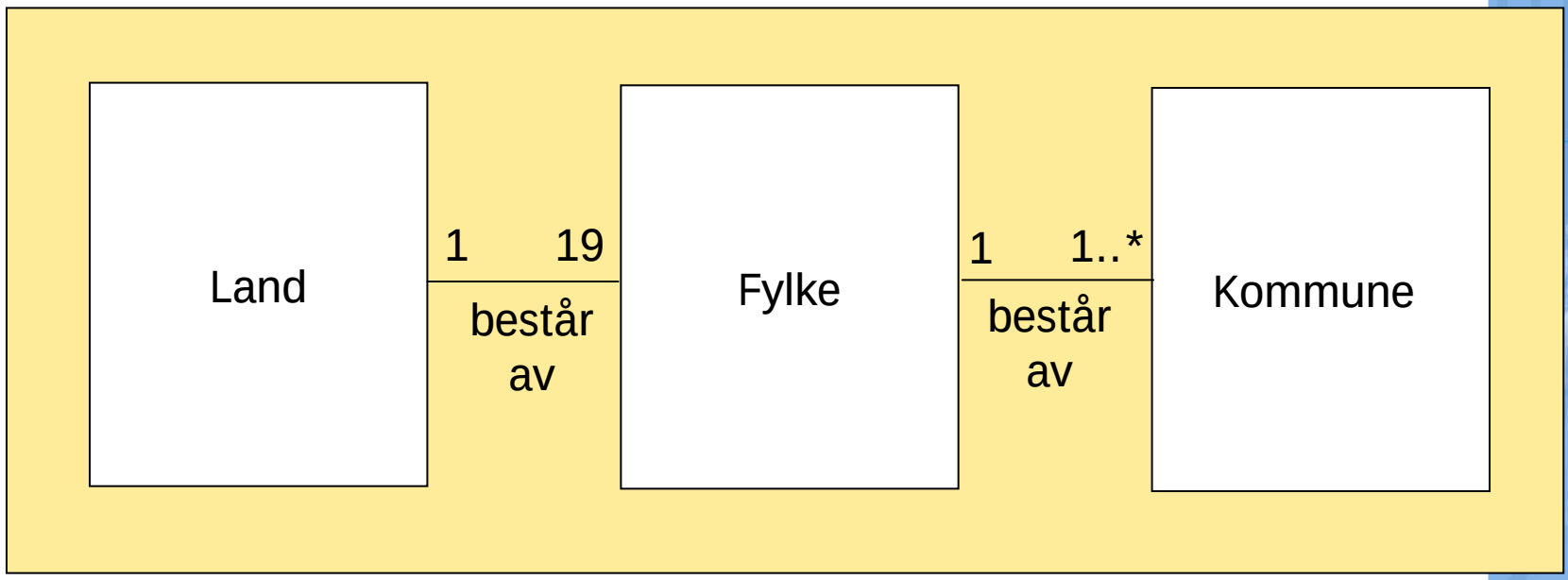
Råd 1: Programmer ”ovenfra ned”

- Hvilke klasser skal være med?
  - Les oppgaven
  - Se etter substantiver
  - Lag klassediagram
- Bestem datastruktur
  - Hvordan er input-dataene?
  - Fyll inn de mest sentrale variablene
  - Trengs nye klasser?
- Følg programflyten når du bestemmer metoder
  - Skriv først metodene på ”toppnivå” f.eks. en kommandoløkke
  - Kall på metoder ved behov, selv om disse ennå ikke er skrevet
  - Skriv metodene du kaller på, og fortsett til programmet er ferdig

UNIVERSITETET I OSLO

Oppgave

Norge består av 19 fylker som hver består av et antall kommuner. Anta at vi modellerer dette ved hjelp av tre klasser **`Land`**, **`Fylke`** og **`Kommune`** slik at Norge representeres ved et objekt av klassen **`Land`**, hvert fylke er representert ved et objekt av klassen **`Fylke`** og hver kommune er representert ved et objekt av klassen **`Kommune`**. Lag et UML klassediagram som viser forholdet mellom de tre klassene **`Land`**, **`Fylke`** og **`Kommune`**, og få på riktige antall i hver ende av forholdene.



Velg datastruktur etter hva som skal gjøres!

- I objekt-orientert programmering
  - *tenker vi i form av objekter*
  - *men programmer i form av klasser*
- Spør: Hva kan objektene gjøre for meg?
- Prøv å gruppere data etter objekter som "eier" dem
  - variable og metoder som logisk hører sammen bør ligge samlet
  - variable og metoder som ikke har noe med hverandre å gjøre bør holdes godt atskilt

UNIVERSITETET I OSLO

Gruppering etter ”eier”

```
String[] navn = new String[100];
String[]  fnr = new String[100];
int[] tlfnr = new int[100];
```

```
class Person {
  String navn;
  String fnr;
  int tlfnr;
}
Person [] personreg = new Person[100];
```

Ikke objektorientert:
Info om person splittet opp i tre arrayer

Info om person samlet i samme objekt

UNIVERSITETET I OSLO

Data og metoder hører sammen

```
... data om studenter...
... data om ansatte ...
... data om kurs ...
... student-metoder ...
... ansatt-metoder ...
... kurs-metoder ...
```

```
class Student {
   ... data om studenter ...
   ... student-metoder ...
}
class Ansatt {
   ... data om ansatte ...
   ... ansatt-metoder ...
}
class Kurs {
   ... data om kurs ...
   ... kurs-metoder ...
}
```

Metoder og data som hører sammen bør samles

- Lett å se hvilke metoder som jobber på hvilke data
- Lett å kopiere alt som har med personer å gjøre (data + metoder) til andre programmer

UNIVERSITETET I OSLO

Valg av datamodell: nytt eksempel

Gitt fil med opplysninger om antall registrerte tilfeller det var av tre ulike sykdommer i Norge :

	INFLUENSA	KYSSESYKE	MENINGITT
1950	...	...	...
1951	...	...	...
1952	...	...	...
.			
.			
2000	...	...	...

- Hvordan er det naturlig å modellere dette?
- Beste datastruktur avhenger av hva du skal bruke dataene til!

Gruppere tellinger relatert til samme sykdom

```
class Sykdom {
    String sykdomsNavn;
    int[] antallTilfeller = new int[51];
}
```

Gruppere tellinger foretatt samtidig

```
class Aarsdata {
    int antInfluensa;
    int antKyssesyke;int antMeningitt;
}
```

Ingen gruppering – tre arrayer

```
int[] influensatilfeller = new int[51];
int[] kyssesyketilfeller = new int[51];
int[] meningittilfeller = new int[51];
```

Ingen gruppering – en 2D-array

```
int[][] sykdomstilfeller = new int[3][51];
```

Råd 2: Metoder "utenfra og inn"

- Hva er input og output til metoden du skal skrive?
- Input:
  - Eventuelle parametere til metoden
  - Kan også være klassevariable/objektvariable
- Output:
  - Eventuell returverdi fra metoden
  - Kan også være modifikasjoner av klassevariable/ objektvariable (f.eks. endring av innholdet i en HashMap).

Råd 3: Deleger oppgaver

- Stykk opp oppgavene og fordel dem
- Dermed blir hver enkelt del mer oversiktlig
  - faren for feil minker
  - lettere å finne feil senere
- Ofte lurt: Deleger operasjoner på data til objekter som er ”nærme” dataene
- Ikke overdriv delegering!
  - hvert objekt trenger ikke metoder for å lese fra terminal!
  - av og til bedre å gjøre ting sentralt og kalle på metoder i objektene for å oppdatere deres variable

Ideen bak objektorientering

- Hvert objekt skal ha sin bestemte og naturlige oppgave
- Kollektivt samarbeid om å løse oppgavene
- Objektene opprettes som instanser av klasser
- Objektene samarbeider ved å sende meldinger:
  - kaller hverandres metoder
  - overfører informasjon i form av parametere og returverdier
- Metodekallene har en entydig mottager som overtar ansvaret for oppgaven

Helhet og deloppgaver i OOP

- *Vi trenger ikke å ha oversikt over hele programmet eller hele datastrukturen* når vi skriver en metode
- Vi ”skifter hatt” og ser systemet gjennom øynene til hver av aktørene for seg
  - Når vi er kelner, beskriver vi kelneren ut fra kelnerens perspektiv, når vi er hovmesteren ser vi det ut fra hans perspektiv osv.
- Vi programmerer en klasse ut fra klassens perspektiv og glemmer da resten av helheten

Objekter svarer til programmer

- Vi kan tenke oss at hvert objekt av en gitt klasse svarer til en ”kjøring av klassen”
- Et objekt kan *samarbeide* med alle det kjenner til ved metodekall
- Vi starter ett av ”programmene” ved å kalle klassens main-metode
- Deretter: objektene *opprettes* når programmet kjører

Råd 4: Formater koden

UNIVERSITETET I OSLO

```
    class Eksempel {
public static void main (String [] args) {
    int x = 0;
  for (int i=0; i<10; i++) {
x = x + 1;
      } if (x < 0)
  {System.out.println("Det var rart");
   }}}
```

DÅRLIG!

```
class Eksempel {
  public static void main (String [] args) {
    int x = 0;
    for (int i=0; i<10; i++) {
      x = x + 1;
    }
    if (x < 0){
      System.out.println("Det var rart");
    }
  }
}
```

BRA!

Råd 5: Ingen skam å snu!

- Programmer blir til ved at vi jobber litt her og der
  - Vi kan bruke mange runder før vi er fornøyd
- Vi finner ofte ut at vi trenger flere klasser
  - eller at en klasse bare er “i veien” og fjerner den
- Det endelige programmet kan ha andre klasser og metoder enn vi startet med
- Pass likevel på å holde programmet kompilerbart!
  - Lag tomme metoder som du kan fylle ut siden
  - Hold koden ryddig