UNIVERSITETET I OSLO

IN1010 våren 2019

Onsdag 30. januar

Arv og subklasser – del 1

Stein Gjessing
Institutt for informatikk
Universitetet i Oslo

Når du har lært om subklasser kan du programmere med:

Første uke:

- Spesialisering (og generalisering)
- Klasse-hierarkier - arv
- Referanser (pekere) – sterk typing
- Nøkkelordet instanceof
- Konvertering av referanser
  - Klassen Object
- Abstrakte klasser

Andre uke (onsdag 6. februar)

- Virtuelle metoder - polymorfi
  - Nøkkelordet super
- Gjenbruk av klasser og begreper
  - Ved sammensetning (komposisjon)
  - Ved arv
- Konstruktører

Tredje uke (onsdag 13. februar)

- Interface

Hva er en subklasse?

- En klasse, Kl, beskriver objekter med visse felles egenskaper
- En subklasse av Kl, Sub, beskriver objekter som har de samme egenskapene (som beskrevet av Kl), men i tillegg er Sub-objektene noe mer, de har flere og / eller mer spesielle egenskaper . . .

- class Kl { . . . }
- class Sub extends Kl { . . . }

Nytt Java nøkkelord: extends

Navn på klasser og subklasser (Kl og Sub her) bør best mulig beskrive hva klassene representerer

Eksempel: Universitetsregister

I et mini-system for Universitetet i Oslo skal alle studenter registreres med navn og telefonnummer (åtte siffer), samt hvilket studieprogram de er tatt opp til. Det skal være mulig for studenter å bytte program.

Systemet skal også inneholde informasjon om de ansatte ved universitetet, nemlig navn, telefonnummer, lønnstrinn og antall arbeidstimer per uke. Teknisk-administrativt ansatte har en arbeidsuke på 37,5 timer, mens vitenskapelig ansatte har 40-timers arbeidsuke.

Alle personer skal behandles som . . . . . .

Eksempel: Universitetsregister

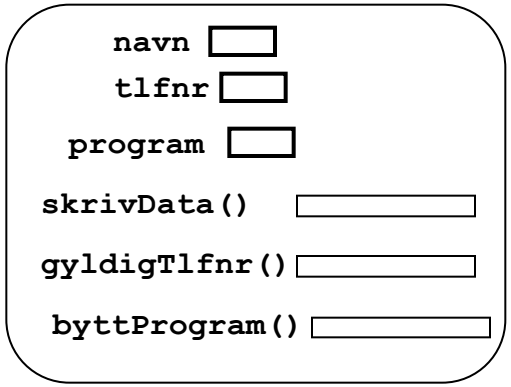
I et mini-system for Universitetet i Oslo skal alle studenter registreres med navn og telefonnummer (åtte siffer), samt hvilket studieprogram de er tatt opp til. Det skal være mulig for studenter å bytte program.

Systemet skal også inneholde informasjon om de ansatte ved universitetet, nemlig navn, telefonnummer, lønnstrinn og antall arbeidstimer per uke. Teknisk-administrativt ansatte har en arbeidsuke på 37,5 timer, mens vitenskapelig ansatte har 40-timers arbeidsuke.

Alle personer skal behandles som . . . . . .

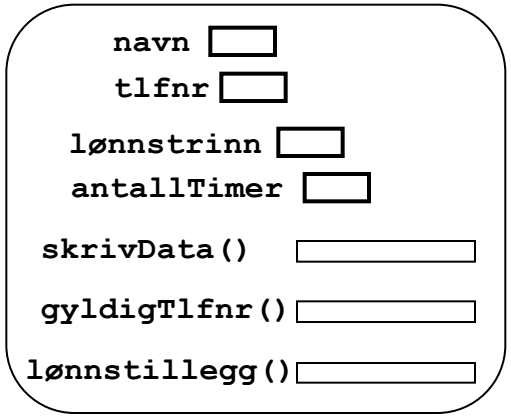
Substantivmetoden

Klassen Student

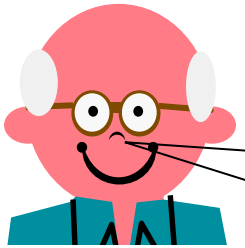



```
class Student {
        String navn;
        int tlfnr;
        String program;


        void skrivData() {
                System.out.println("Navn: " + navn);
                System.out.println("Telefon: " + tlfnr);
                System.out.println("Studieprogram: " + program);
        }


        boolean gyldigTlfnr() {
                return tlfnr >= 10000000 && tlfnr <= 99999999;
        }

        void byttProgram(String nytt) {
                program = nytt;
       }
}
```

Klassen Ansatt




```
class Ansatt {
          String navn;
          int tlfnr;
          int lønnstrinn;
          int antallTimer;

          void skrivData() {
                    System.out.println("Navn: " + navn);
                    System.out.println("Telefon: " + tlfnr);
                    System.out.println("Lønnstrinn: " + lønnstrinn);
                    System.out.println("Timer: " + antallTimer);
          }

          boolean gyldigTlfnr() {
                    return tlfnr >= 10000000 && tlfnr <= 99999999;
          }

          void lønnstillegg(int tillegg) {
                    lønnstrinn += tillegg;
         }
}
```



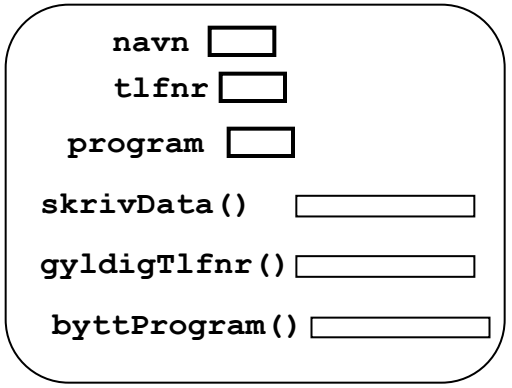
Hm –
på forrige lysark var det ingen egenskaper som var ”private” eller public” ?

OK –
Vi kommer tilbake til det på lysark 25

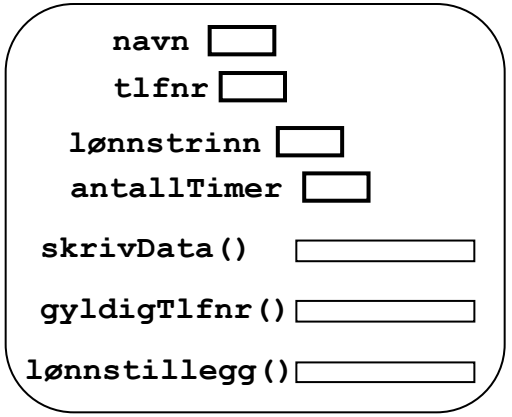
Og det er jo slik at om det ikke står noe, så er egenskapene synlige i hele fil-katalogen

Student vs Ansatt

- Felles variable:
  - **`navn`**, **`tlfnr`**
- Egne variable:
  - Student: **`program`**
  - Ansatt: **`lønnstrinn`**, **`antallTimer`**
- Felles metoder:
  - **`gyldigTlfnr()`**
- Lignende metoder:
  - **`skrivData()`**
- Egne metoder:
  - Student: **`byttProgram(String nytt)`**
  - Ansatt: **`lønnstillegg(int tillegg)`**



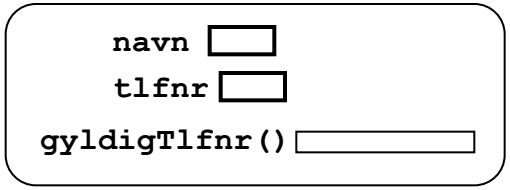
Student-objekt



Ansatt-objekt

Klassen Person

Kan samle det som er felles
i en egen, mer generell, klasse




```
class Person {
        String navn;
        int tlfnr;

        boolean gyldigTlfnr() {
                return tlfnr >= 10000000 && tlfnr <= 99999999;
        }
}
```

Klassen Person beskriver alt som er felles for studenter og ansatte

Student og Ansatt som subklasser

Kan nå gjøre Student og Ansatt til *subklasser* av Person:

```
class Student extends Person {
        String program;

        void byttProgram(String nytt) {
                program = nytt;
        }
}

class Ansatt extends Person {
        int lønnstrinn;
        int antallTimer;

        void lønnstillegg(int tillegg) {
                lønnstrinn += tillegg;
        }
}
```

Angir at klassene Student og Ansatt er subklasser (= utvidelser) av klassen Person.

Hva med skrivData()?
- Kommer tilbake til denne…

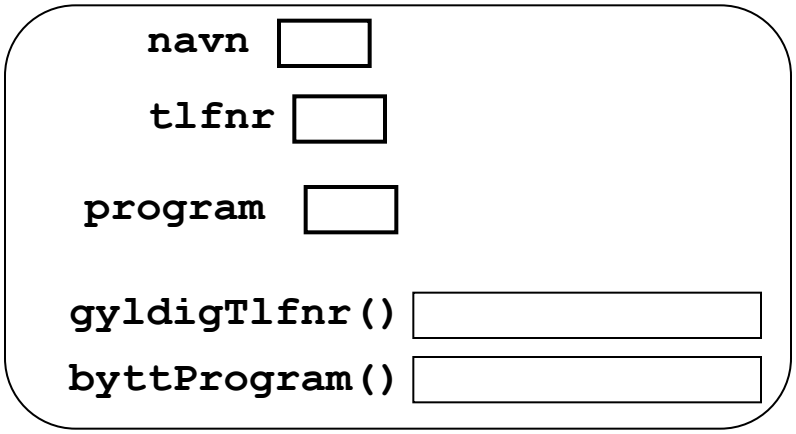
Nytt Java nøkkelord: extends

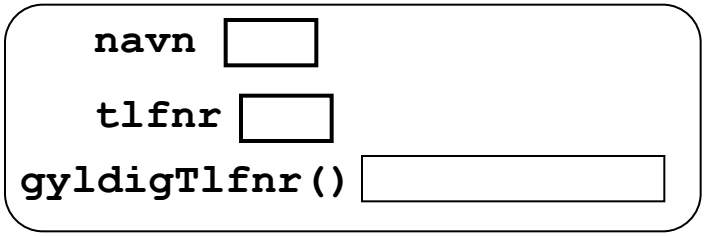
```
class Student {
  String navn;
  int tlfnr;
  String program;

  boolean gyldigTlfnr() {...}
  void byttProgram(String nytt) {...}
}
```

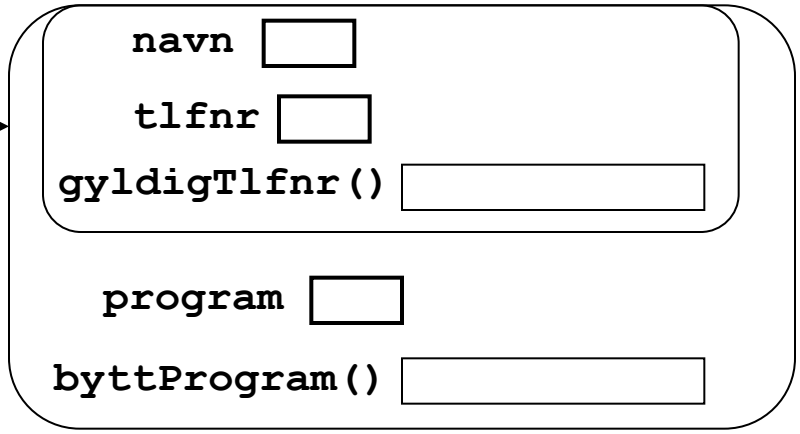
```
class Person {
  String navn;
  int tlfnr;
  boolean gyldigTlfnr() {...}
}
```

```
class Student extends Person {
  String program;
  void byttProgram(String nytt) {...}
}
```

Eksempler på objekter av klassene







Bruk av en subklasse

Vi kan bruke variable og metoder i en subklasse på samme måte som om vi hadde definert alt i én klasse:

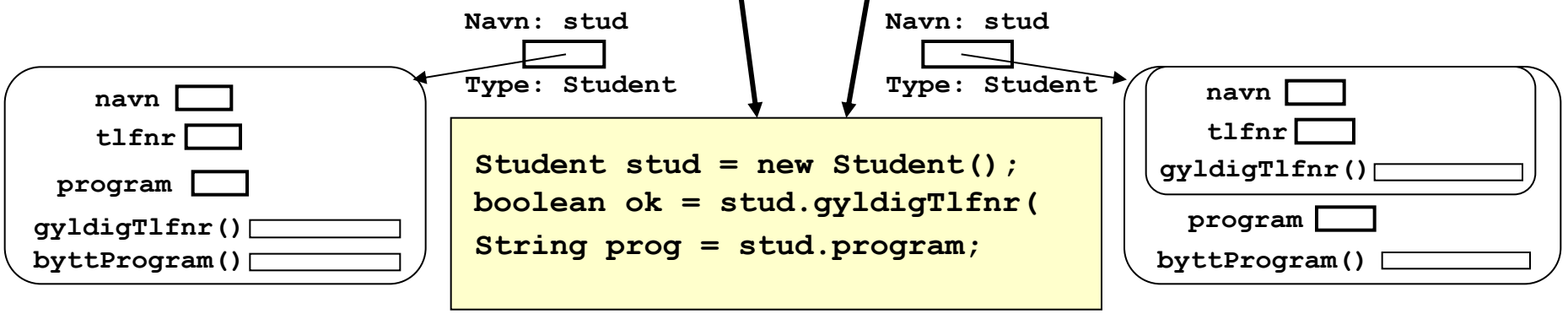
Uten bruk av subklasser (før):

```
class Student {
  String navn;
  int tlfnr;
  String program;

  boolean gyldigTlfnr() {...}
  void byttProgram(String nytt) {...}
}
```

eller *Med* bruk av subklasser (nå):

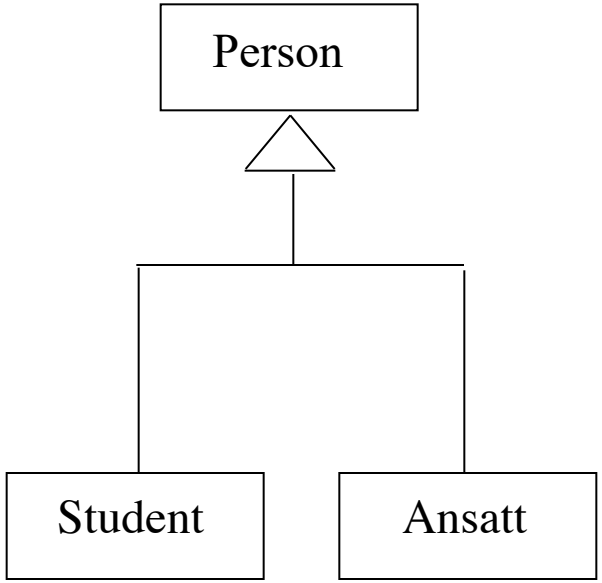
```
class Person {
  String navn;
  int tlfnr;
  boolean gyldigTlfnr() {...}
}
```

```
class Student extends Person {
  String program;
  void byttProgram(String nytt) {...}
}
```



UML-notasjon for **subklassehierarki**

```
class Person {
  String navn;
  int tlfnr;
  boolean gyldigTlfnr() {...}
}

class Student extends Person {
  String program;
  void byttProgram(String nytt)
    {...}
}

class Ansatt extends Person {
   int lønnstrinn;
   void lønnstillegg (int tillegg){
       lønnstrinn += tillegg;
   }
}
```



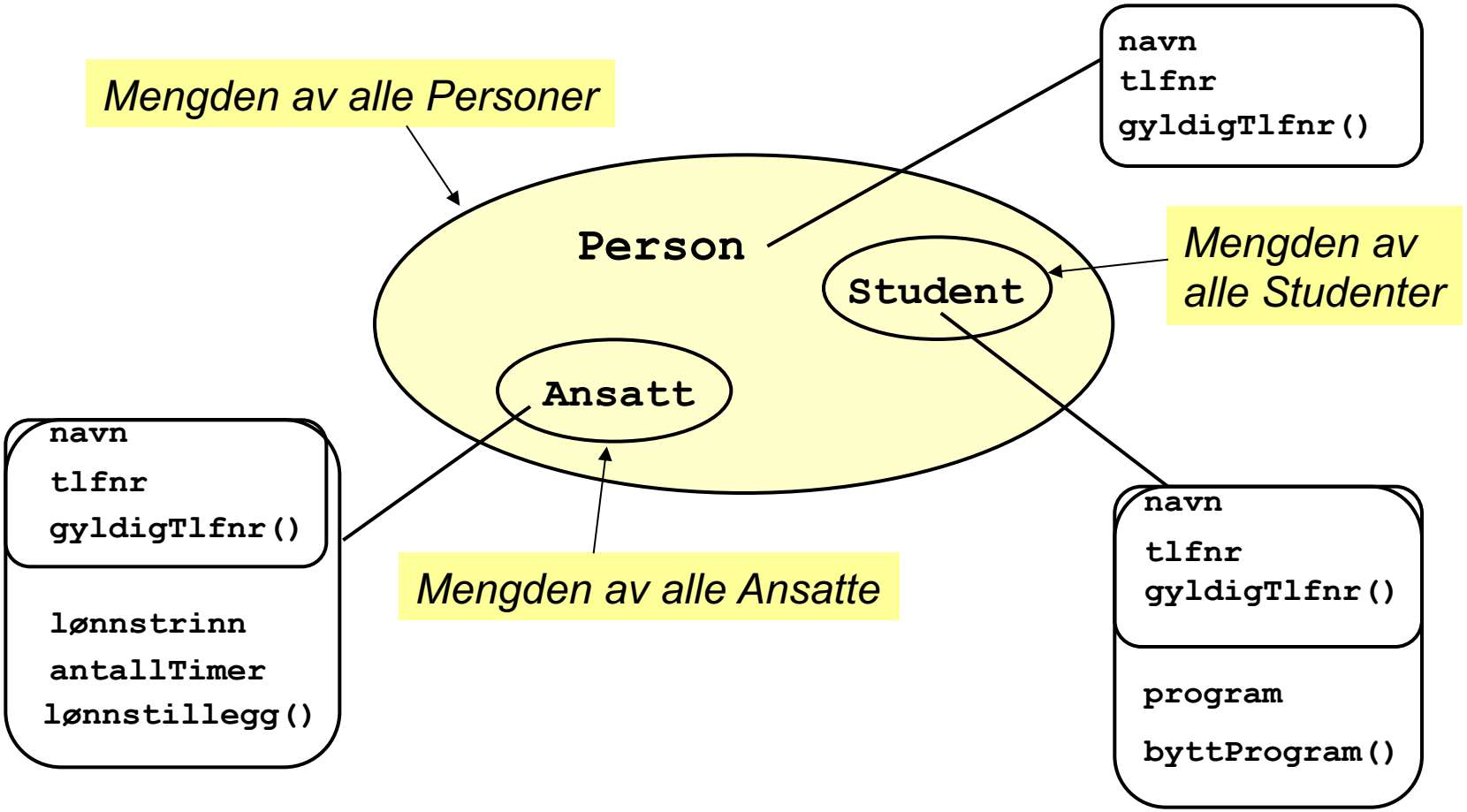
I IN1010 er riktig UML-notasjon ikke viktig (men subklassehierarkier er viktig)

Igjen: Hva er en **subklasse**?

- En subklasse er en klasse som bygger på en allerede spesifisert klasse, og som dermed ***arver*** dennes egenskaper i tillegg til å utvide med egne egenskaper (metoder/variable/konstanter).
- En subklasse er altså en mer **spesialisert** utgave av klassen den bygger på.
- Klassen vi bygger på kalles en ***superklasse***.

Dette fant Ole-Johan Dahl og Kristen Nygaard på i ca. 1963, og laget programmeringsspråket Simula

Spesialisering - Generalisering

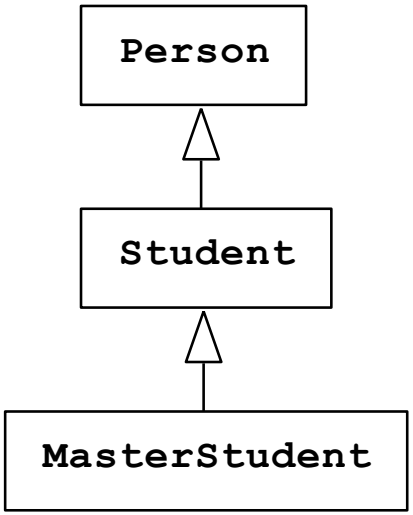


Sub-klasse ~ Sub-mengde (del-mengde)

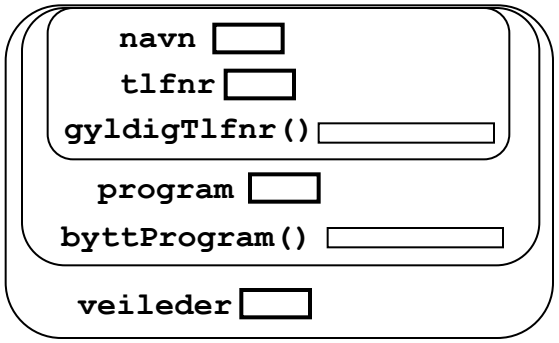
Klasse-hierarkier

Det er mulig å definere subklasser av en subklasse (etc.):

```
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() {...}
}

class Student extends Person {
    String program;
    void byttProgram(String nytt){...}
}

class MasterStudent extends Student {
    String veileder;
}
```

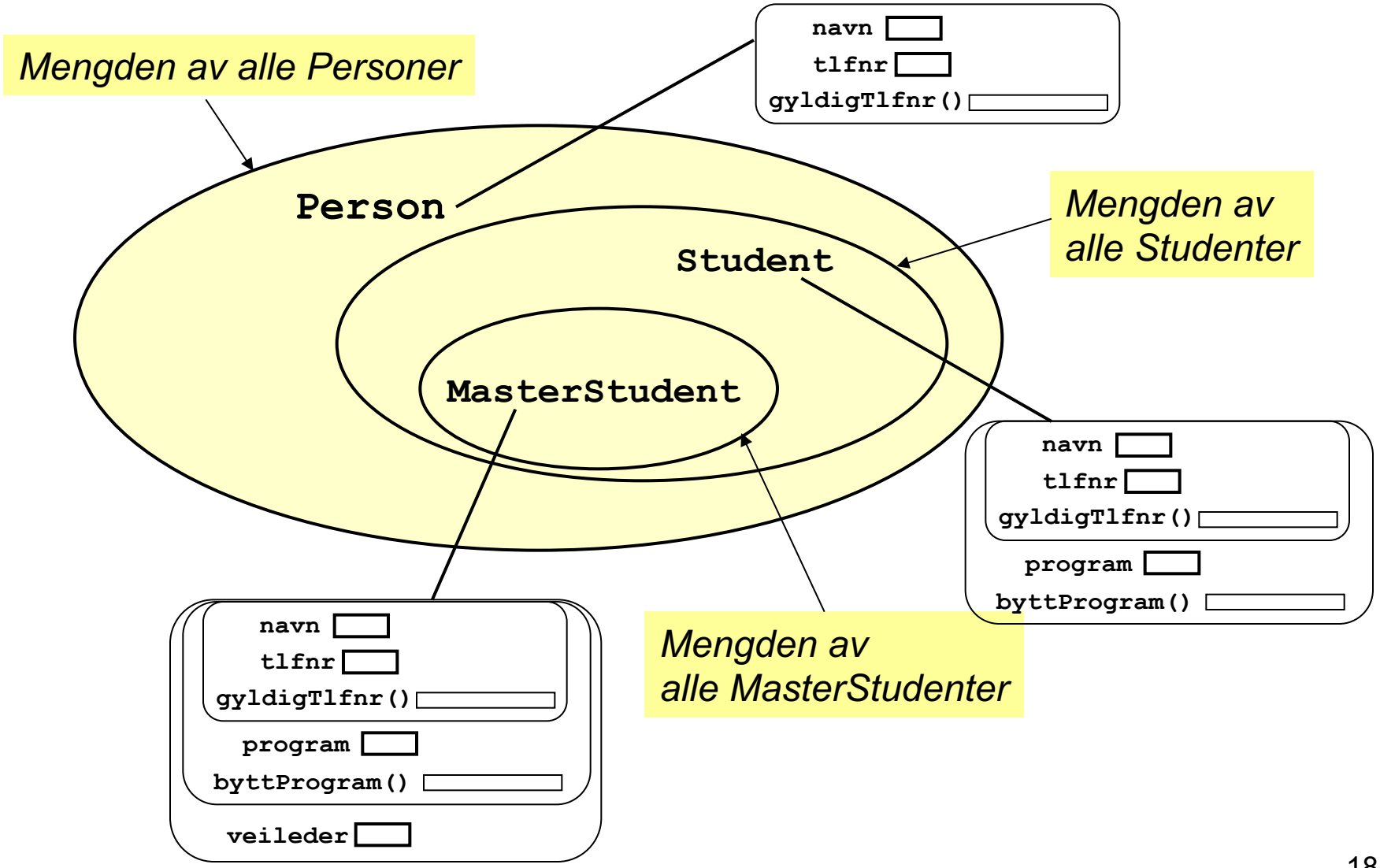


Obs: Her er MasterStudent en subklasse av *både* Student og Person, og arver egenskaper fra begge disse.



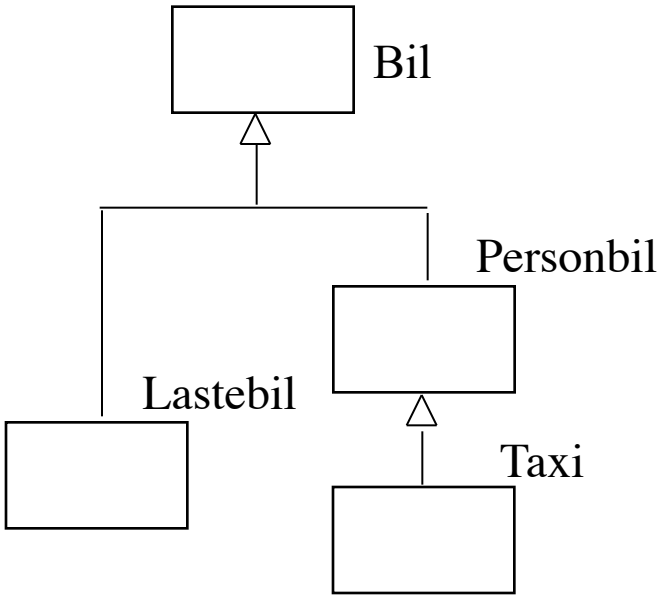
MasterStudent-objekt

Generalisering – spesialisering



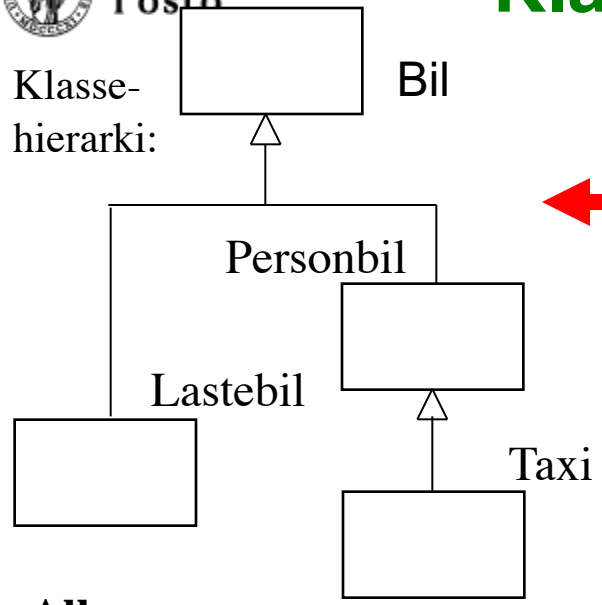
Klasser - Subklasser

Klassehierarki:

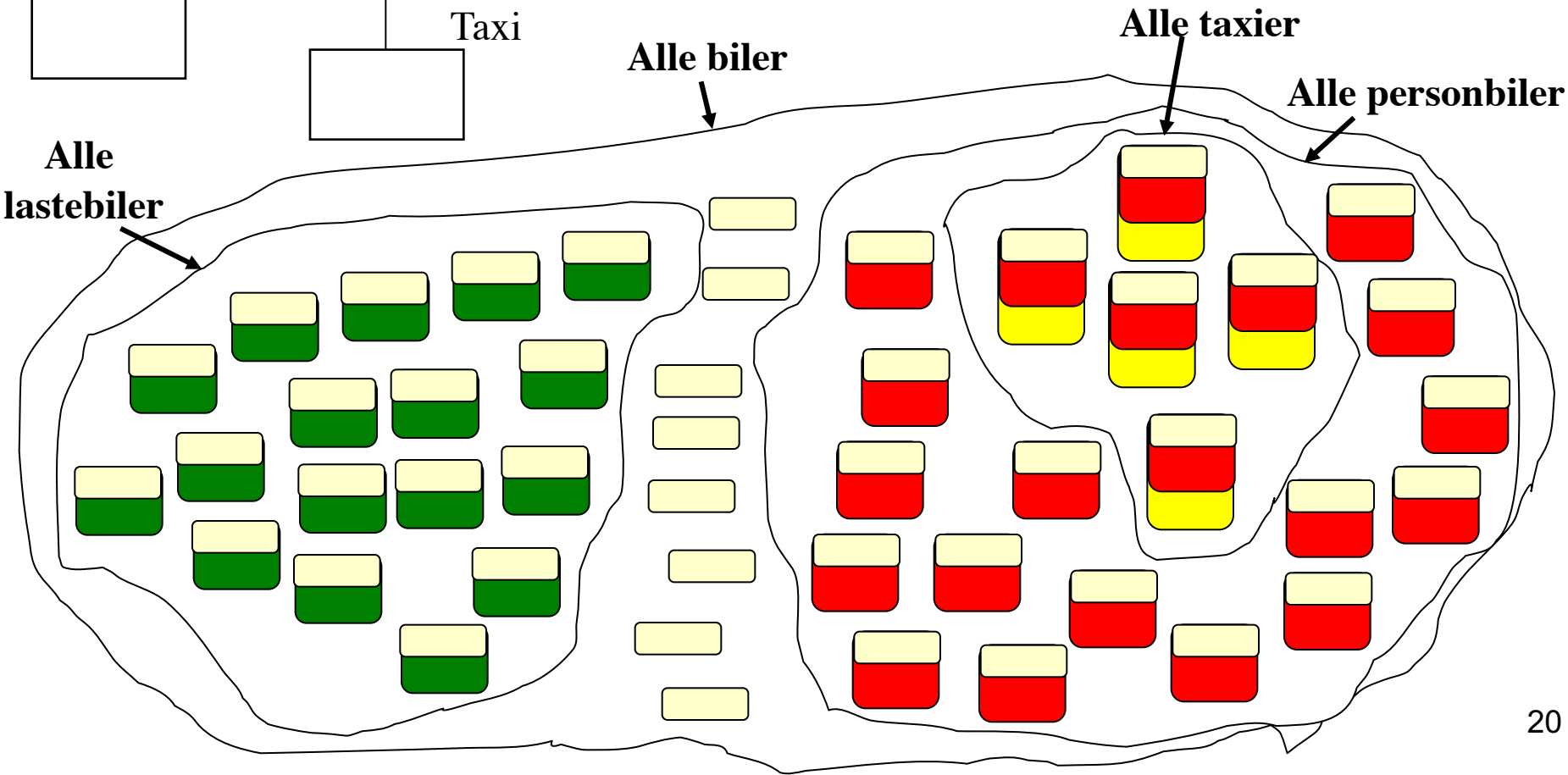


```
class Bil { . . . }

class Personbil extends Bil { . . . }

class Lastebil extends Bil { . . . }

class Taxi extends Personbil { . . . }
```

Klasser - Subklasser




```
class Bil { <lys beige egenskaper>}
class Personbil extends Bil { <røde egenskaper> }
class Lastebil extends Bil { <grønne egenskaper> }
class Taxi extends Personbil { <gule egenskaper> }
```



Hvorfor bruker vi subklasser?

1. Klasser og subklasser avspeiler **virkeligheten**
   - Bra når vi skal modellere virkeligheten i et datasystem
2. Klasser og subklasser avspeiler **arkitekturen** til datasystemet / dataprogrammet
   - Bra når vi skal lage et oversiktlig stort program

3. Klasser og subklasser kan brukes til å forenkle og gjøre programmer mer forståelig, og spare arbeid:
   Gjenbruk av programdeler
   - ”Bottom up” – programmering
     - Lage verktøy
   - ”Top down” programmering
     - Postulere verktøy

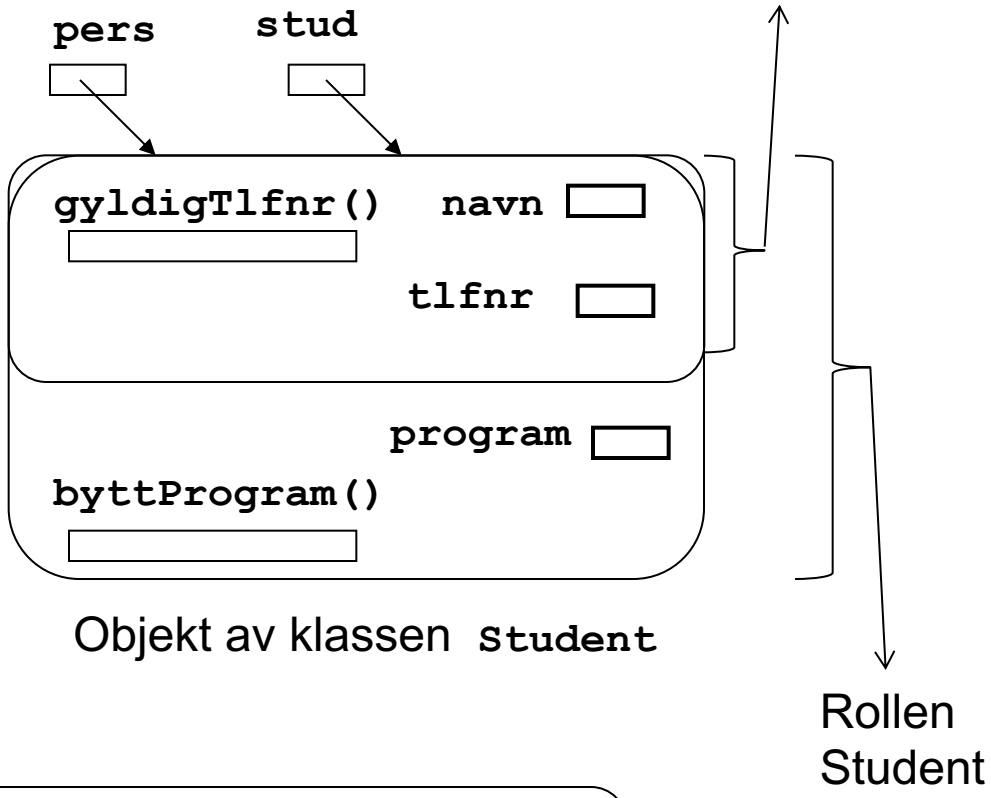
Mer om dette neste uke

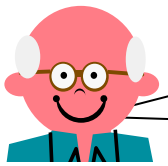
NYTT

Ulike referansetyper (pekertyper)

```
class Person {
   String navn;
   int tlfnr;
   boolean gyldigTlfnr() {...}
}

class Student extends Person {
    String program;
    void byttProgram(String nytt){...}
}
```

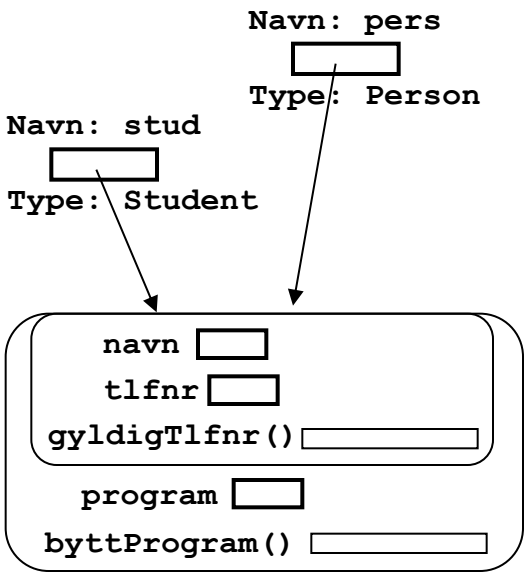
Person pers;
Student stud;





forskjellige referansetyper =
forskjellige **roller** =
forskjellige **briller**

Ulike måter å se et objekt på



Typen (klassen) til hele dette objektet er ***Student***

- Typen (klassen) til et objekt er uforanderlig.
  Et objekt kan likevel ***fremtre*** *for oss* på ulike måter
  Det kan spille forskjellige roller.
- Et objekt av klassen

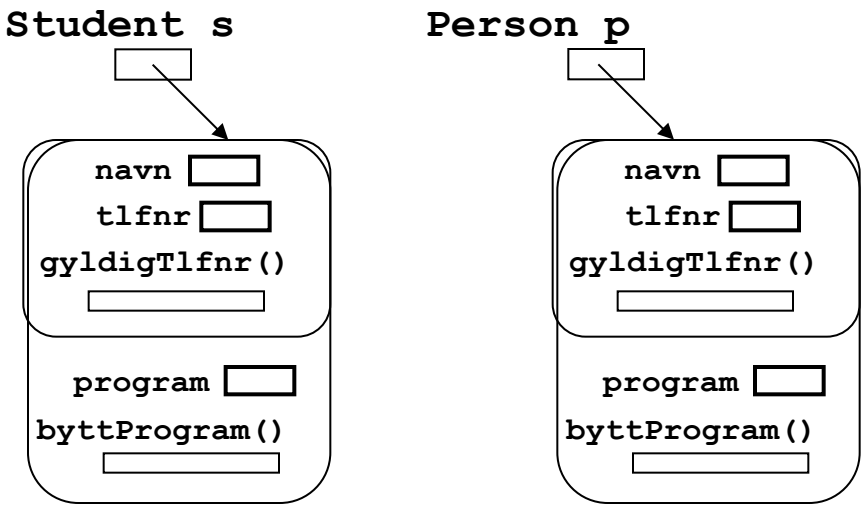
  `class Student extends Person {...}`

  kan vi se på som et objekt av typen (klassen)
  - **`Person`**: da er egenskapene som er spesielle for Student ikke synlige (men de er der fortsatt!).
  - **`Student`**: da er både Person- og Student-egenskapene synlige for oss.
- Det er *referansens (pekerens) type* som avgjør hvordan objektet fremtrer.
  (med untak av ”virtuelle” metoder,
  som vi skal lære om neste uke)

```
Student stud;
Person pers;
stud = new Student();
pers = stud;
```

Eksempler

```
class Person {
   String navn;
   int tlfnr;
   boolean gyldigTlfnr() {...}
}

class Student extends Person {
  String program;
  void byttProgram(String nytt) {...}
}
```



Anta:

Student s = new Student();
Person p = new Student();

Hvilke av følgende uttrykk er nå lovlige?

s.navn = ”Ole-Morten";
. . . s.gyldigTlfnr();
s.program = "Matte";
s.byttProgram("Data");

p.navn = ”Ole-Ivar";
. . . p.gyldigTlfnr();
~~**p.program = "Matte";**~~
~~**p.byttProgram("Data");**~~

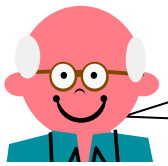
Private og public i subklasser

Private i en klasse gjør at ingen subklasser kan se denne egenskapen

Protected i en klasse gjør at alle subklasser kan se denne egenskapen
Men ingen utenfor klassen (bortsett fra i samme katalog/pakke)

Public er som før

```
class Person {
        protected String navn;
        protected int tlfnr;

        public boolean gyldigTlfnr() {
                return tlfnr >= 10000000 && tlfnr <= 99999999;
        }
}
```



Nytt reservert ord i Java:
protected

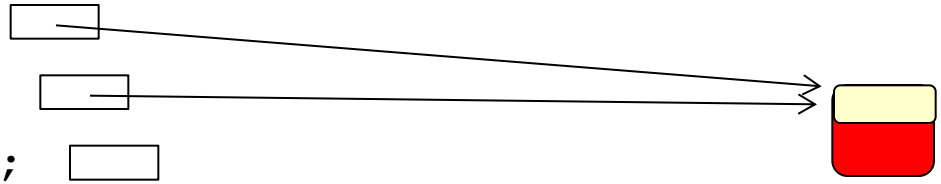
*Student og Ansatt med **protected***

```
class Student extends Person {
        protected String program;

        public void byttProgram(String nytt) {
                program = nytt;
        }
}

class Ansatt extends Person {
        protected int lønnstrinn;
        protected int antallTimer;

        public void lønnstillegg(int tillegg) {
                lønnstrinn += tillegg;
        }
}
```

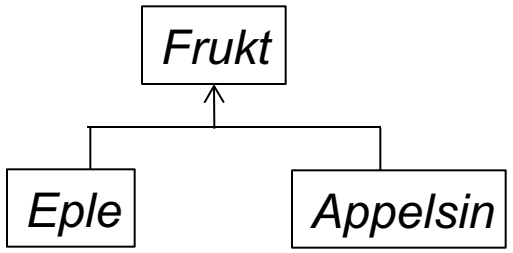
Om det hadde stått “private antallTimer”, så ville ingen subklasser til Ansatt kunne se denne egenskapen

Tilordning av referanser

```
class LagFrukt {
    public static void main(String[] args) {
       Frukt f;
       Eple e;
       Appelsin a;
       e = new Eple();
       f = e;
       a = f;  // ???
     }
}
```



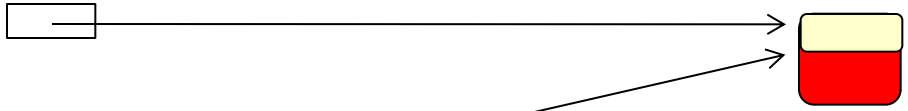
```
class Frukt { .. }
class Eple extends Frukt { .. }
class Appelsin extends Frukt { .. }
```



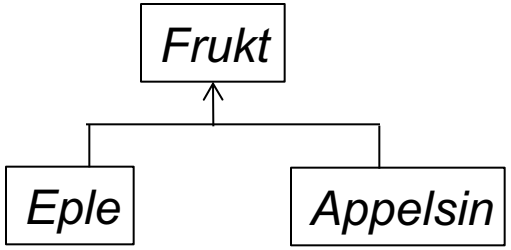
Hva slags objekt er dette?

Den boolske operatoren **instanceof** hjelper oss å finne ut av hvilken klasse et gitt objekt er, noe som er nyttig i mange tilfeller:

```
class TestFrukt {
  public static void main(String[] args) {
     Eple e = new Eple();
     skrivUt(e);
  }
  static void skrivUt(Frukt f) {
     if (f instanceof Eple)
        System.out.println("Dette er et eple!");
     else if (f instanceof Appelsin)
        System.out.println("Dette er en appelsin!");
  }
}
```



```
class Frukt { .. }
class Eple extends Frukt { .. }
class Appelsin extends Frukt { .. }
```



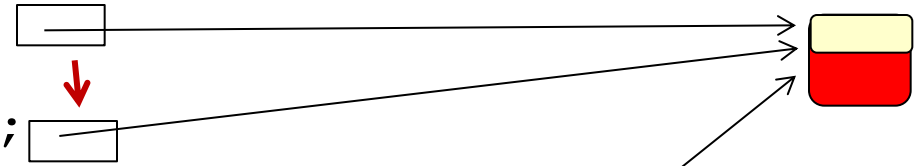
Konvertering av referanser

- Anta at vi har:

```
class Student extends Person {…}
Student stud = new Student();
```

- Ved tilordningen

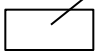
```
Person pers;
pers = stud;
```



har vi en implisitt konvertering fra Student- til Person-referanse.

- Hvis vi nå ønsker å få tak i de spesielle Student-egenskapene, må vi foreta en eksplisitt konvertering tilbake til Student igjen:

```
Student stud2 = (Student) pers;
```



Dette kalles ”casting” (class-cast på engelsk), typekonvertering på norsk. Medfører kjøretidstest.

Konvertering av referanser (forts.)

- Hva hvis vi istedenfor hadde hatt:

```
Person pers = new Person();
Student stud = (Student) pers;
```

- Dette godkjennes av kompilatoren, men ved kjøring går det galt, og vi får feilmeldingen

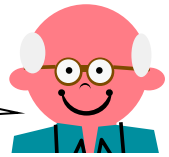
```
java.lang.ClassCastException
```

(fordi pers ikke peker på et objekt med alle "Student" egenskapene)

- For å unngå denne feilen, bør **instanceof** brukes:

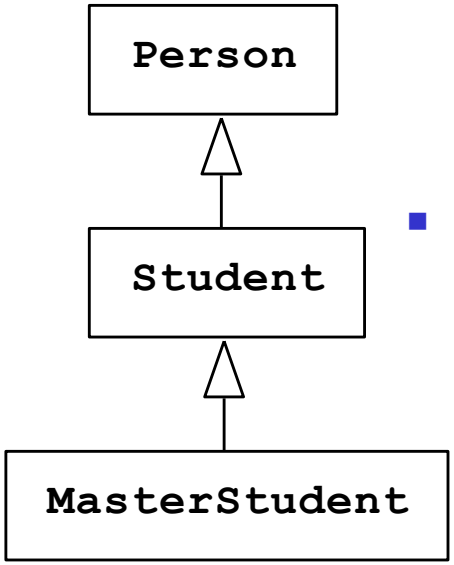
```
if (pers instanceof Student) {
    Student stud = (Student) pers;
}
```

Har objektet som pers peker på alle "Student"-egenskapene ?



Konvertering mellom flere nivåer

```
MasterStudent master = new MasterStudent();
```



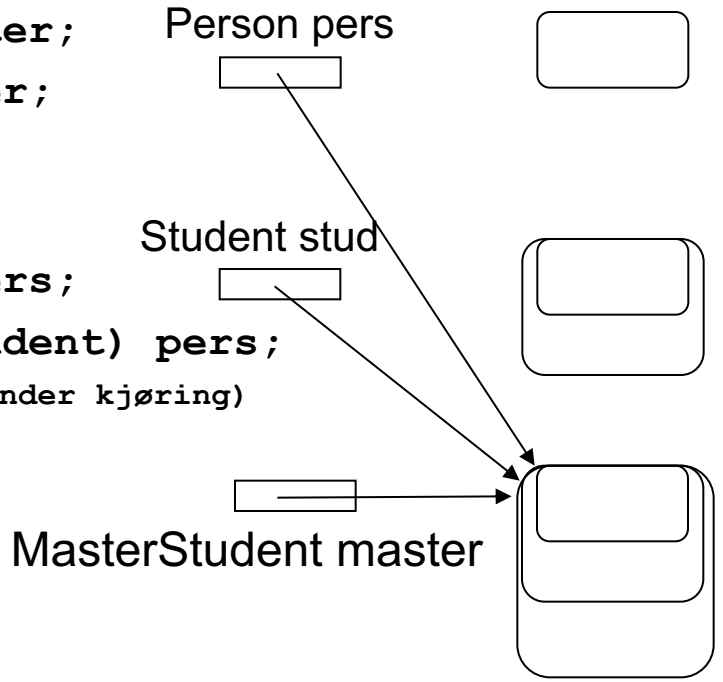
- **Konvertering oppover:**

```
Student stud = master;
Person pers = master;
```

- **Konvertering nedover:**

```
stud = (Student) pers;
master = (MasterStudent) pers;
```

(fordi dette krever kontroll under kjøring)



Regel: ” Alle referanser har lov til å peke bortover og nedover”

(men ikke ”oppover”)

Klassen Object

- class Object {. . . } er alle klassers mor (alle klassers superklasse)
- D.v.s. at alle klasser i Java er subklasser av klassen Object. Når vi skriver

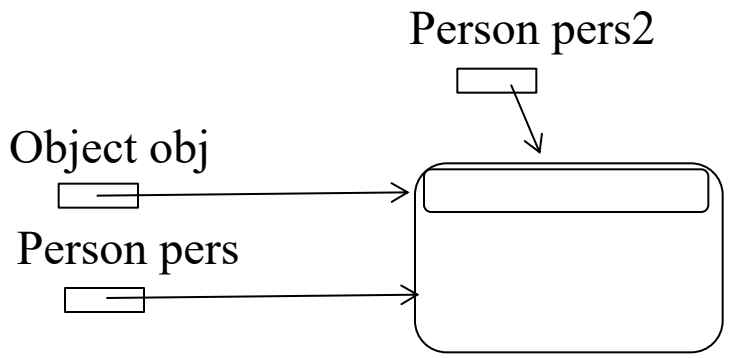
```
class Person { … }
```

så tolker Java dette som

```
class Person extends Object { … }
```

- Dermed kan en referanse av typen Object peke på et hvilket som helst objekt:

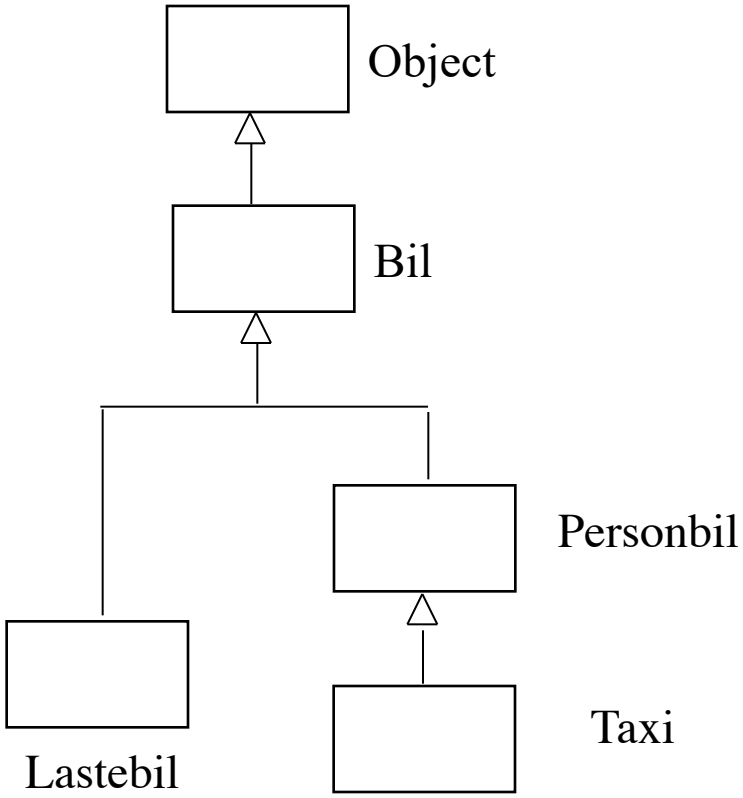
```
Person pers = new Person();
    Object obj = pers;
Person pers2 = (Person)obj;
```

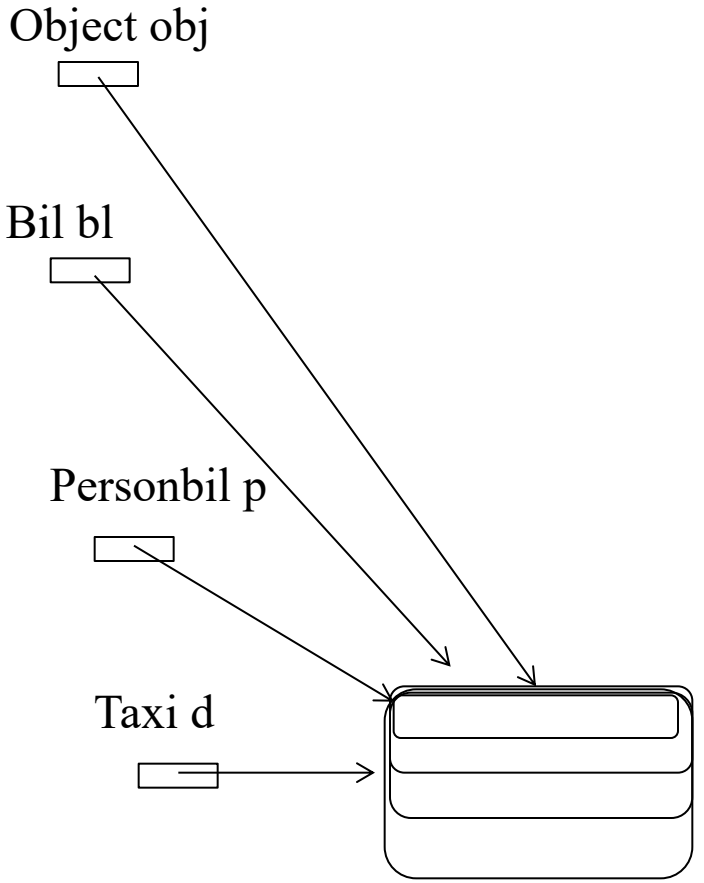


Mer neste gang om hva som er inne i Object-delen av et objekt.

class Object - eksempel

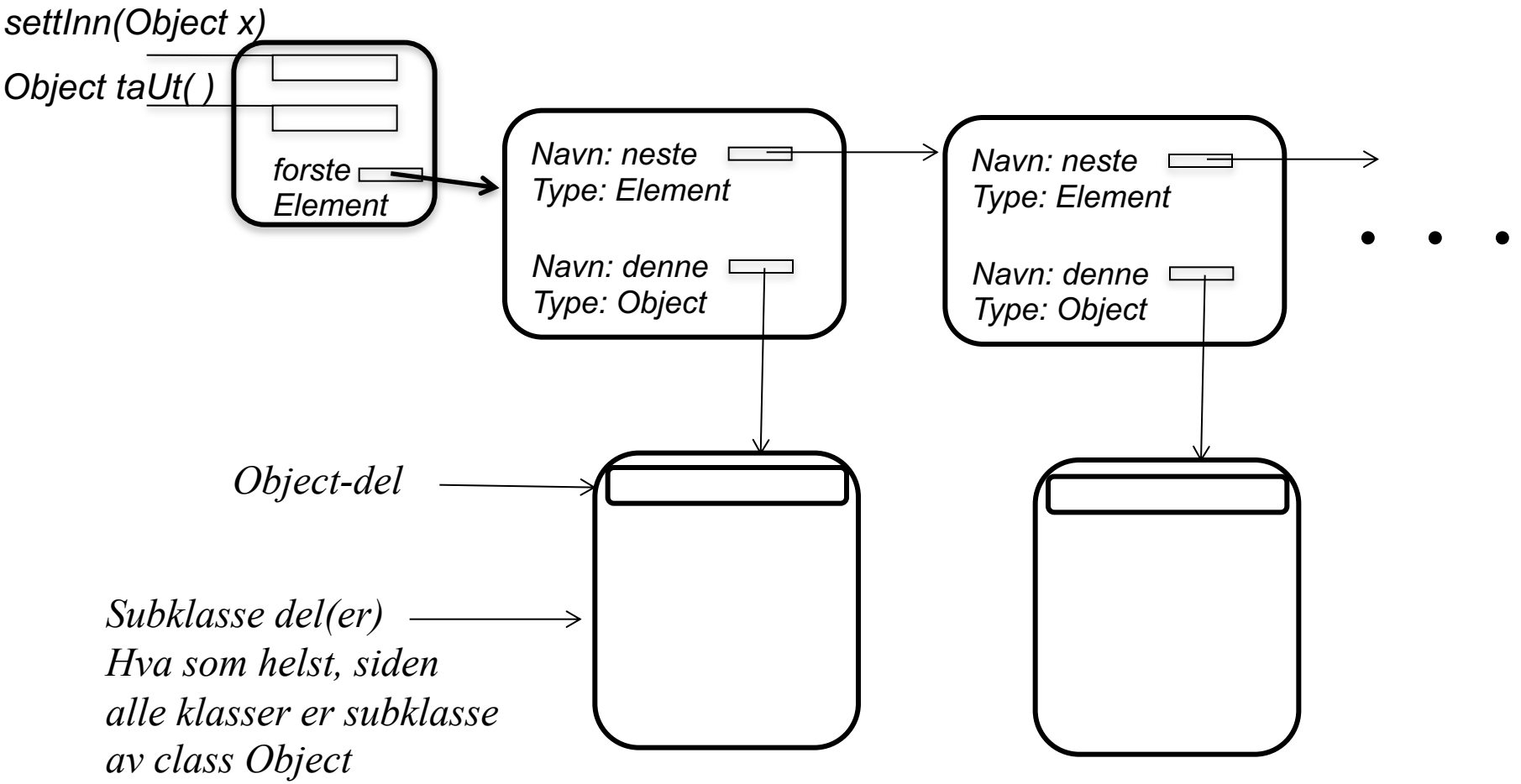
Klassehierarki:





En lenket liste med noder

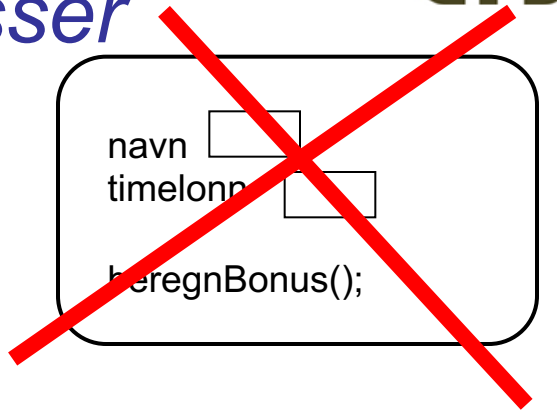
Dette venter vi noen uker med

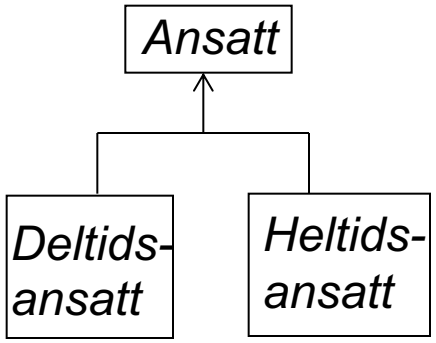


Abstrakte metoder og klasser

```
abstract class Ansatt  {
    protected String navn;
    protected double timelonn;

    public abstract double beregnBonus();
}
```

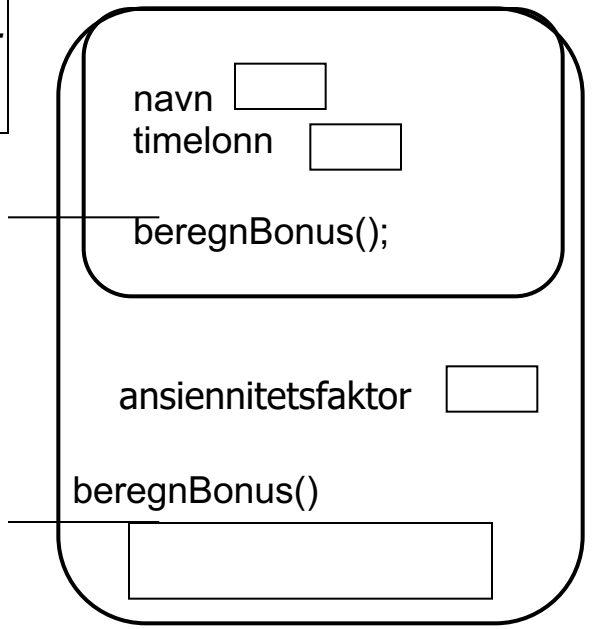
Ikke lov å si new Ansatt() !






```
class Deltidsansatt extends Ansatt {
    public double beregnBonus() {
          return 0;
    }
}
```

Heltidsansatt objekt




```
class Heltidsansatt extends Ansatt {
    protected int ansiennitetsfaktor;

    public double beregnBonus() {
            return timelonn* ansiennitetsfaktor;
    }
}
```