

IN1010 våren 2023

Tirsdag 7. februar

Arv og subklasser – del 1

Stein Gjessing.



Agenda

- Subklasser
 - Objekter av subklasser
- Referanser til objekter av subklasser
 - Casting / typekonvertering. (sterk typing)
 - “instanceof”
- class Object
- Abstrakte klasser



Ukens tema: Arv og subklasser I

- Objekter bruker vi til å modellere den virkelige verden inne i datamaskinen.
 - Pluss strukturere selve programmet vår (programvarearkitektur)
- Å programmere vil si å strukturere og arrangere begreper og lage ryddige, oversiktlige og utvidbare komponenter, moduler og modeller
- Til dette bruker vi bl.a. Objekter og klasser med arv og subklasser
- Noen problemer som kan løses med subklasser kan også løses (bedre ?) på andre måter
 - Bla. a. ved hjelp av delegering (minner om komposisjon)
 - Mer om dette senere
 - Men uansett må dere være eksperter på subklasser

Kortversjon: Definisjon av subklasser

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

```
class Klasse { . . . }
```

```
class Subklasse extends Klasse { . . . }
```



Nytt Java nøkkelord:
extends

Navn på klasser og subklasser (Klasse og Subklasse 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



En lite side-kommentar

- Når vi skal lage et program må vi velge hva og hvilke deler av den virkelige verden vi skal modellere inne i datamaskinen
- Vi tar bare med det vi trenger
- Vi modellerer bare de aspektene av den virkelige verden som vi trenger for å løse programmets oppgave
- Dette kaller vi gjerne for vårt *perspektiv* på systemet som vi modellerer
- Kristen Nygaard:
 - Et perspektiv er ikke nøytralt (etikk og informatikk)

Modeller av båter





Modeller av fly



Hva er hensikten med modellen?
Hvor nøyaktig skal vi modellere?

**Stein, du må
huske på å
 snakke om
«Digitale tvillinger»
 senere**



Tilbake til vårt system: Universitetsregisteret

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

Kristen
Nygaard



Hva er vårt perspektiv her?

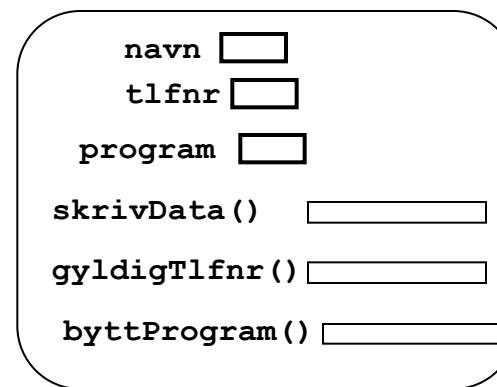
Klassen Student

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

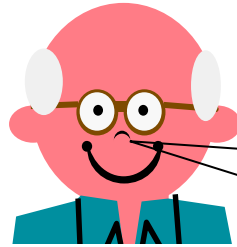
    void skrivData() {
        System.out.println("Navn: " + navn);
        System.out.println("Telefon: " + tlfnr);
        System.out.println("Studieprogram: " + program);
    }

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

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



objekt av klassen Student



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



**OK –
Vi kommer tilbake til det på lysark 48

Og det er jo slik at om det ikke står noe,
så er egenskapene synlige i hele
fil-katalogen (package private)**

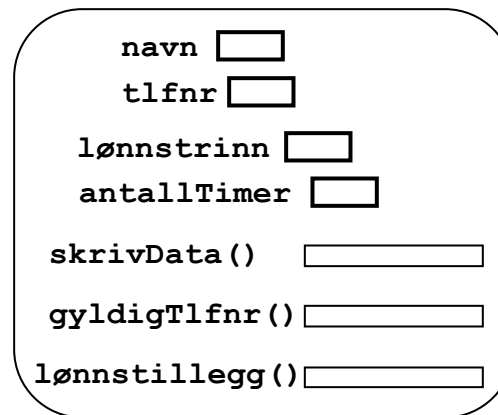
Klassen Ansatt

```
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;
    }

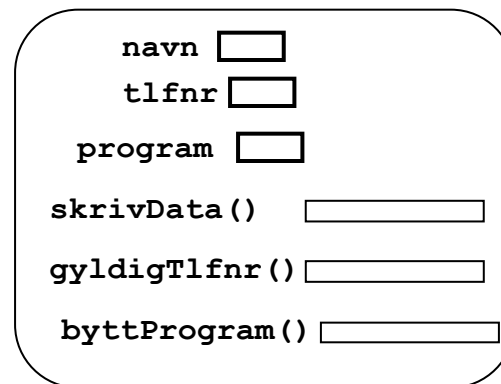
    void lønnstillegg(int tillegg) {
        lønnstrinn += tillegg;
    }
}
```



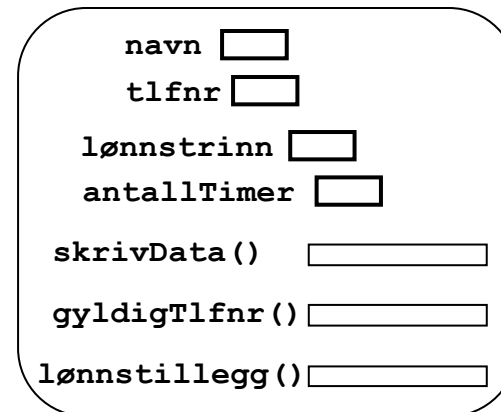
objekt av klassen Ansatt

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;  
    }  
}
```

navn	<input type="text"/>
tlfnr	<input type="text"/>
gyldigTlfnr()	<input type="text"/>

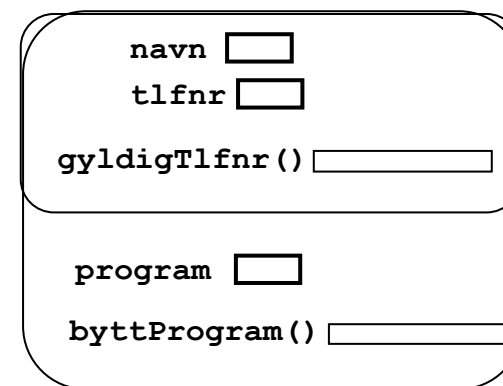
Klassen Person beskriver alt som er felles for studenter og ansatte



Og det er kanskje ikke helt unaturlig

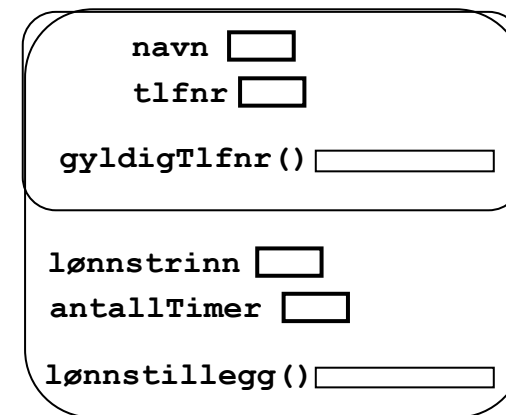
Student og Ansatt som subklasser av Person

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



Student-objekt

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



Ansatt-objekt

Student og Ansatt som subklasser av Person

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

extends

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

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

Nytt Java nøkkelord:

extends



Hva med skrivData()?

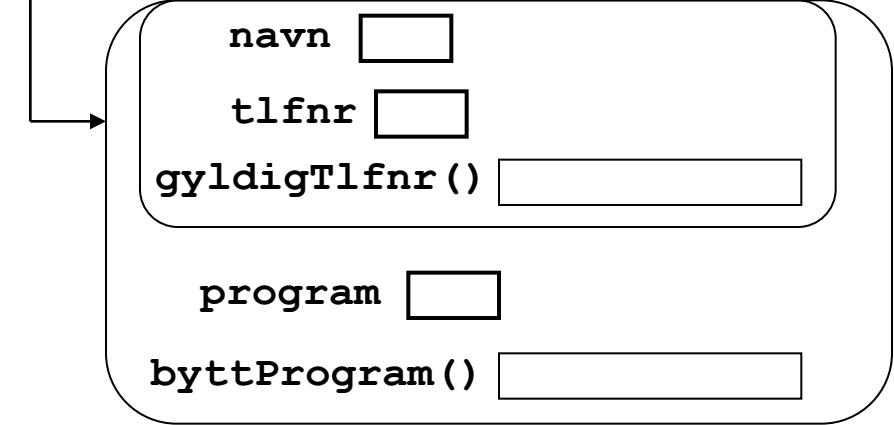
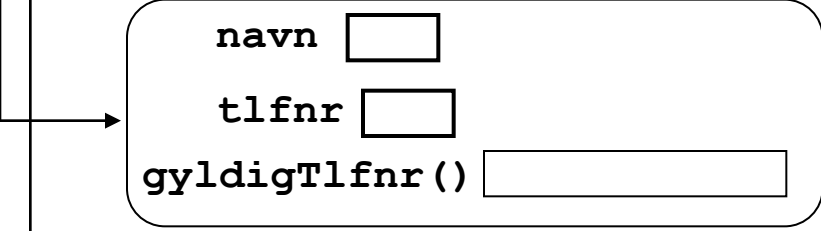
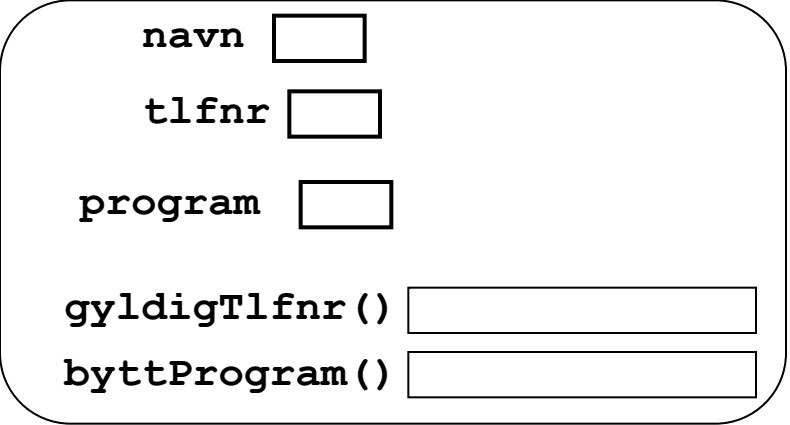
- Kommer tilbake til denne...

```
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:

Før:

eller med bruk av subklasser (nå):

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

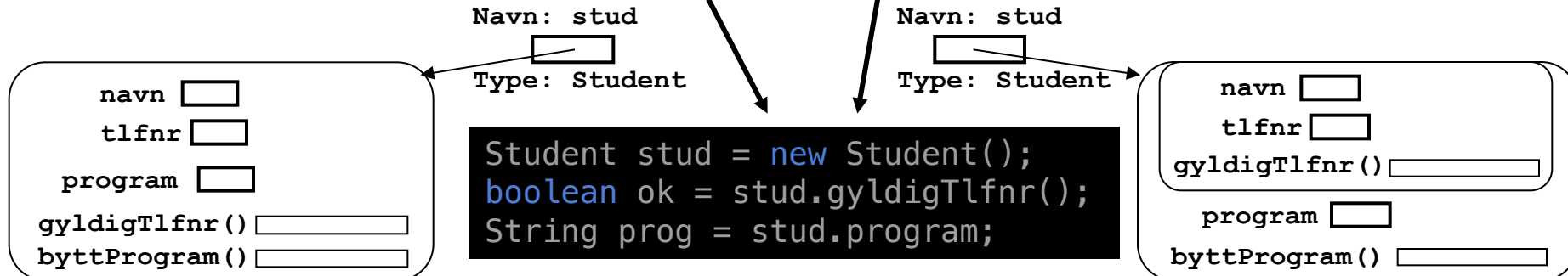
    boolean gyldigTlfnr() { . . . }
    void byttProgram(String nytt) { . . . }
}
```

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

    boolean gyldigTlfnr() { . . . }
}
```

```
class Student extends Person {
    String program;

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

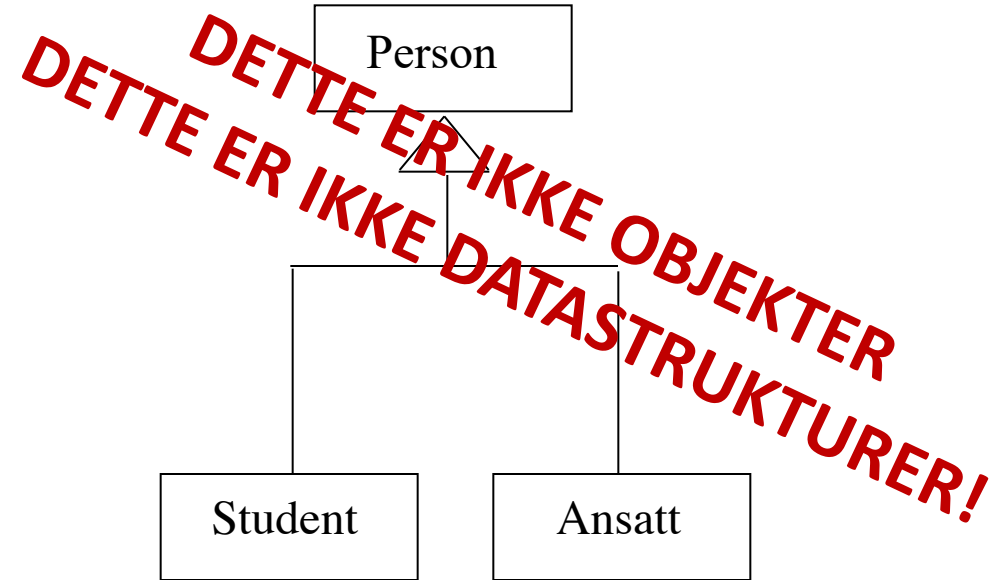


Tegning av subklassehierarki

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

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

```
class Ansatt extends Person {  
    int lønnstrinn;  
    int antallTimer;  
  
    void lønnstillegg(int tillegg) { . . . }  
}
```



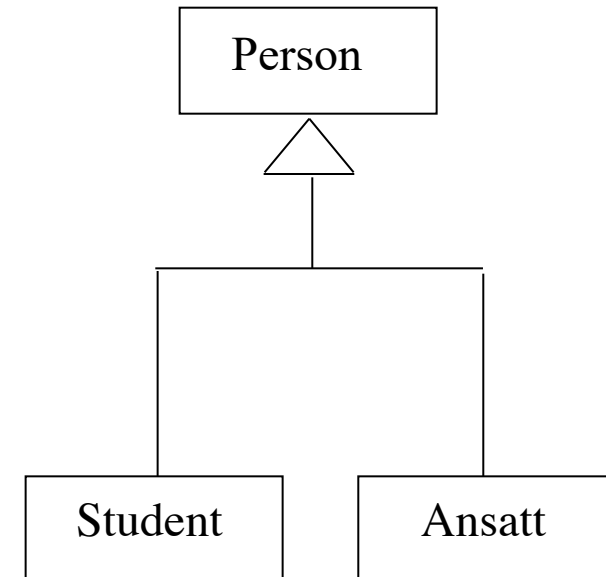
Det er en tegning av strukturen i problemet
Det er en tegning av strukturen i programmet

Notasjon for subklassehierarki

```
class Person {  
}
```

```
class Student extends Person {  
}
```

```
class Ansatt extends Person {  
}
```



En beskrivelse av programmet / problemet

Igjen: Hva er en subklasse?

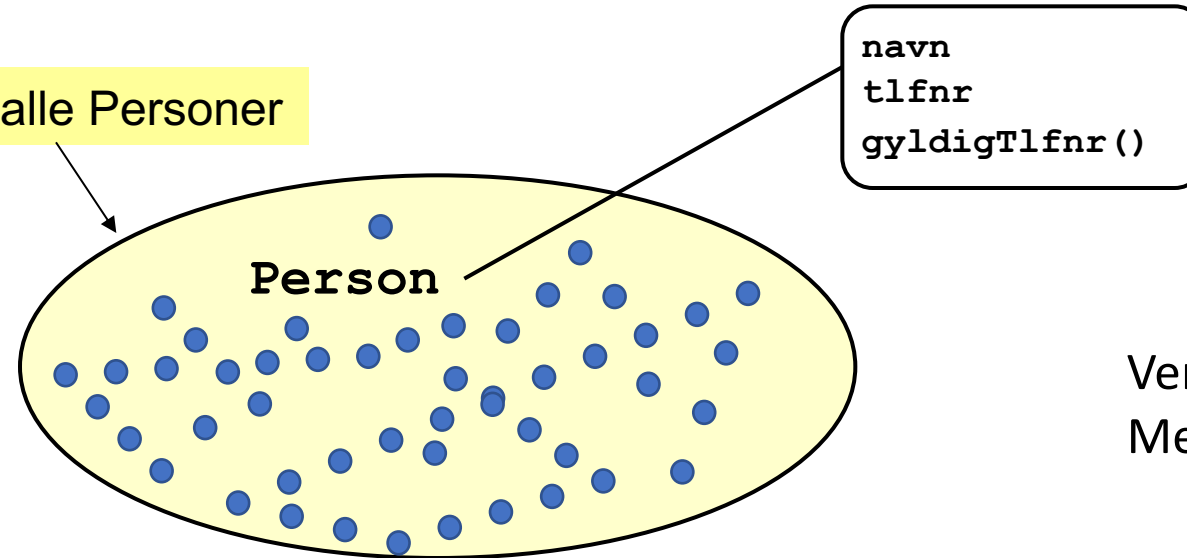
- En subklasse er en klasse som bygger på en allerede spesifisert klasse, og som dermed **arver** dennes egenskaper i tillegg til å utvide med egne egenskaper (metoder/variable/konstanter).
- En subklasse er altså en mer **spesialisert** utgave av klassen den bygger på.
- Klassen vi bygger på kalles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



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

Alle Personene i programmet vårt

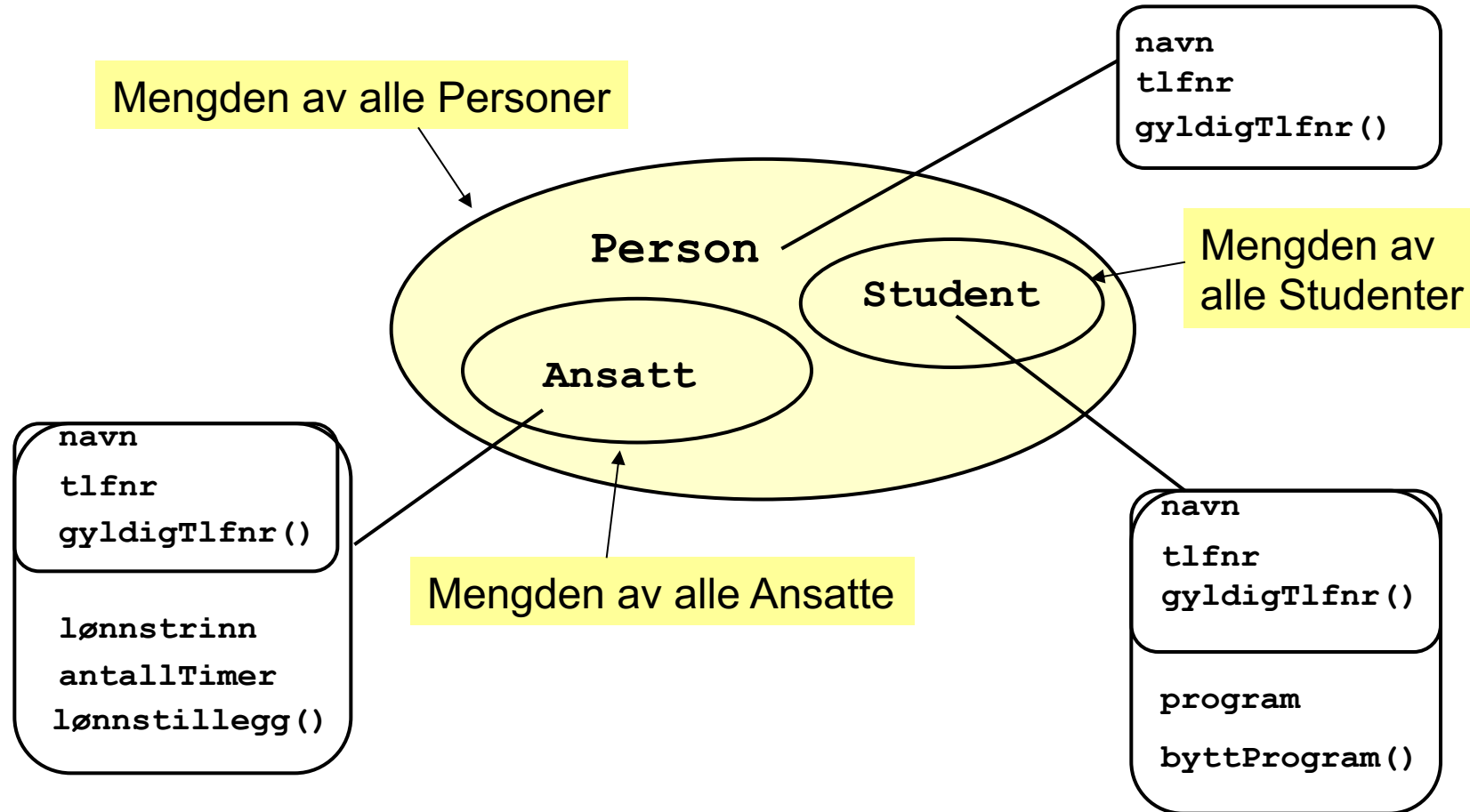
Mengden av alle Personer



Venn-diagram
Mengde-diagram

- Alle de blå prikkene er alle personer vi modellerer (i vårt system, i "vår verden")
- Alle de blå prikkene er alle personer som er med i programmet vårt
- Alle de blå prikkene er alle Person-objektene i programmet vårt
- Alle disse blå prikkene har de samme egenskapene og de er definert av **klassen Person**

Spesialisering - Generalisering



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

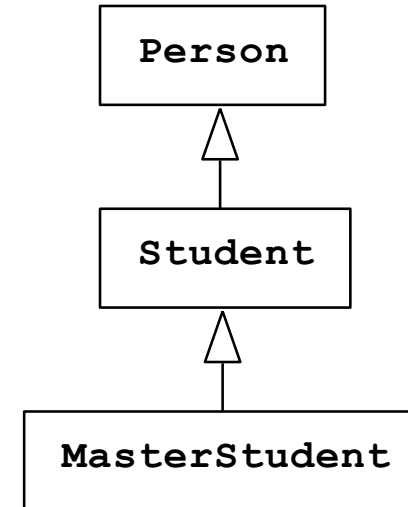
Klasse-hierarkier

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

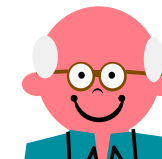
```
class Person {  
  
}
```

```
class Student extends Person {  
  
}
```

```
class MasterStudent extends Student {  
  
}
```



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

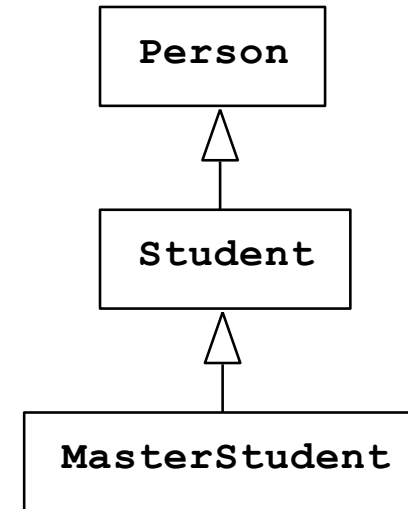


**NB! KLASSE-HIERARKI,
IKKE
DATASTRUKTUR !**

```
class Person {  
  
}
```

```
class Student extends Person {  
  
}
```

```
class MasterStudent extends Student {  
  
}
```



- **Først modellerer vi virkeligheten og lager et klassehierarki.**
- Deretter lager vi programmet.
- Programmets subklassestruktur og klassehierarkiet uttrykker det samme.
- **Når programmet utføres er tegningen av klassehierarkiet glemt** (men det avspeiles jo av programmet)

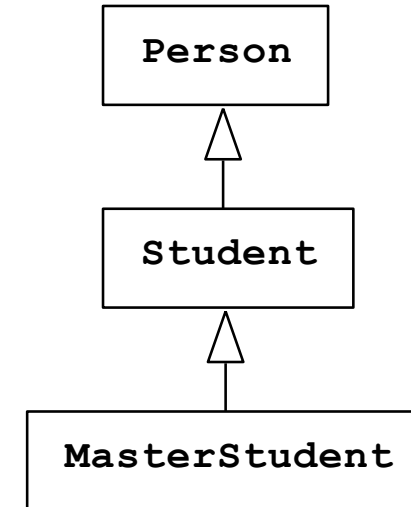
Klasse-hierarkier

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

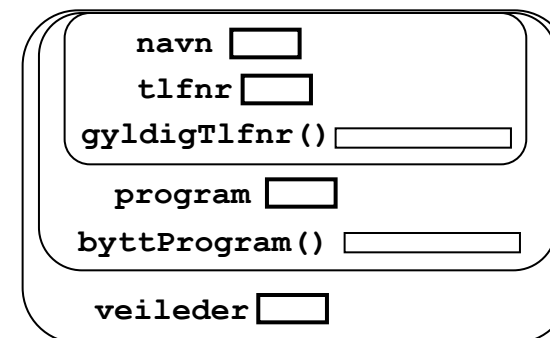
```
class Person {  
    String navn;  
    int tlfnr;  
    boolean gyldigTlfnr() { . . . }  
}
```

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

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



MasterStudent-objekt

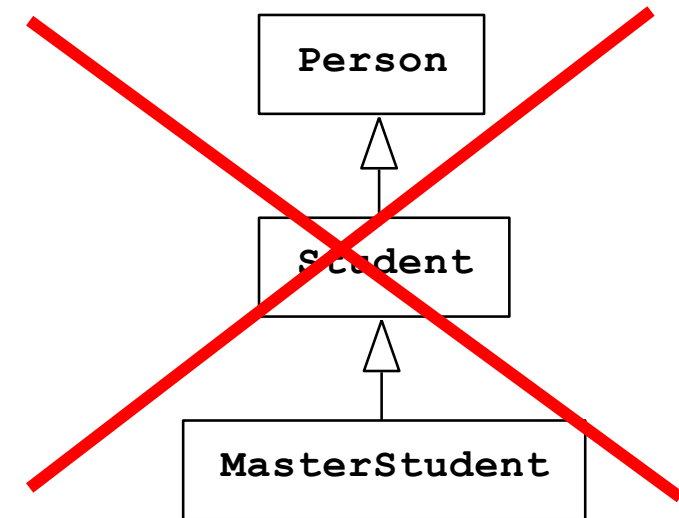


Når programmet utføres lages det datastrukturer / objekter

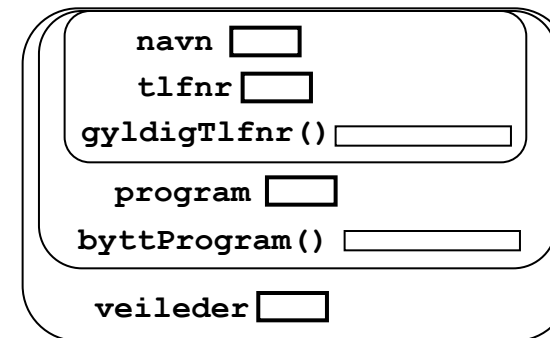
```
class Person {  
    String navn;  
    int tlfnr;  
    boolean gyldigTlfnr() { . . . }  
}
```

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

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

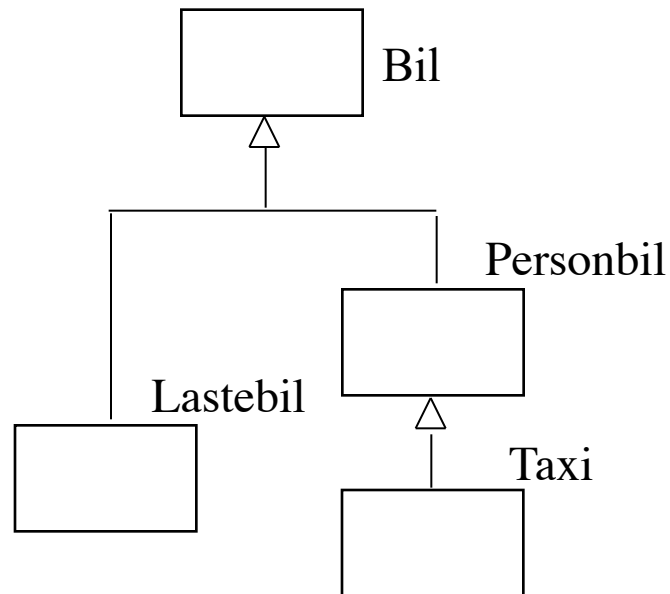


MasterStudent-objekt



Klasser - Subklasser

Klassehierarki:



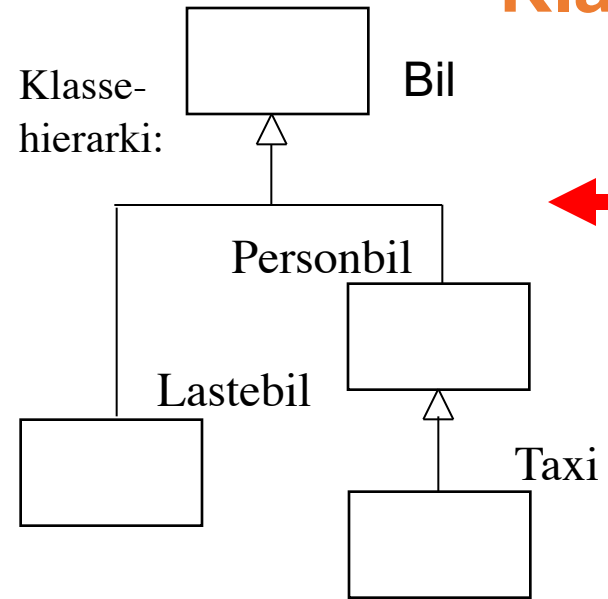
```
class Bil { ... }
```

```
class Personbil extends Bil { ... }
```

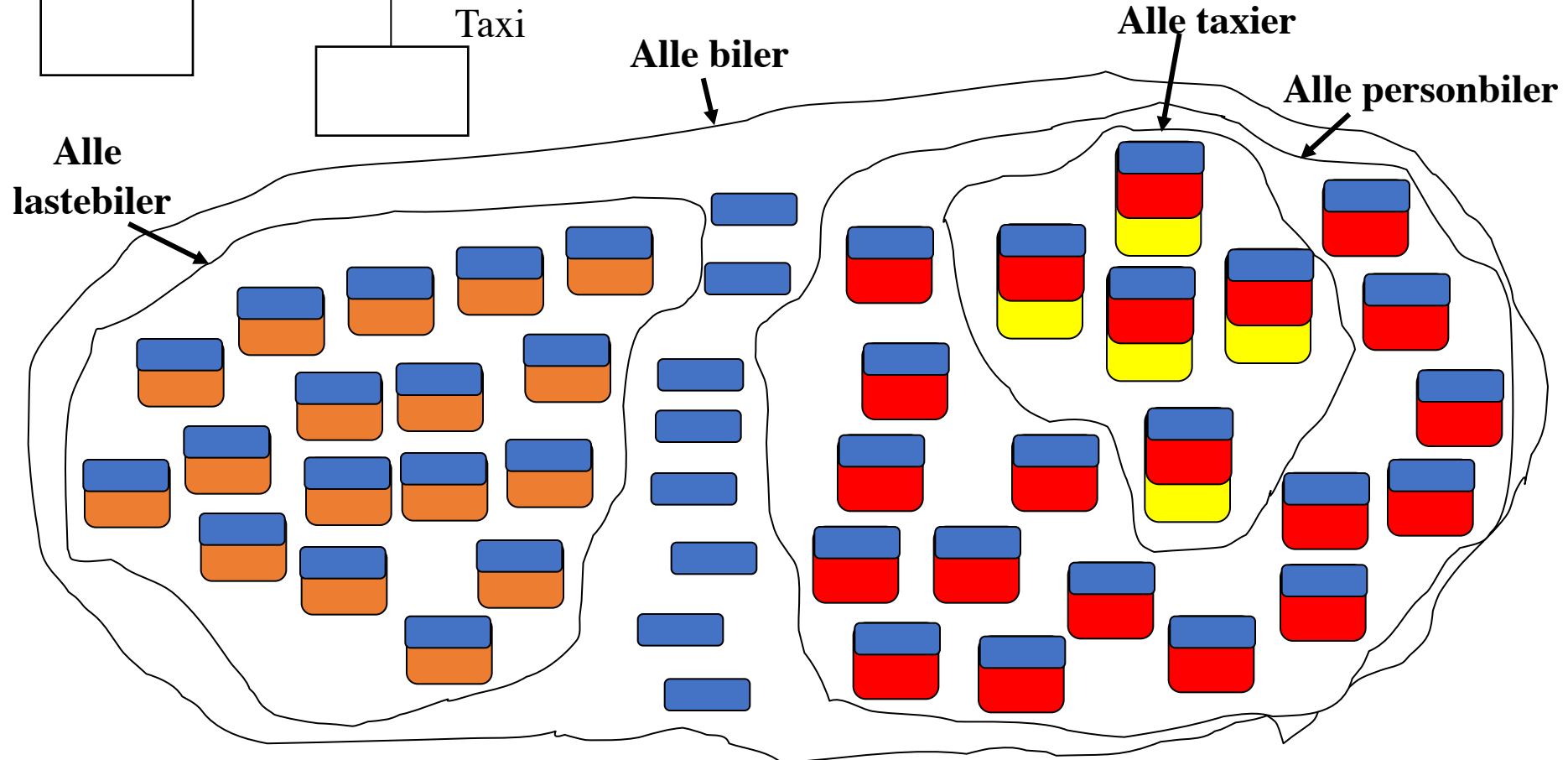
```
class Lastebil extends Bil { ... }
```

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

Klasser - Subklasser



```
class Bil { <blå egenskaper> }
class Personbil extends Bil { <røde egenskaper> }
class Lastebil extends Bil { <brune 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
 - Se f.eks. Javas API
 - 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

1 og 2 er klart viktigst



Mer om dette neste uke



Referanser og subklasser

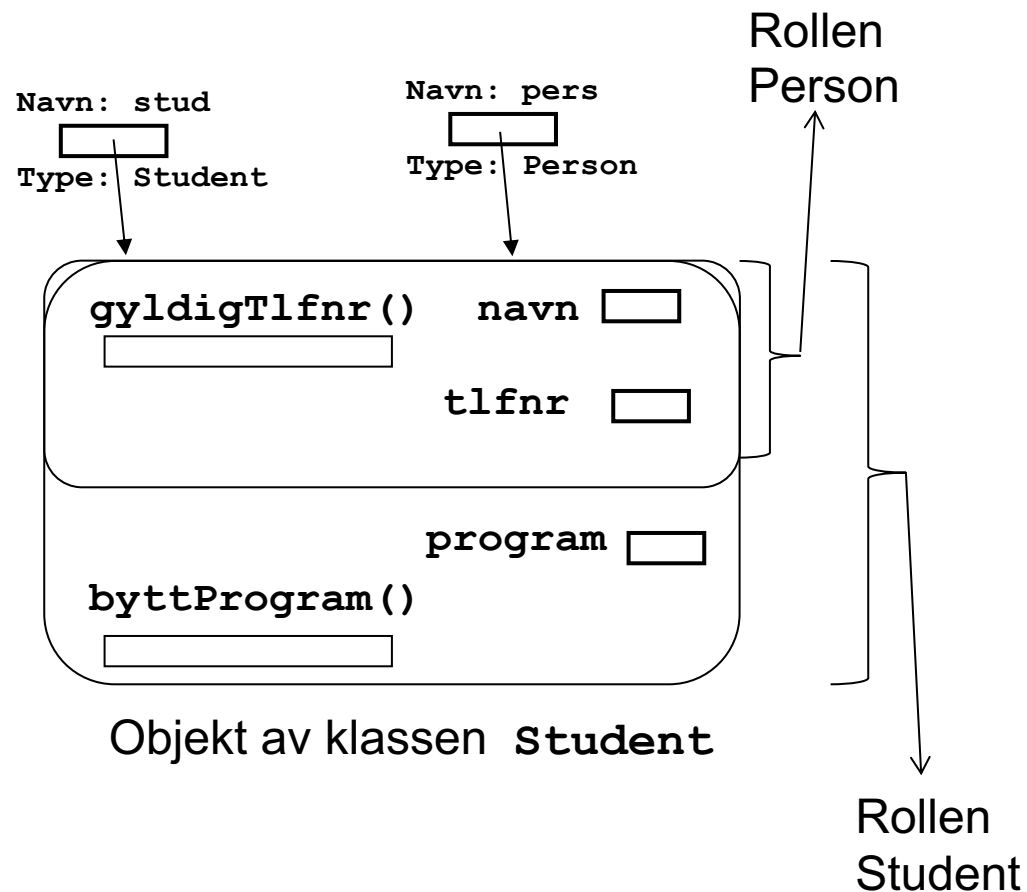
- Nå vet vi litt om subklasser
- Vi vet bl.a. at objekter av subklasser inneholder flere “deler”
- Java er et sterkt typet språk!
 - Hva har det å si for referanser til slike oppdelte subklasse-objekter?

Ulike referansetyper (pekertyper)

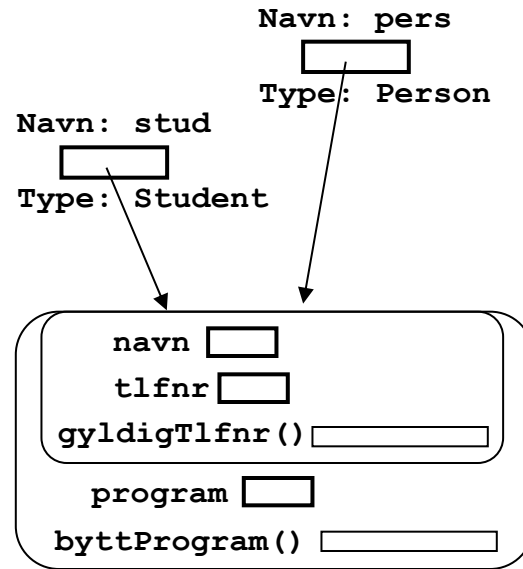
```
Student stud;  
Person pers;  
stud = new Student();  
pers = stud;
```

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

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



Ulike måter å se et objekt på



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

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

- **Typen (klassen) til selve objektet er uforanderlig.** Et objekt kan likevel *fremtre for oss* på ulike måter Det kan spille forskjellige roller.
 - Når et objekt først er laget kan det aldri utvides eller krympes senere
- 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.

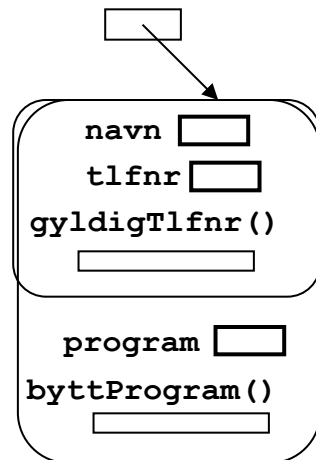
(med unntak av polymorfe metoder, som vi skal lære om neste uke)

Flere eksempler

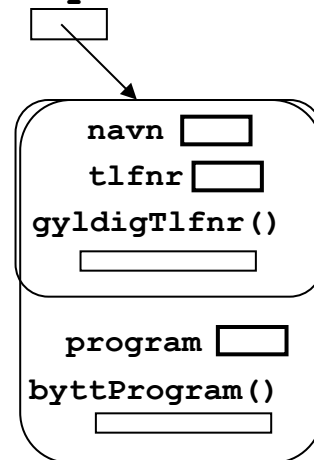
```
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() { . . . }
}
```

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

Student s



Person p



```
// 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("Informatikk");
```

```
p.navn = "Ole-Ivar";
. . . p.gyldigTlfnr();
p.program = "Matte";
p.byttProgram("Informtaikk");
```

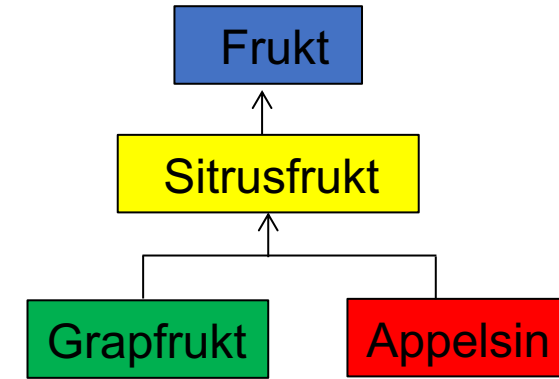


forskjellige referansetyper =
forskjellige **roller** =
forskjellige **briller**



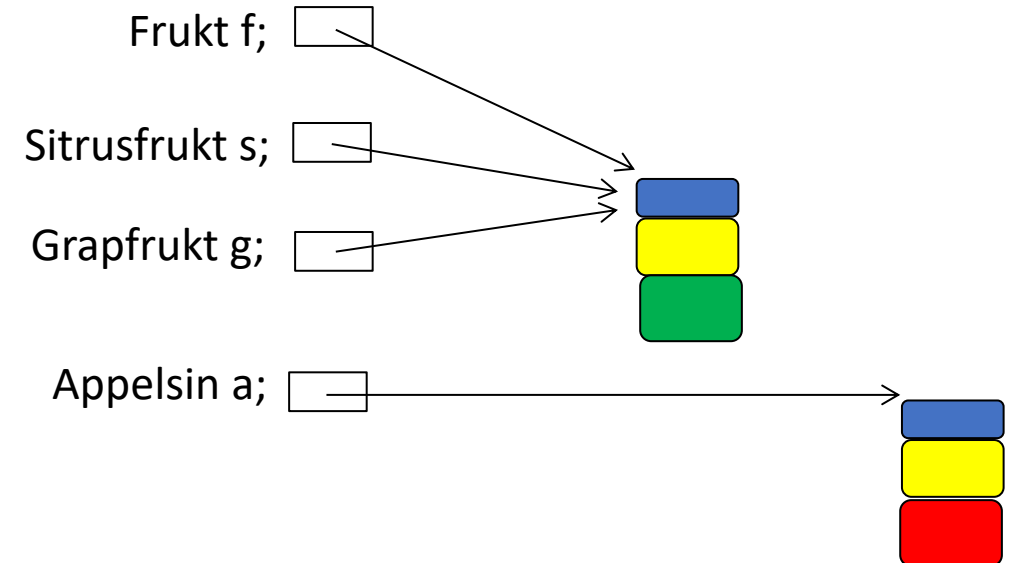
Regel om referanser – Sterk typing

```
class Frukt { .. }  
class Sitrusfrukt extends Frukt { .. }  
class Grapefrukt extends Sitrusfrukt { .. }  
class Appelsin extends Sitrusfrukt { .. }
```



Klassehierarki

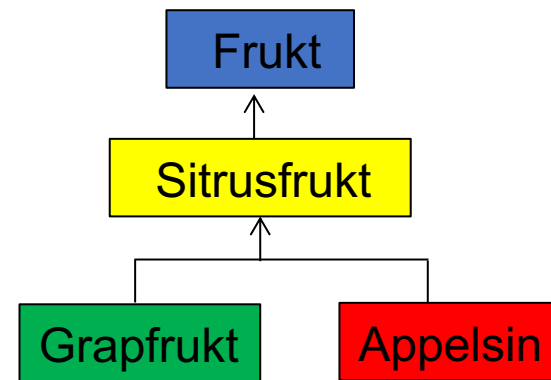
En referanse av typen P har bare lov å peke på objekter som har P-egenskaper



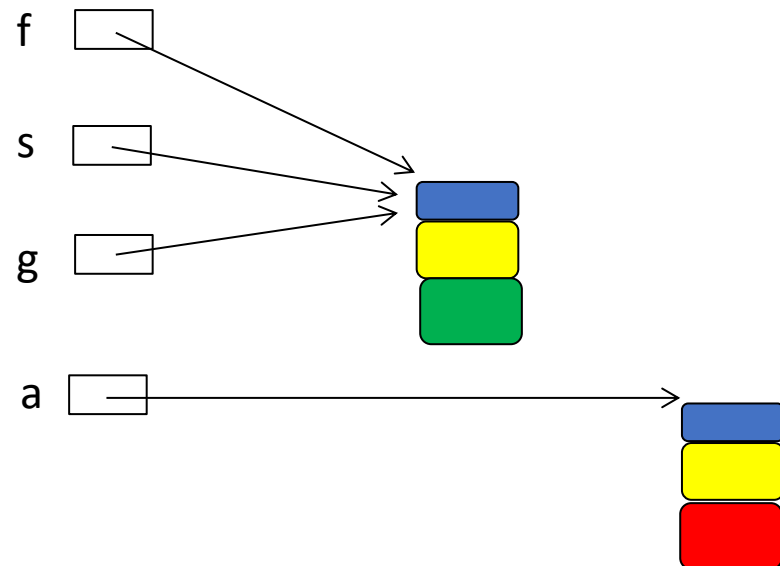
Quiz om referanser til subklasseobjekter

```
class Frukt { .. }  
class Sitrusfrukt extends Frukt { .. }  
class Grapefrukt extends Sitrusfrukt { .. }  
class Appelsin extends Sitrusfrukt { .. }
```

```
Frukt f;  
Sitrusfrukt s;  
Grapefrukt g;  
Appelsin a;  
// Hva er lov  
g = new Grapefrukt();  
f = g;  
s = g;  
a = new Appelsin();
```



Klassehierarki





UNIVERSITETET
I OSLO

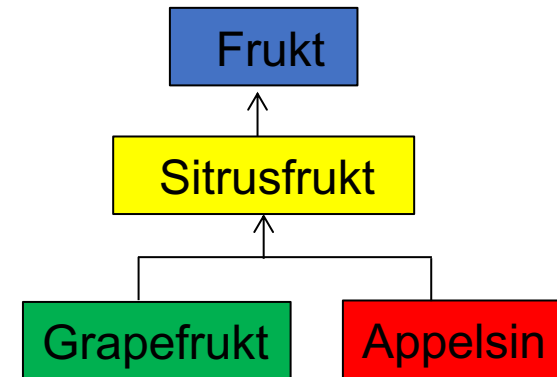


Institutt for informatikk

Menti-quiz

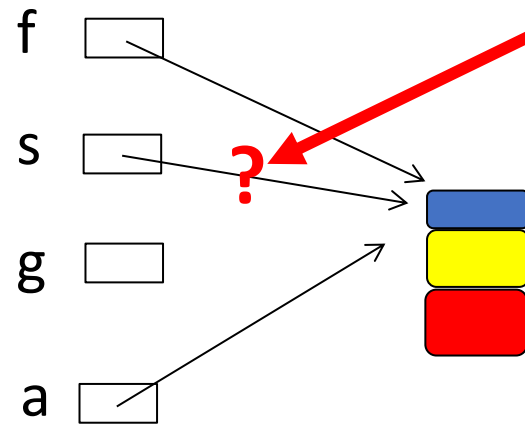
Tilbake til svaret på oppgave 3

```
Frukt f;  
Sitrusfrukt s;  
Grapefrukt g;  
Appelsin a;  
// Hva er lov:  
a = new Appelsin();  
f = a;  
g = f;  
s = f;
```



Klassehierarki

Men dette kan vi rette på



s = f; // Ikke lov

Men f peker jo på et objekt som inneholder Sitrusfrukt-egenskaper, så dette burde jo være i orden?

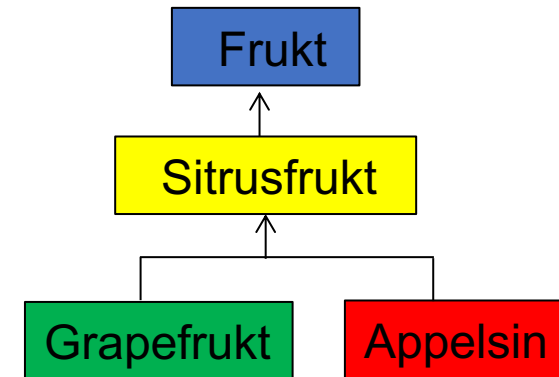
Men det vet ikke kompilatoren !!!

Løsning: La kjøretidsystemet sjekke

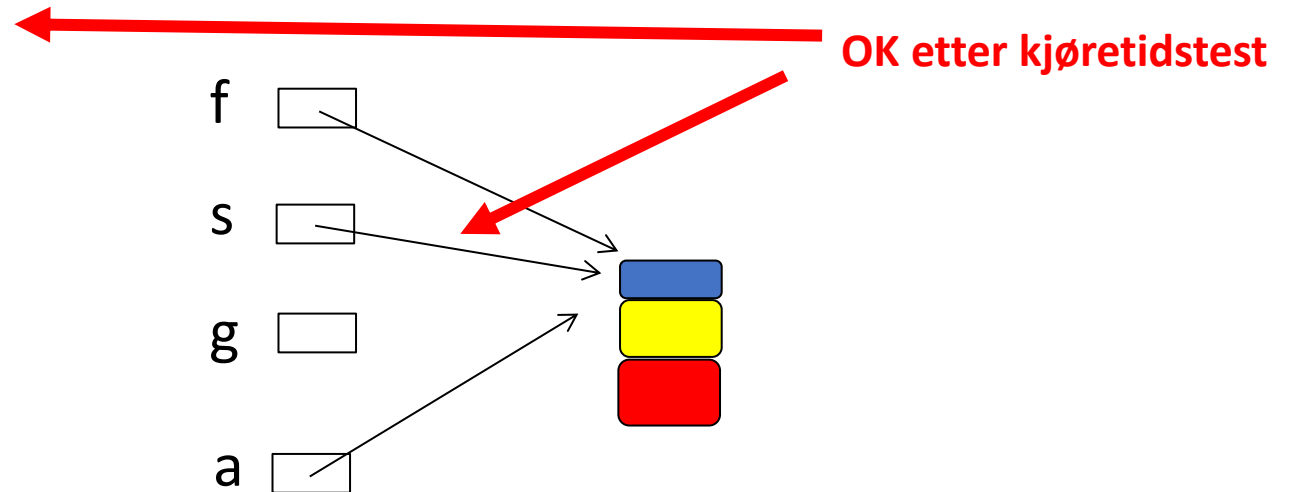
```
Frukt f;  
Sitrusfrukt s;  
Grapefrukt g;  
Appelsin a;  
// Hva er lov:  
a = new Appelsin();  
f = a;  
g = f;  
s = (Sitrusfrukt)f;
```

```
s = f; // Ikke lov  
S = (Sitrusfrukt) f; // Er lov
```

Dette blir en kontroll under kjøring at f virkelig peker på et objekt med Sitrusfrukt-egenskaper



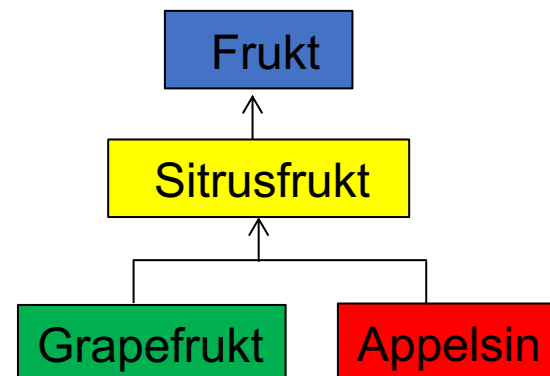
Klassehierarki



Løsning:

La kjøretidsystemet sjekke og **konvertere** om OK

```
Frukt f;  
Sitrusfrukt s;  
Grapefrukt g;  
Appelsin a;  
// Hva er lov:  
a = new Appelsin();  
f = a;  
g = f;  
s = (Sitrusfrukt)f;
```



Klassehierarki

`s = (Sitrusfrukt) f;`

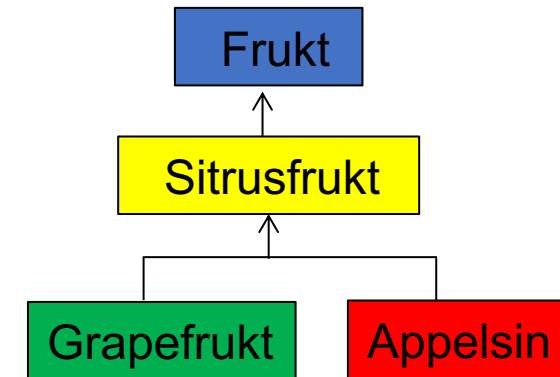
Svaret på uttrykket `f` er en verdi av typen `Frukt`.

Denne verdien kan vi **konvertere** / “**caste**” til en verdi av typen `Sitrusfrukt`.

Etter dette kan vi legge en verdi av riktig type ned i variable `s`.

Løsning: La kjøretidsystemet sjekke

```
Frukt f;  
Sitrusfrukt s;  
Grapefrukt g;  
Appelsin a;  
// Hva er lov:  
f = new Frukt();  
s = (Sitrusfrukt)f;
```

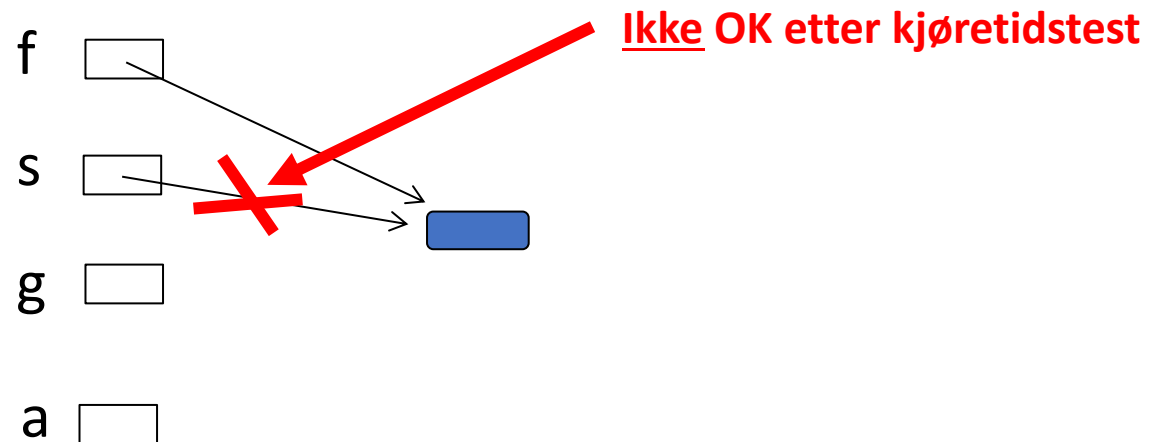


Klassehierarki

Dette blir en kontroll under kjøring om f virkelig peker på et objekt med Sitrusfrukt-egenskaper

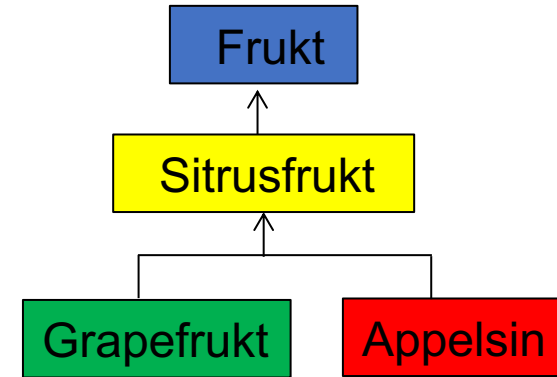
Og her er det ikke OK: Programmet stopper

Bruk: **ClassCastException**

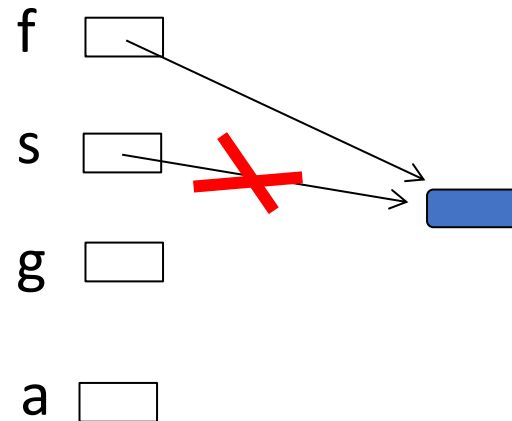


```
Frukt f;  
Sitrusfrukt s;  
Grapefrukt g;  
Appelsin a;  
// Hva er lov:  
f = new Sitrusrukt();  
.  
.  
.  
if (f instanceof Sitrusfrukt)  
    { s = (Sitrusfrukt)f; }
```

If-testen regnes ut til «false» og da vil tilordningen til s ikke bli utført



Klassehierarki

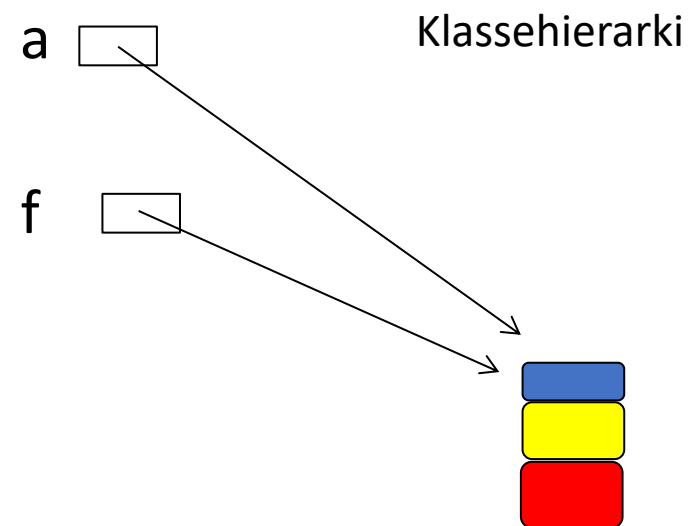
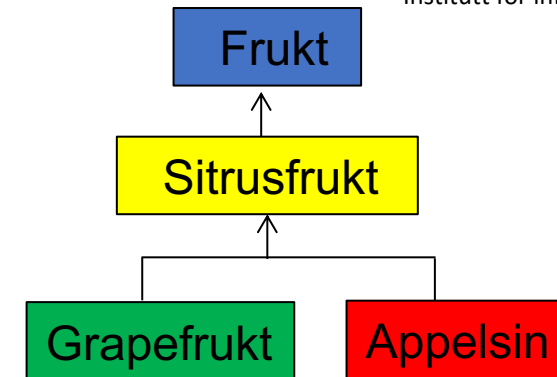


Hva slags objekt er dette?

Ved inngangen til metoden vet referansevariabelen ikke annet enn at f peker på en Frukt.

Men hva slags objekt er det virkelig.

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



Flere eksempler på konvertering av referanser

- Anta at vi har:

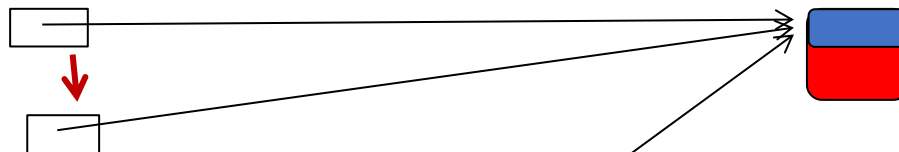
```
class Student extends Person {...}
```

```
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;
```

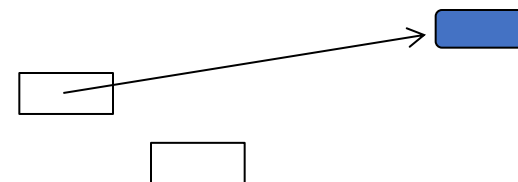


**Dette kalles "casting" (class-cast på engelsk),
typekonvertering på norsk. Medfører kjøretidstest.**

Flere eksempler på konvertering (forts.)

- Hva hvis vi har:

```
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
```

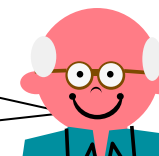
(fordi pers ikke peker på et objekt med alle «Studentegenskapene»)

- Igjen: for å unngå denne feilen, bør **instanceof** brukes:

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

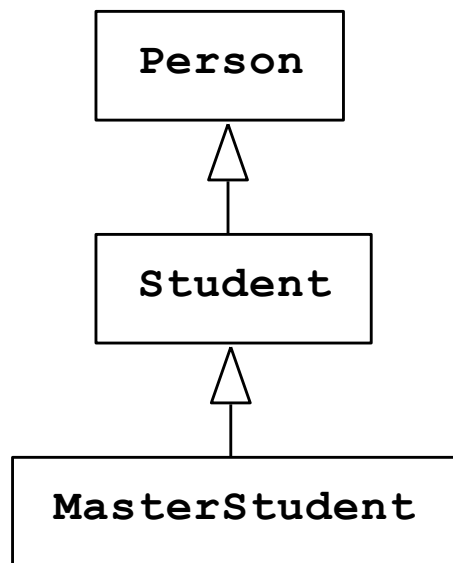
pers instanceof Student =

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



Konvertering mellom flere nivåer

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



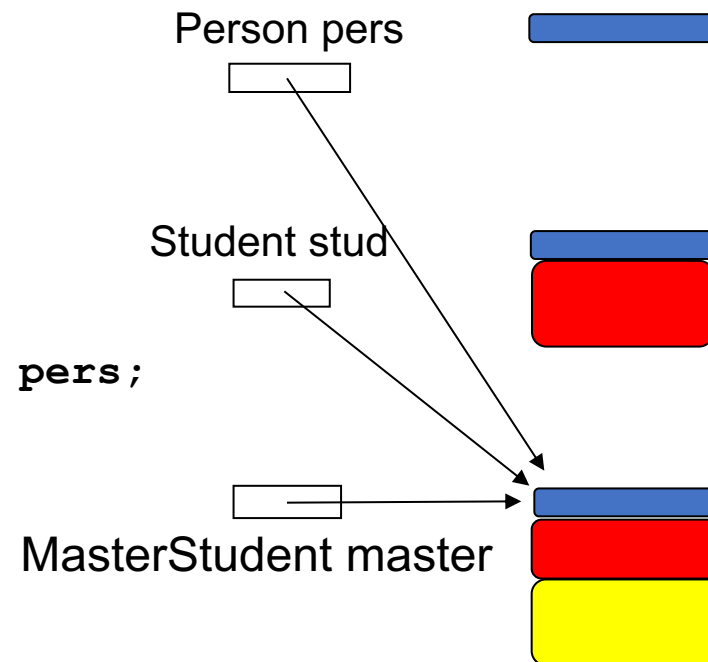
- Konvertering oppover:

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

- Konvertering nedover:

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

(Dette krever kontroll under kjøring)



**Regel: ” Alle referanser har lov til å peke bortover og nedover”
(men ikke ”oppover”)**

Endelig: private og public i subklasser

private gjør at ingen subklasser kan se denne egenskapen

protected 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 / nøkkelord 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 String program”, så ville ingen subklasser til Student kunne se denne egenskapen

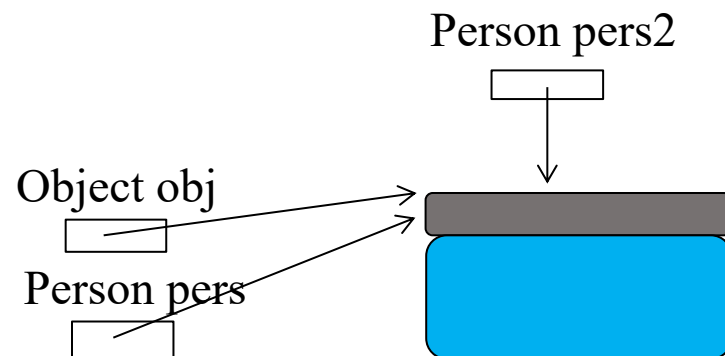
Klassen Object

- `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;
```



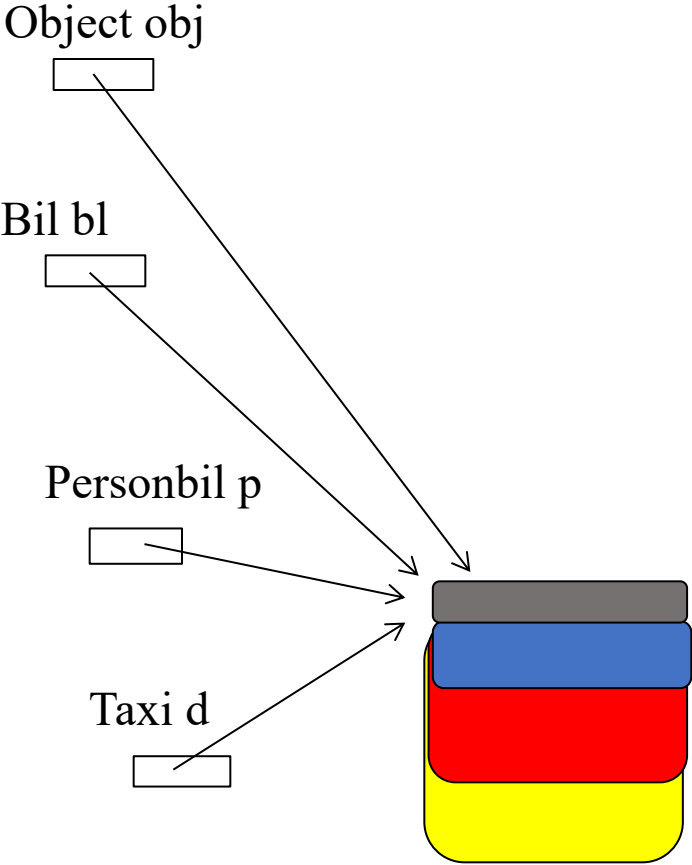
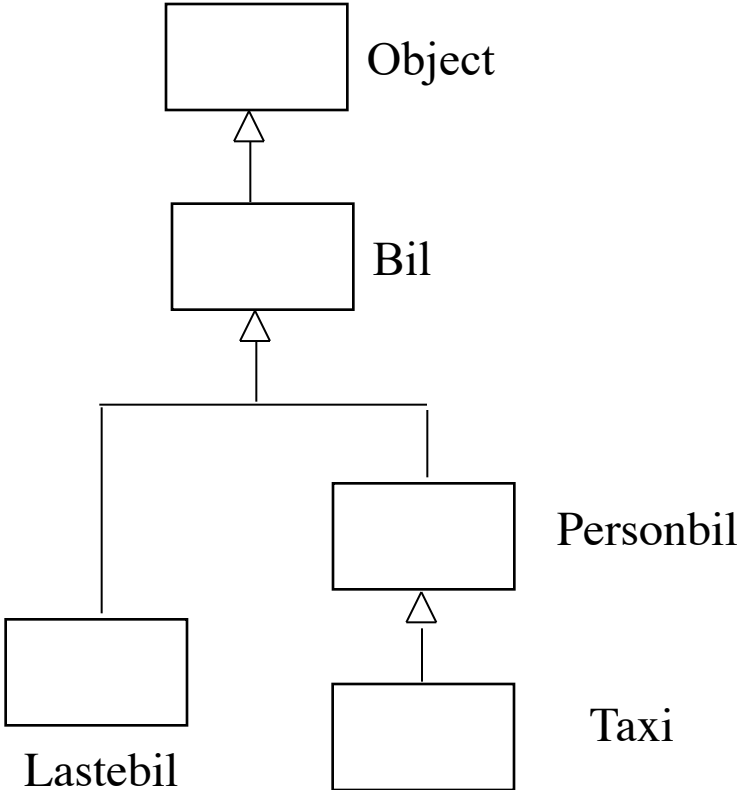
Mer neste gang om hva som er inne
i Object-delen av et objekt.



Hint: Bl.a. `equals()` og `toString()`

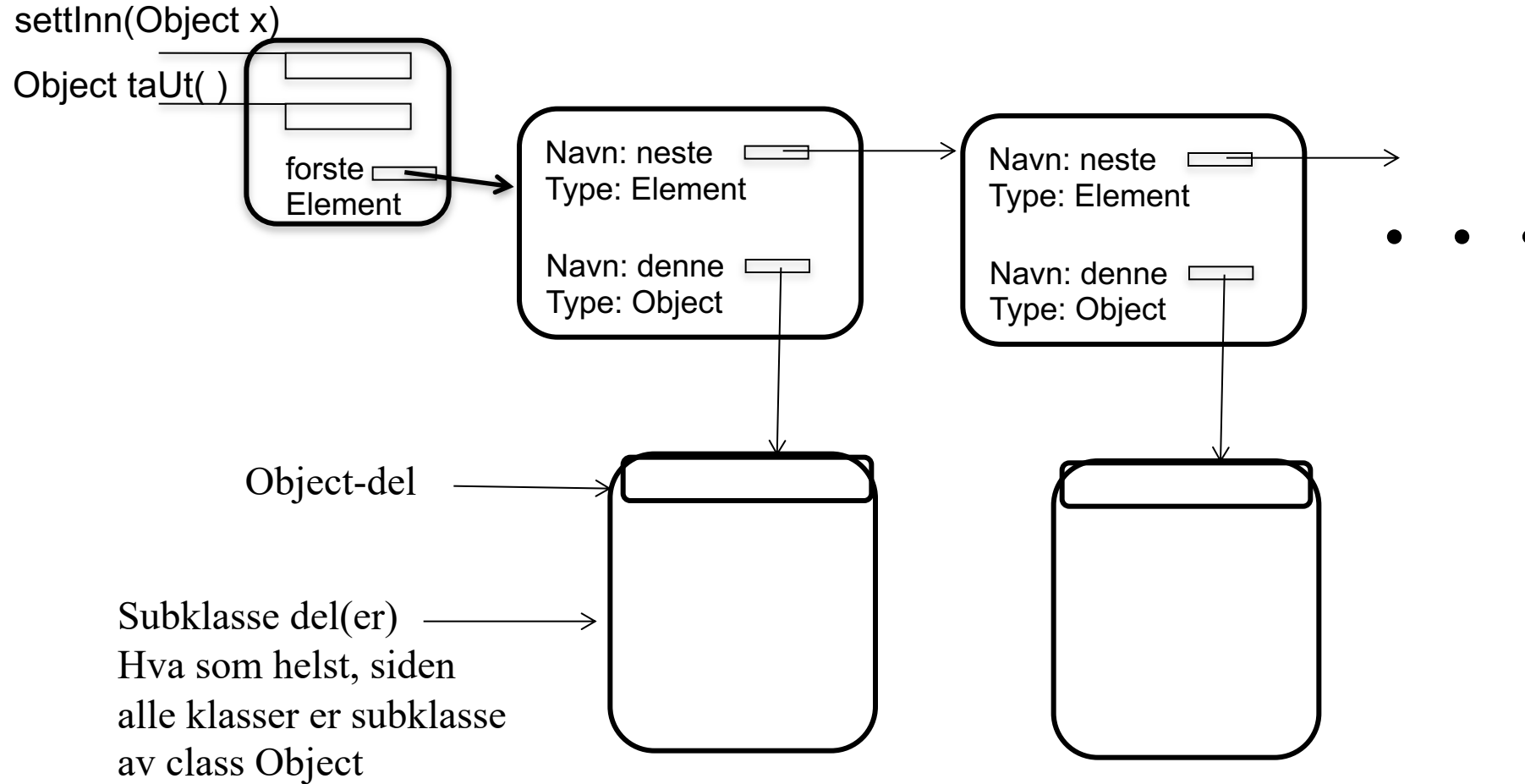
class Object - eksempel

Klasshierarki:



En lenket liste med noder

Dette venter vi noen uker med





Konstanter i Java

- En konstant i Java deklarerer med «final» og kan ikke endres etter at den er initialisert:

```
class KonstDemoKlasse {  
    protected final int objektId;  
    KonstDemoKlasse (int objektId) {  
        this.objektId = objektId;  
    }  
}
```

Helt på siden av dette temaet (men har med scop å gjøre):

«this» er en referanse til objektet som koden er inne i.

Bare når parameteren og instansvariabelen har samme navn brukes «this» i konstruktører/metoder!

```
class KonstDemoKlasse {  
    protected final int objektId;  
    KonstDemoKlasse (int obId) {  
        objektId = obId;  
    }  
}
```

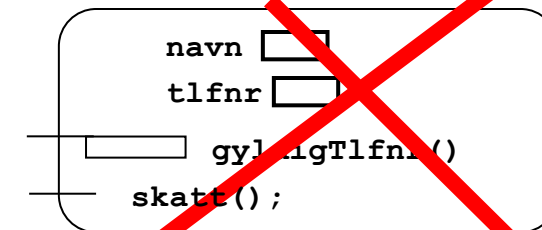
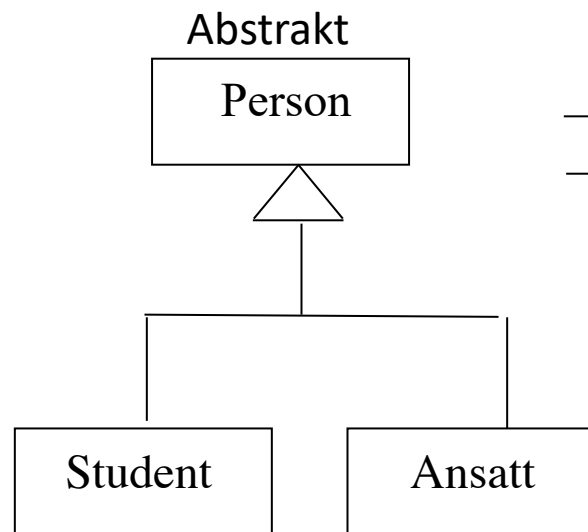
Uten "this" fordi parameter og instansvariabel har forskjellig navn

Abstrakte metoder og abstrakte klasser

```
abstract class Person {  
    protected String navn;  
    protected int tlfnr;  
    public abstract boolean skatt();  
    public boolean gyldigTlfnr() { . . . }  
}
```

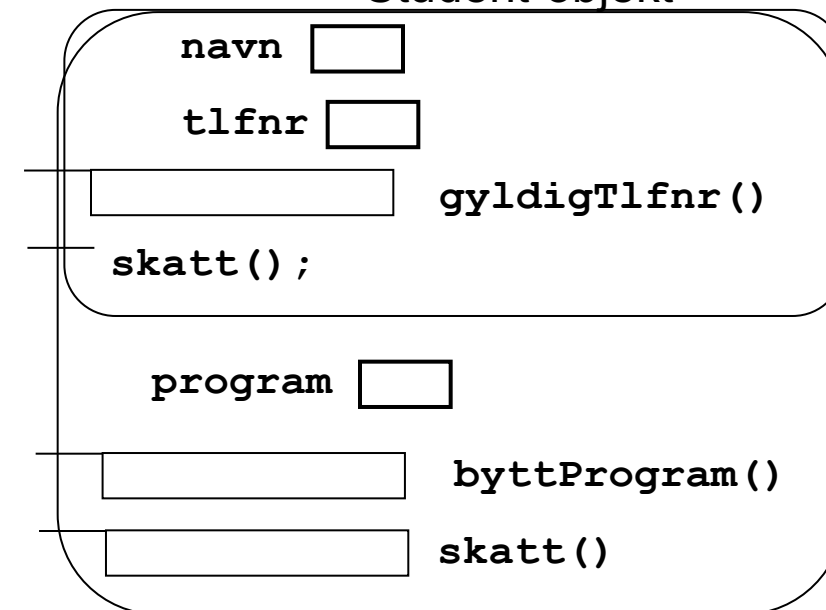
```
class Student extends Person {  
    protected String program;  
    public boolean skatt() {return 0;}  
    void byttProgram(String nytt) { . . . }  
}
```

```
class Ansatt extends Person {  
    protected int lønnstrinn;  
    protected int antallTimer;  
    public boolean skatt() {return 100000;}  
    public void lønnstillegg(int tillegg) { ... }  
}
```



Ikke lov å si
new Person() !

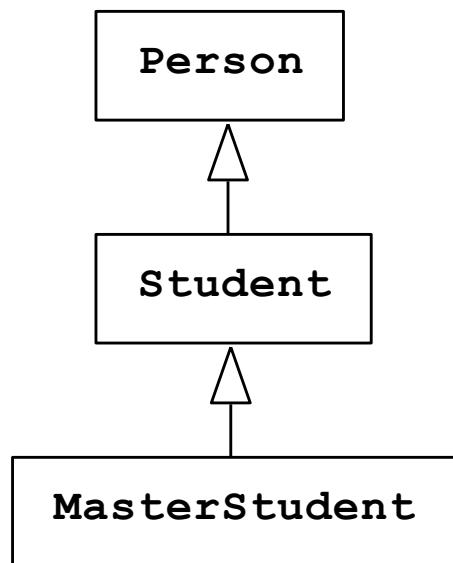
Eksempel på et
Student-objekt



Mer forklaring neste uke

Programmering med subklasser

Hoved-"take away" i dag

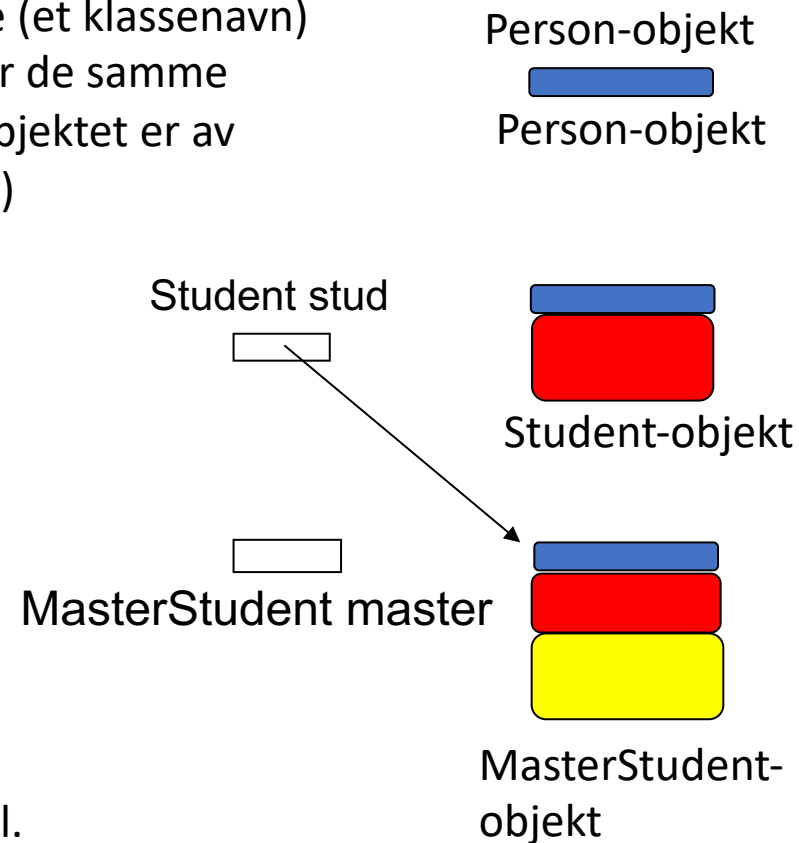


En peker (en referanseverdi) har en type (et klassenavn) og den vil bare peke på et objekt som har de samme (eller flere) egenskapene som klassen (objektet er av den samme klassen eller av en subklasse)

Tror du at objektet har flere egenskaper enn typen til pekeren tilsier, kan du teste dette med **instanceof**, og du kan konvertere verdien til en subklassetype ved "casting", f.eks.:

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

Var dette ikke riktig får du en kjøretidsfeil. Fang «ClassCastException»s.





I dag har vi lært

- Subklasser - extends
- Generalisering - spesialisering
- Subklasser - submengder
- Referanser av subklassetyper / subklassenavn
- Tilordninger mellom referanser - opp og ned i klassehierarkiet
- Test på objektets egenskaper: instanceof
- class Object
- Abstrakte classer og metoder – abstract