UNIVERSITETET I OSLO

Institutt for informatikk

IN1010 våren 2022

Tirsdag 1. 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

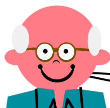
Institutt for informatikk

- 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. 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.

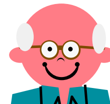
Alle personer skal behandles som . . . .

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

ifi
Institutt for informatikk

Et lite sidesprang

- 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)

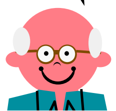
Modeller av båter

Sidespranget avsluttes her

Modeller av fly

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

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



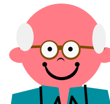
Tilbake til vårt system: Universitetsregisteret

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 . . . .

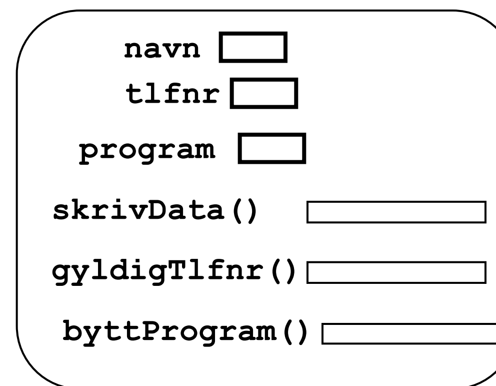
Kristen Nygaard



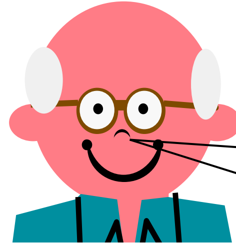
Hva er vårt perspektiv her?

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



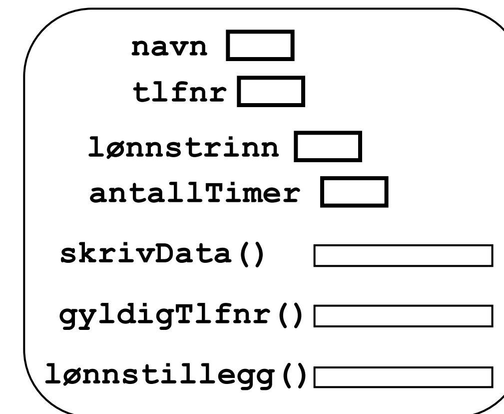
Hm –
på forrige lysark var det ingen egenskaper som var ”private” eller public” ?

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

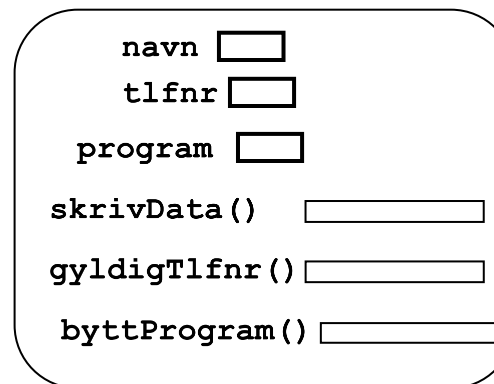
```java
class Ansatt {
    String navn;
    int tlfnr;
    int lønnstrinn;
    int antallTimer;

    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;
    }
}
```



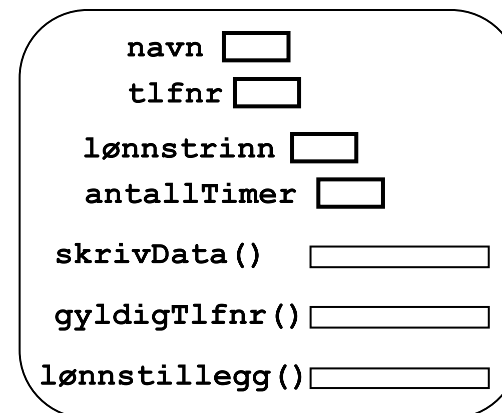
objekt av klassen Ansatt

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

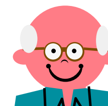
Kan samle det som er **felles** i en egen, mer generell, klasse

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

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

navn
tlfnr
gyldigTlfnr()

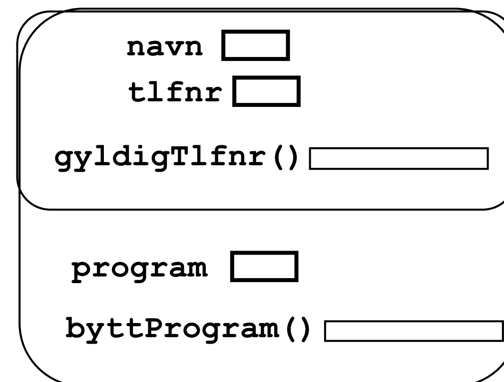
Klassen Person beskriver alt som er felles for studenter og ansatte

Og det er kanskje ikke helt unaturlig



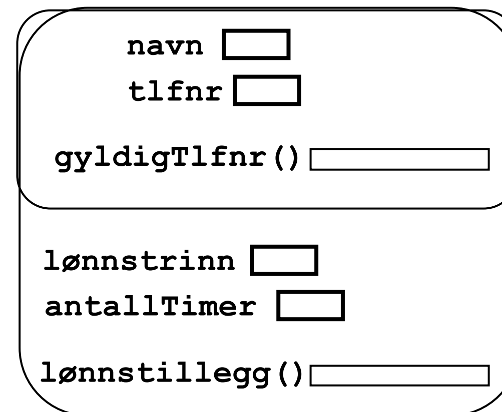
Student og Ansatt som subklasser av Person

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

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



Student-objekt

```java
class Ansatt extends Person {
    int lønnstrinn;
    int antallTimer;

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



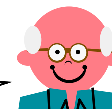
Ansatt-objekt

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.

Nytt Java nøkkelord:
extends



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

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

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

UNIVERSITETET I OSLO

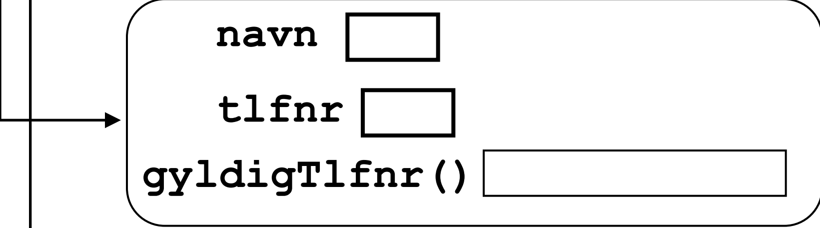
Institutt for informatikk

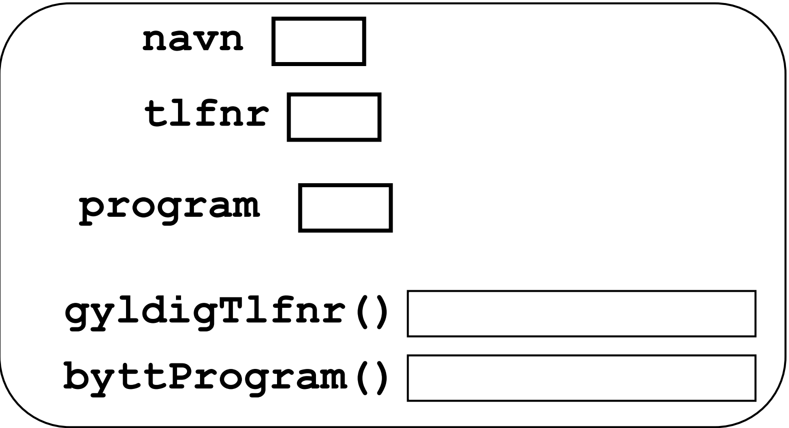
```
class Student {
    String navn;
    int tlfnr;
    String program;

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

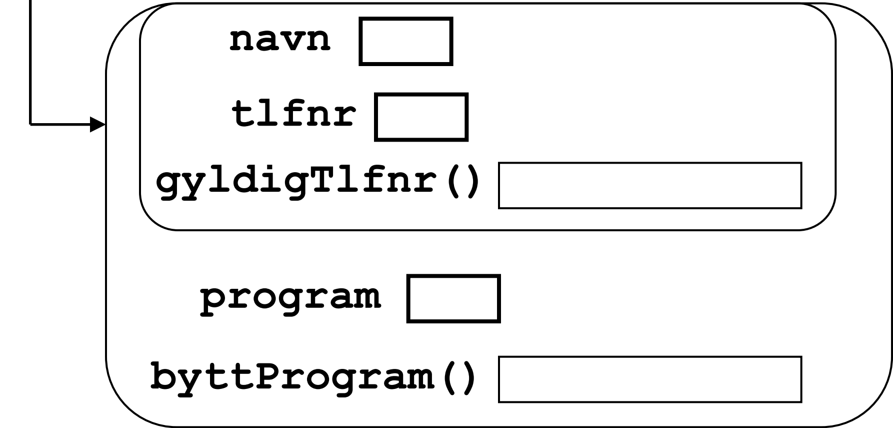
```
class Person {
    String navn;
    int tlfnr;

    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:

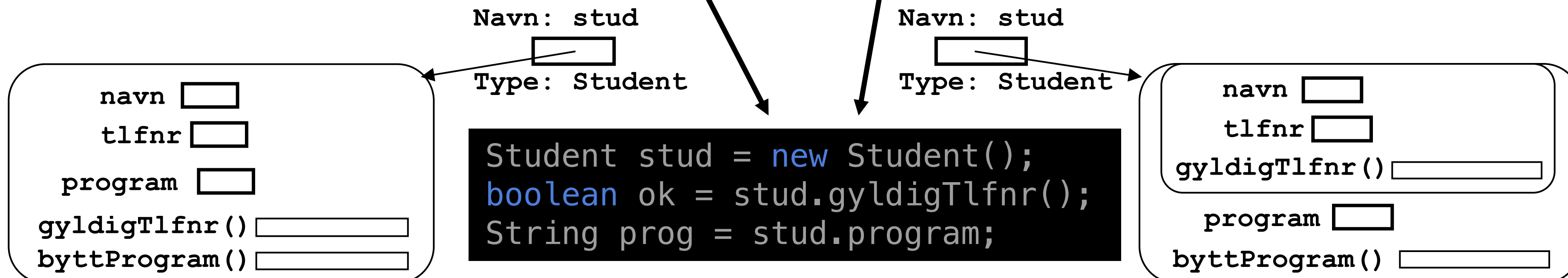
Før:

```
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) { . . .  }
}
```

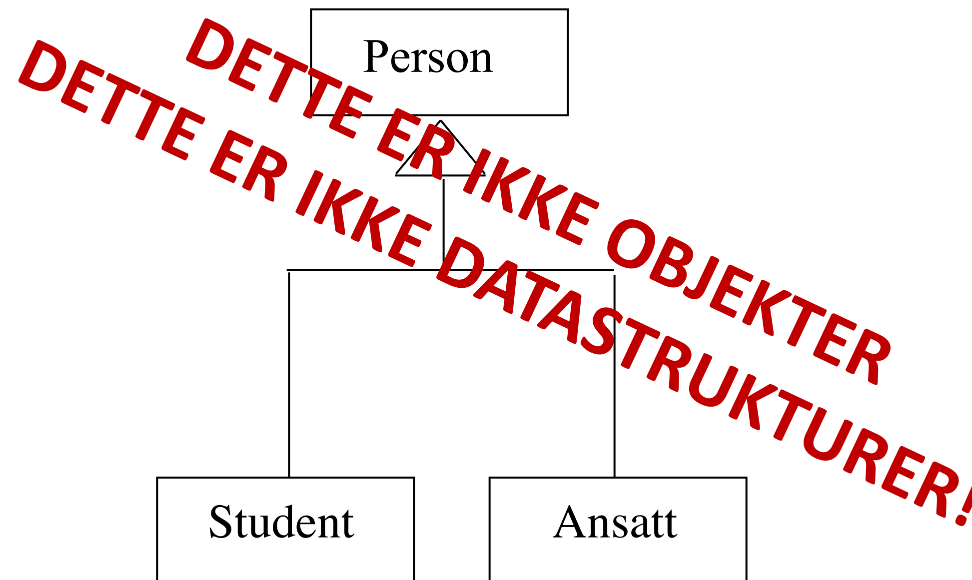


Tegning av subklassehierarki

```
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

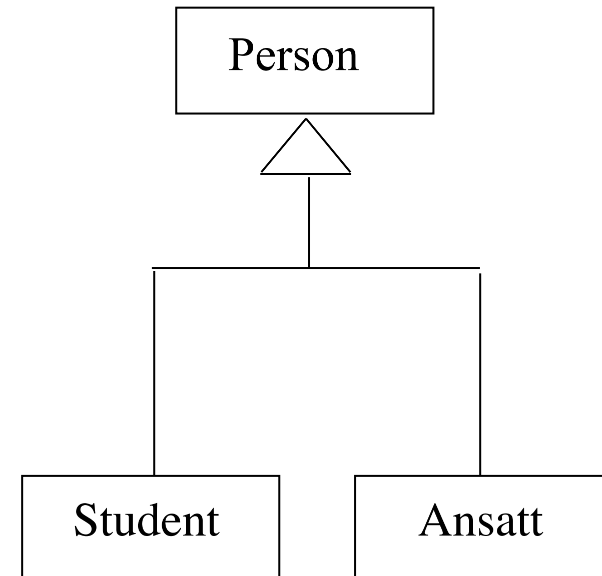
I IN1010 er riktige UML-tegninger ikke viktig (men subklassehierarkier er viktig)

Notasjon for **subklassehierarki**

```
class Person {

}
```

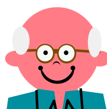
```
class Student extends Person {

}
```

```
class Ansatt extends Person {

}
```



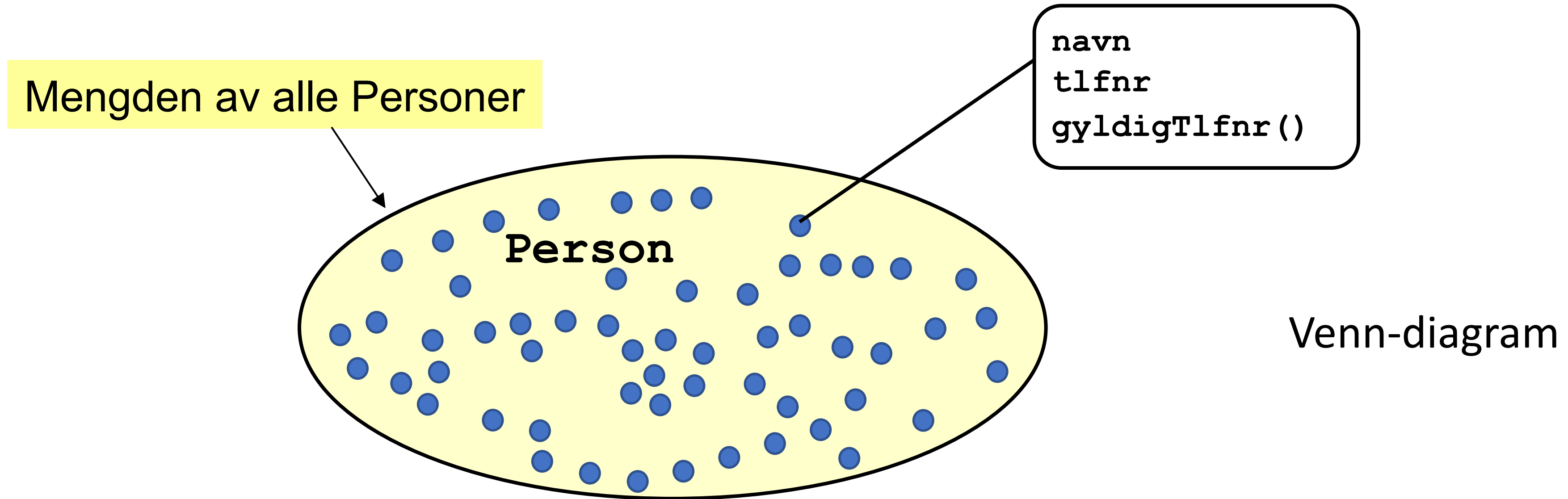
Subklasser og superklasser

- 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/variabler/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



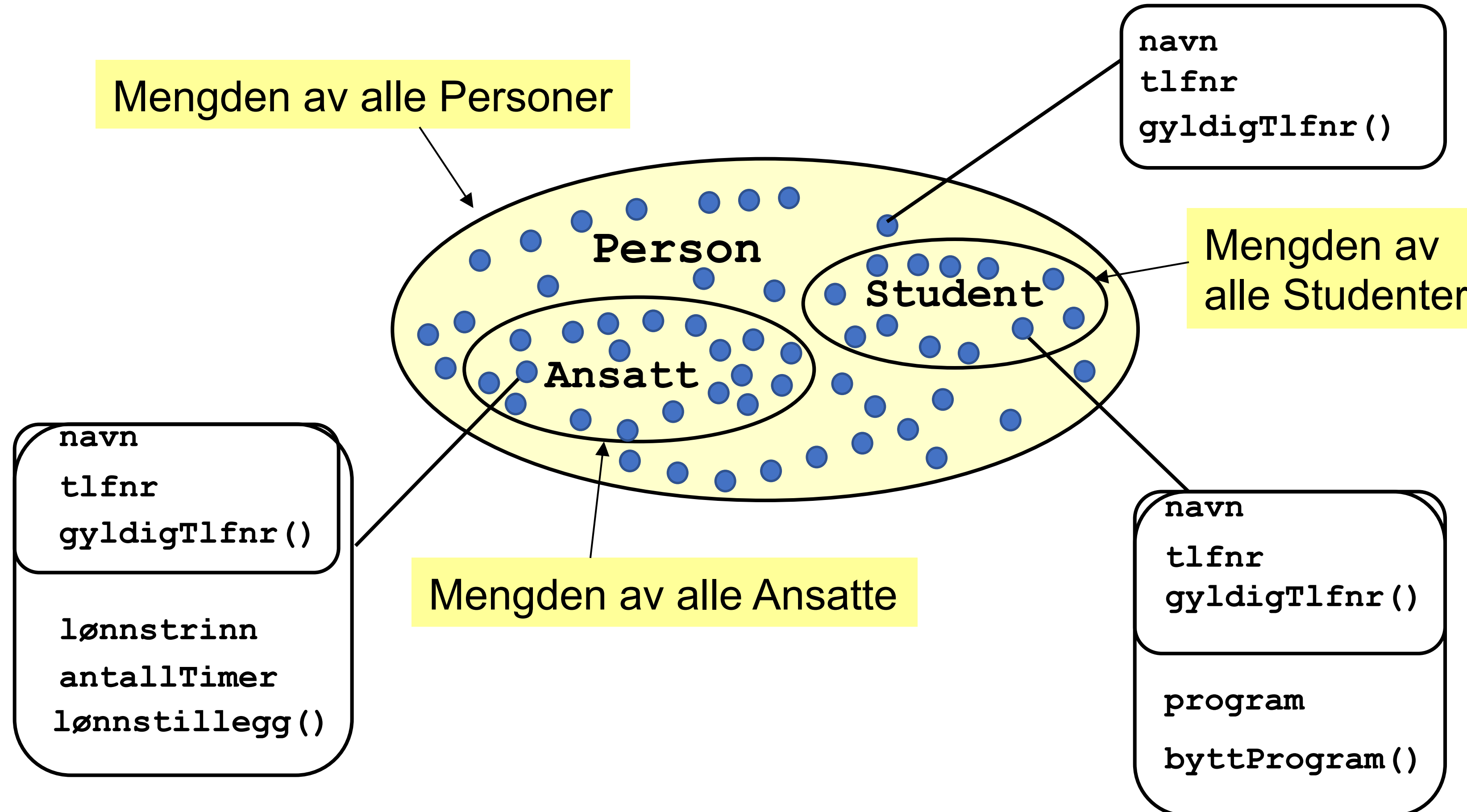
Venn-diagram

- 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**

UNIVERSITETET I OSLO

Spesialisering - Generalisering

Institutt for informatikk



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

UNIVERSITETET I OSLO

Institutt for informatikk

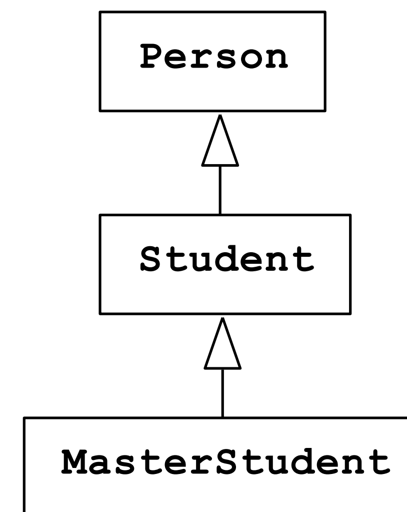
Klasse-hierarkier

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

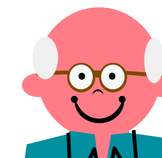
```
class Person {

}
```

```
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.



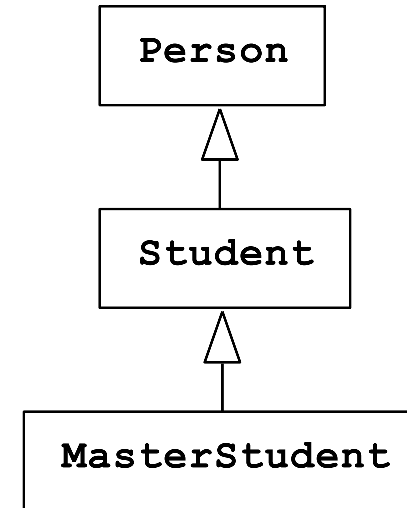
NB! KLASSE-HIERARKI, IKKE DATASTRUKTUR !

Klasse-hierarkier


```
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)

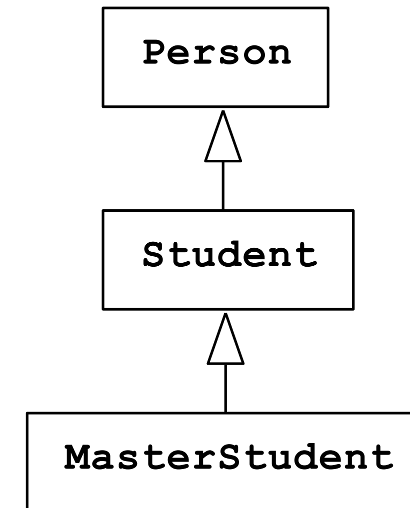
Klasse-hierarkier

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

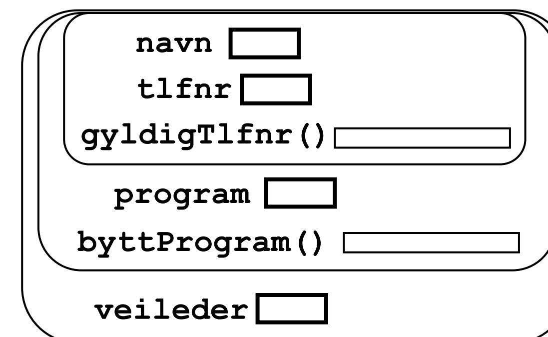
```
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() { . . . }
}
```

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

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



MasterStudent-objekt

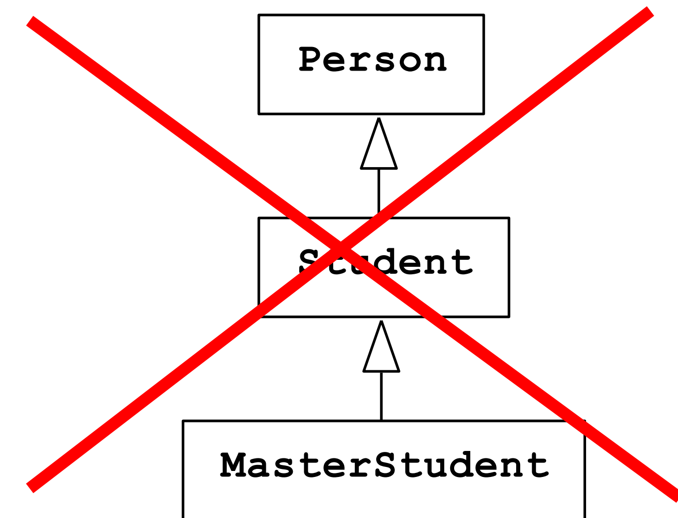


Når programmet utføres lages det datastrukturer / objekter

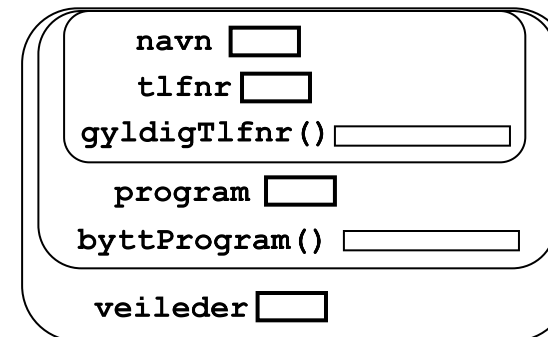

```
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() { . . . }
}
```

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

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

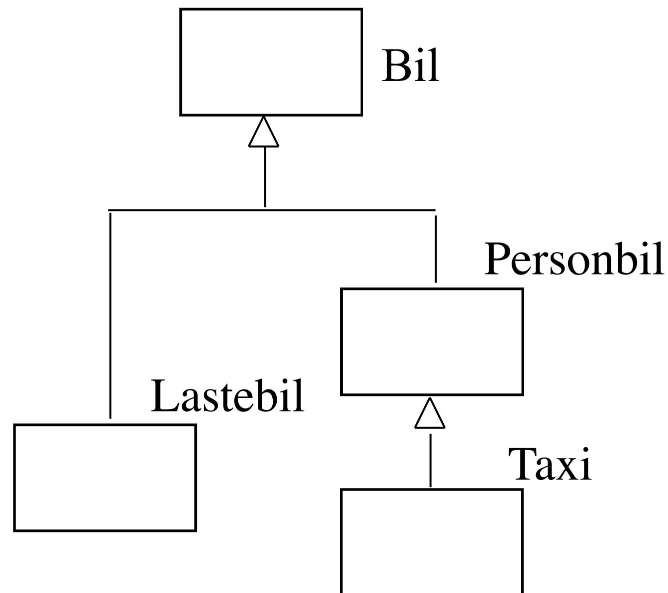


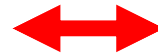
MasterStudent-objekt



Klasser - Subklasser

Klassehierarki:





class Bil { . . . }

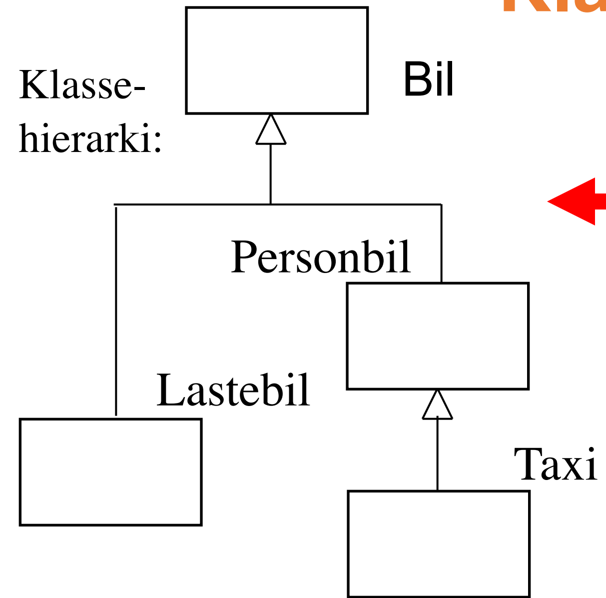
class Personbil extends Bil { . . . }

class Lastebil extends Bil { . . . }

class Taxi extends Personbil { . . . }

29

Klasser - Subklasser




```
class Bil { <blå egenskaper>}
class Personbil extends Bil { <røde egenskaper> }
class Lastebil extends Bil { <brune egenskaper> }
class Taxi extends Personbil { <gule egenskaper> }
```

Alle biler

Alle taxier

Alle personbiler

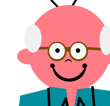
Alle lastebiler

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 forstålig, 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

UNIVERSITETET I OSLO

Institutt for informatikk

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?

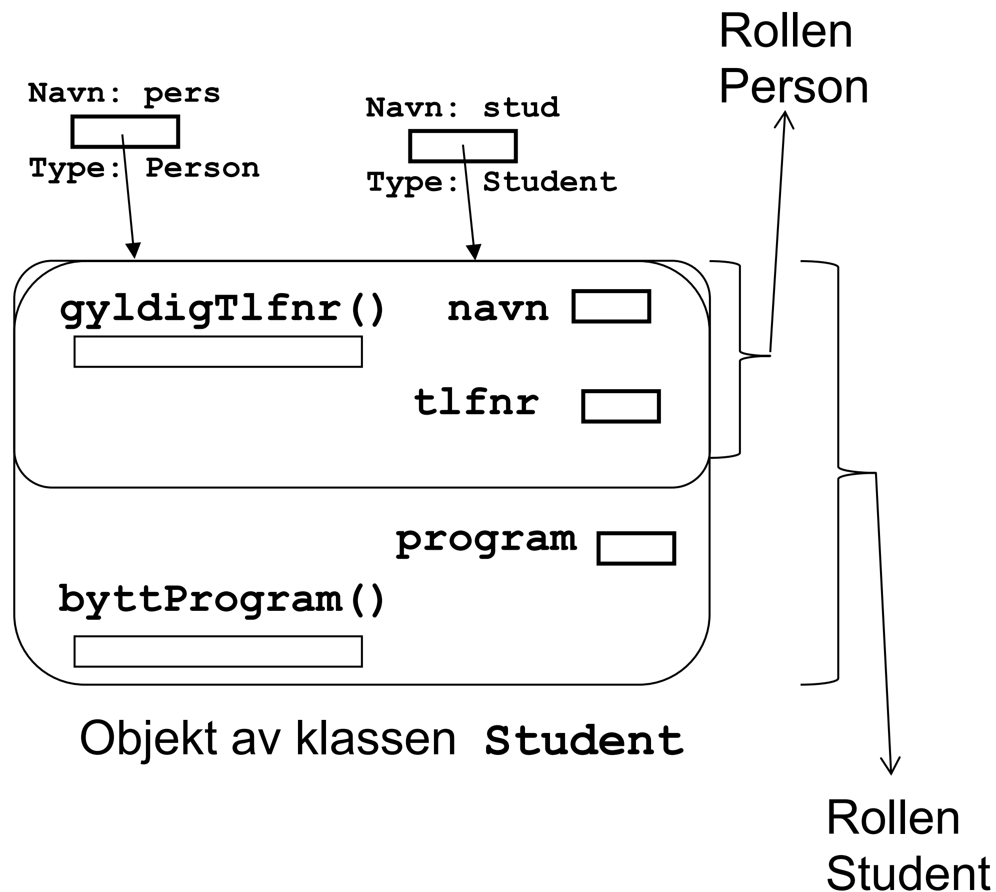
NYTT

Ulike referansetyper (pekertyper)

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

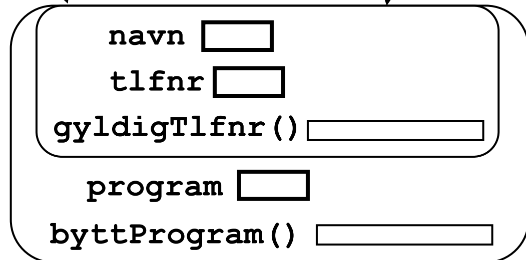
```
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() { . . . }
}
```

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



Objekt av klassen **Student**

Ulike måter å se et objekt på



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

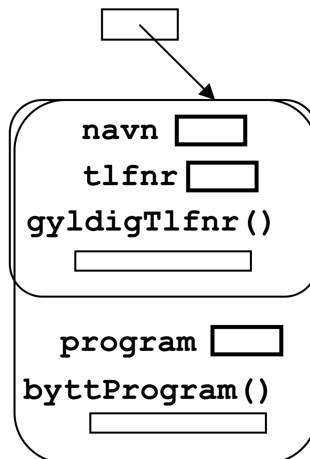
```
Person pers;
Student stud;
stud = new Student();
pers = stud;
```

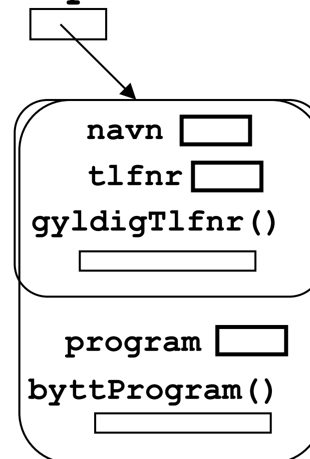
- **Typen (klassen) til selve objektet er uforanderlig.** Et objekt kan likevel ***fremtre** for oss* på ulike måter Det kan spille forskjellige roller.
- Et objekt av klassen
  `class Student extends Person {...}`
  kan vi se på som et objekt av typen (klassen)
  - **`Person`**: da er egenskapene som er spesielle for Student ikke synlige (men de er der fortsatt!).
  - **`Student`**: da er både Person- og Student-egenskapene synlige for oss.
- Det er *referanseverdiens (pekerens) type* som avgjør hvordan objektet fremtrer.
  (med unntak av polymorfe metoder, som vi skal lære om neste uke)

Flere eksempler

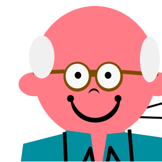
```
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() { . . . }
}
```

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





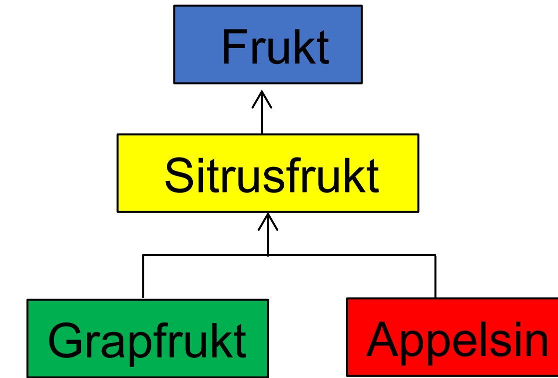

```
// Anta

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**

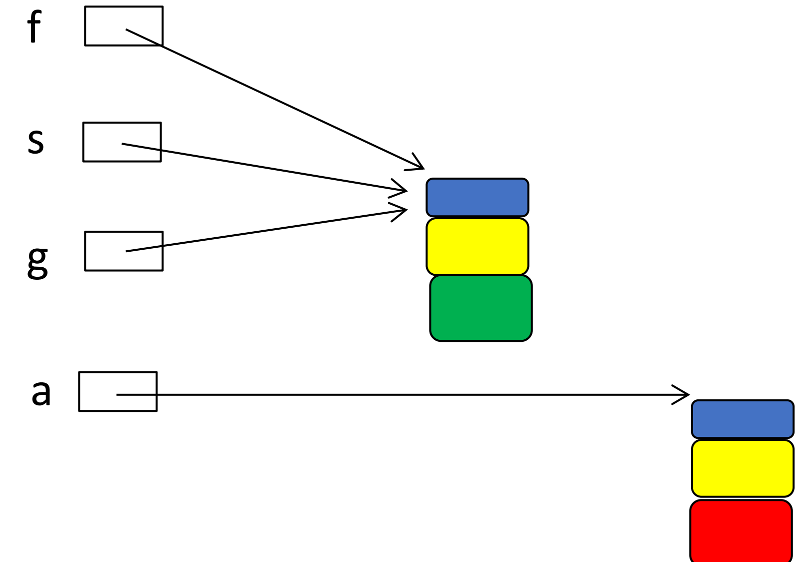
Regel om referanser – Sterk typing


```
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

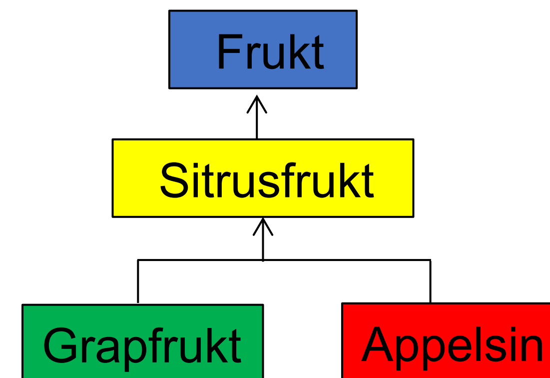


UNIVERSITETET I OSLO

Quiz om referanser til subklasseobjekter

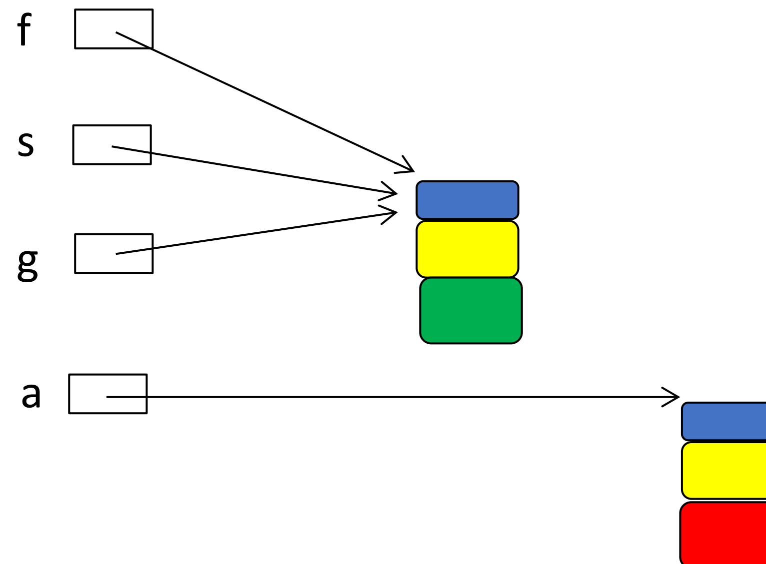
Institutt for informatikk

```
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();
```

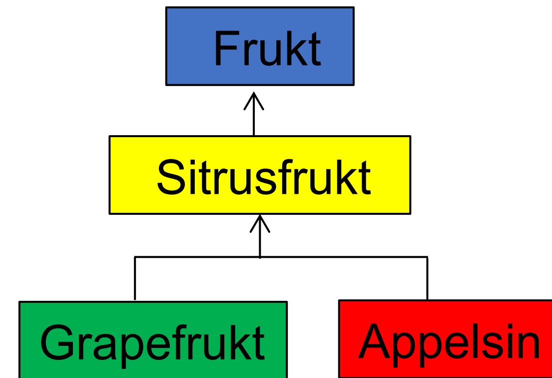
Klassehierarki



Menti-quiz

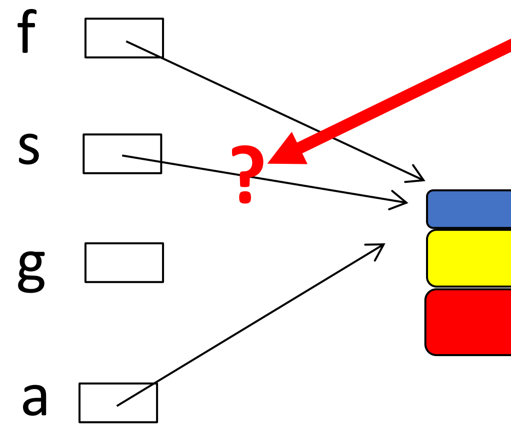
Tilbake til svaret på oppgave 3


```
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 innholder Sitrusfrukt-egenskaper, så dette burde jo være I orden?

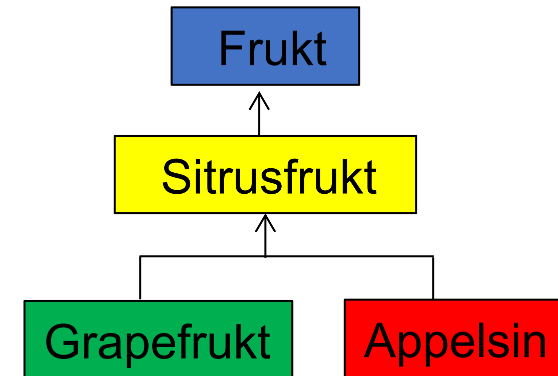
Men det vet ikke kompilatoren !!!

UNIVERSITETET I OSLO

Løsning: La kjøretidsystemet sjekke

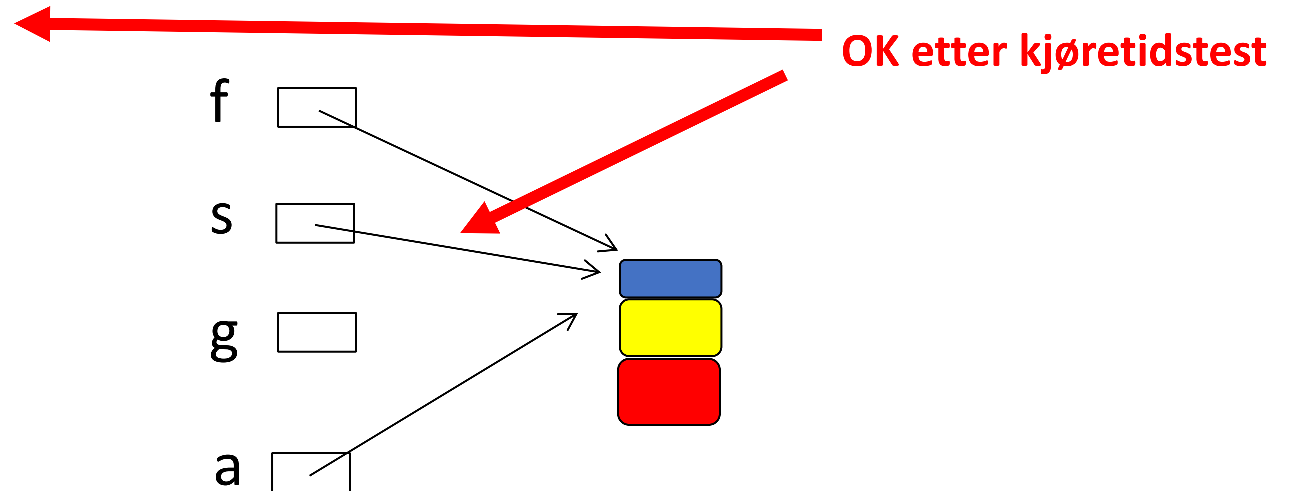
Institutt for informatikk

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



Klassehierarki

OK etter kjøretidstest

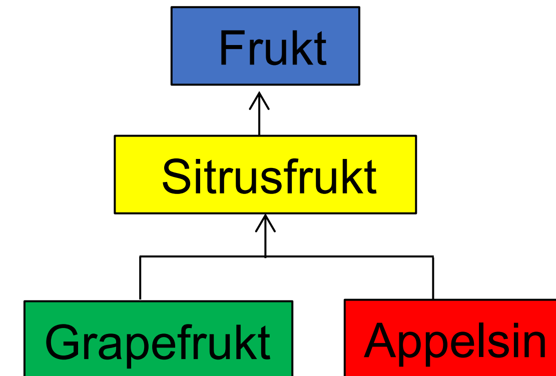


s = f;   // Ikke lov
S = (Sitrusfrukt) f;   // Er lov

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

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 **typekonvertere / "caste"** til en verdi av typen Sitrusfrukt.

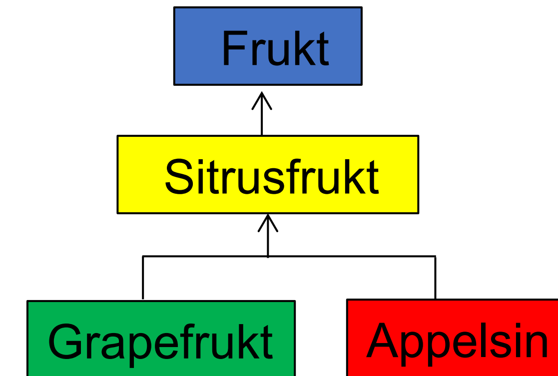
Etter dette kan vi legge en verdi av riktig type ned i variable s.

UNIVERSITETET I OSLO

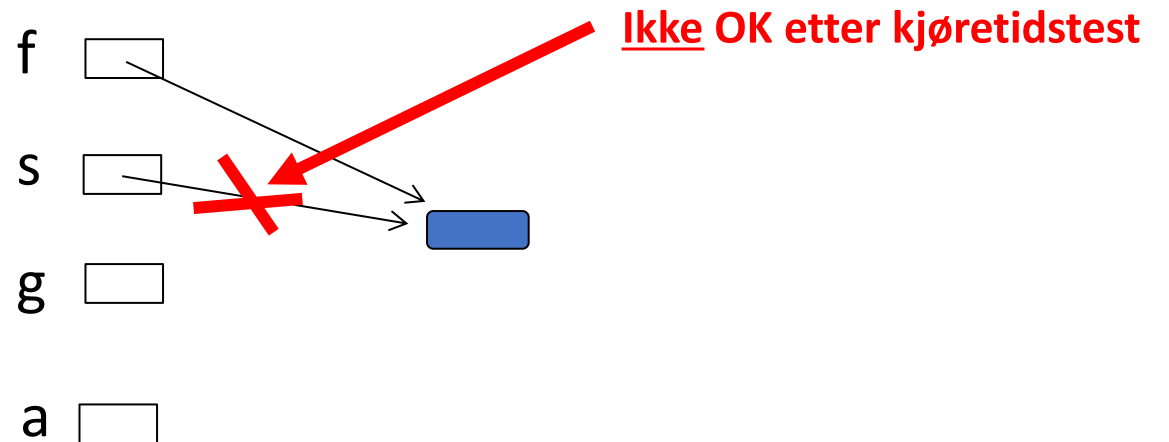
Løsning: La kjøretidsystemet sjekke

Institutt for informatikk

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



Klassehierarki



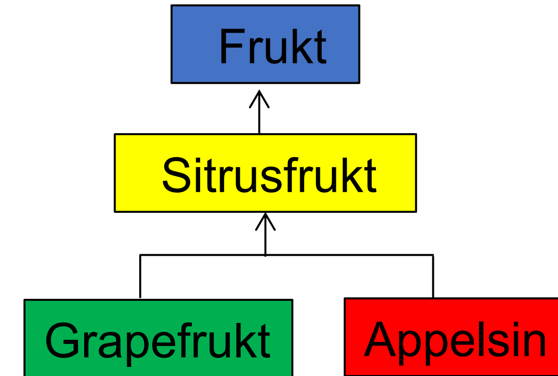
Dette blir en kontroll under kjøring om f virkelig peker på et objekt med Sitrusfrukt-egenskaper

Og her er **det** ikke OK: Programmet stopper

Bruk: **ClassCastException**

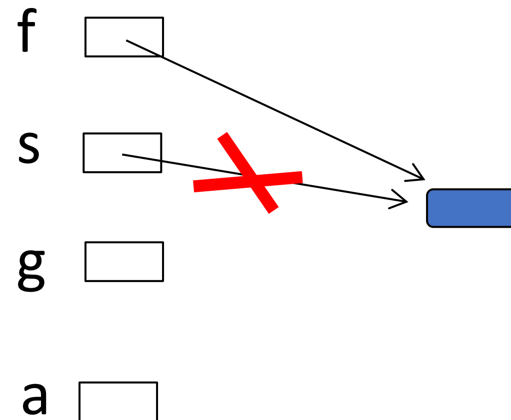
Løsning: instanceof -nytt Java nøkkelord


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



Klassehierarki

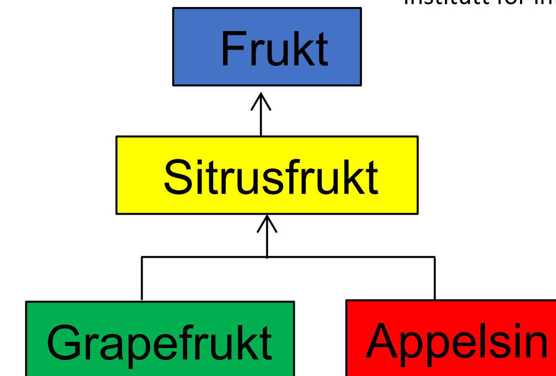
If-testen regnes ut til «false» og da vil tilordningen til s ikke bli utført



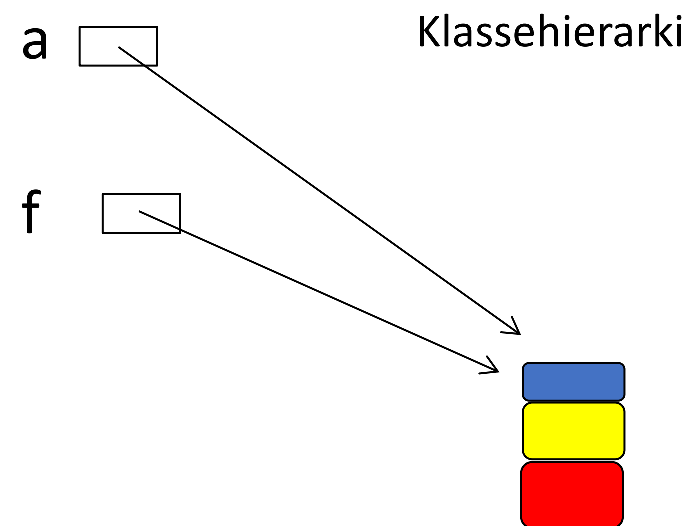
Hva slags objekt er dette?

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

Men hva slags objekt er det virkelig.




```
class TestFrukt {
    public 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!");
     }
}
```



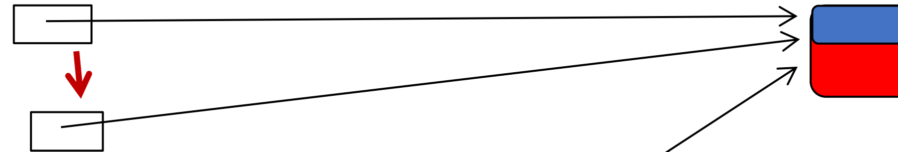
Flere eksempler på konvertering av referanser

- Anta at vi har:

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

- Ved tilordningen

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



har vi en **implisitt typekonvertering** fra Student- til Person-referanse.

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

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

Typekonvertering: “Casting” på engelsk

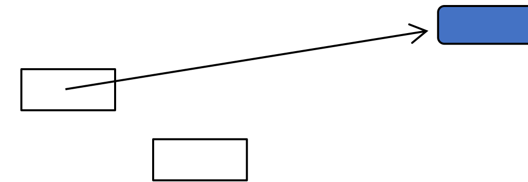
UNIVERSITETET I OSLO

Flere eksempler på konvertering (forts.)

Institutt for informatikk

- Hva hvis vi har:

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



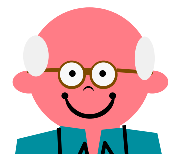
- Dette godkjennes av kompilatoren, men ved kjøring går det galt, og vi får feilmeldingen

```
java.lang.ClassCastException
(fordi pers ikke peker på et objekt med alle ”Studentegenskapene)
```

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

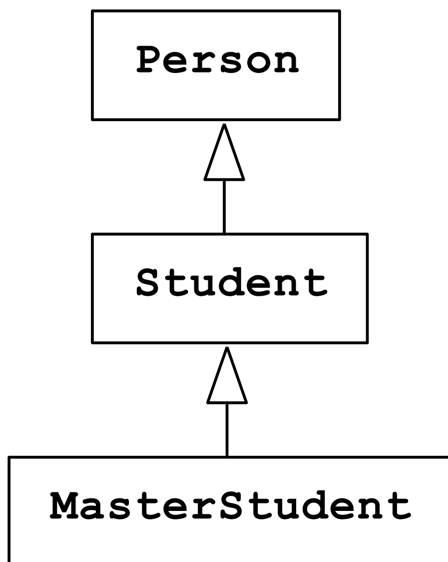
```
if (pers instanceof Student) {
    Student stud = (Student) pers;
}
```

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



Konvertering mellom flere nivåer

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



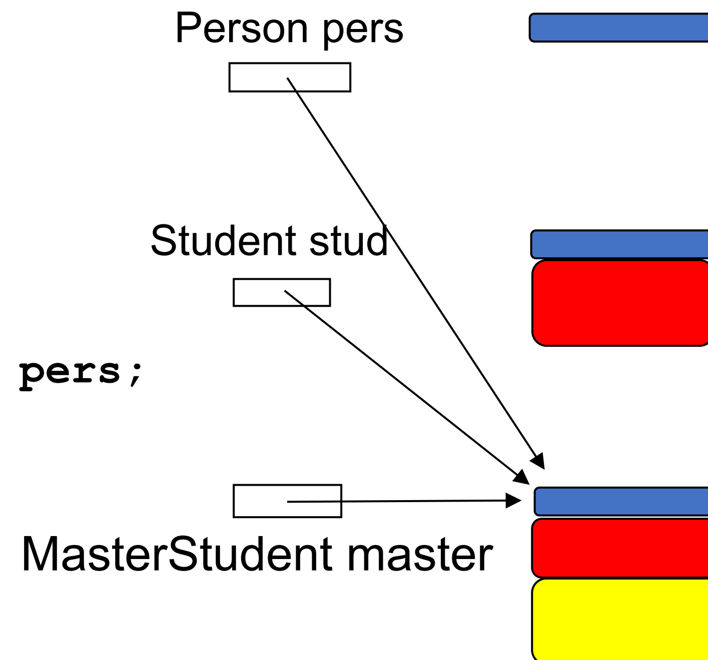
- **Konvertering oppover:**

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

- **Konvertering nedover:**

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

(Dette krever kontroll under kjøring)



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

(men ikke "oppover")

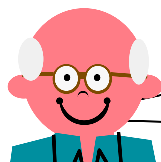
Endelig: private og public i subklasser

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

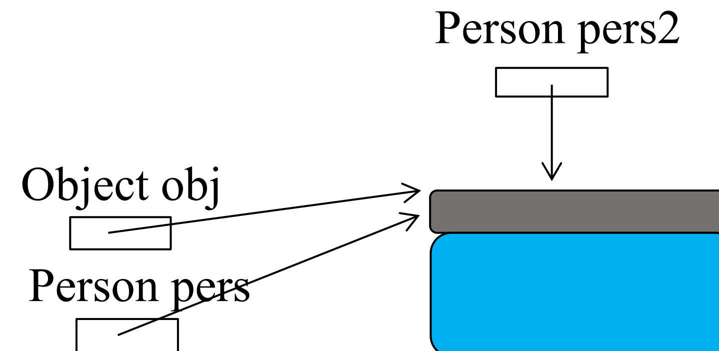
```
    class Person { … }
```

  så tolker Java dette som

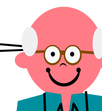
```
    class Person extends Object { … }
```

- 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;
```



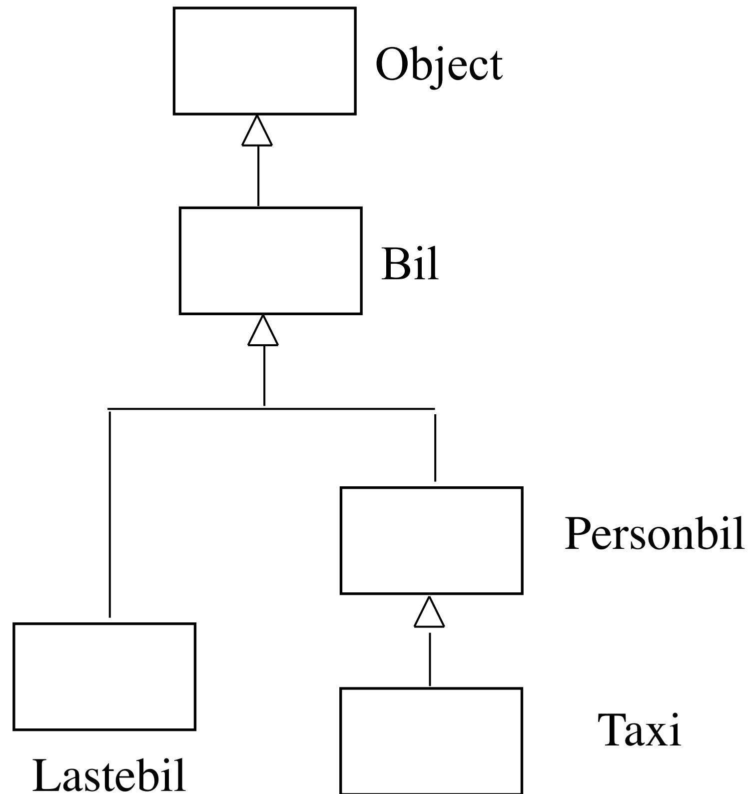
Mer neste gang om hva som er inne i Object-delen av et objekt.

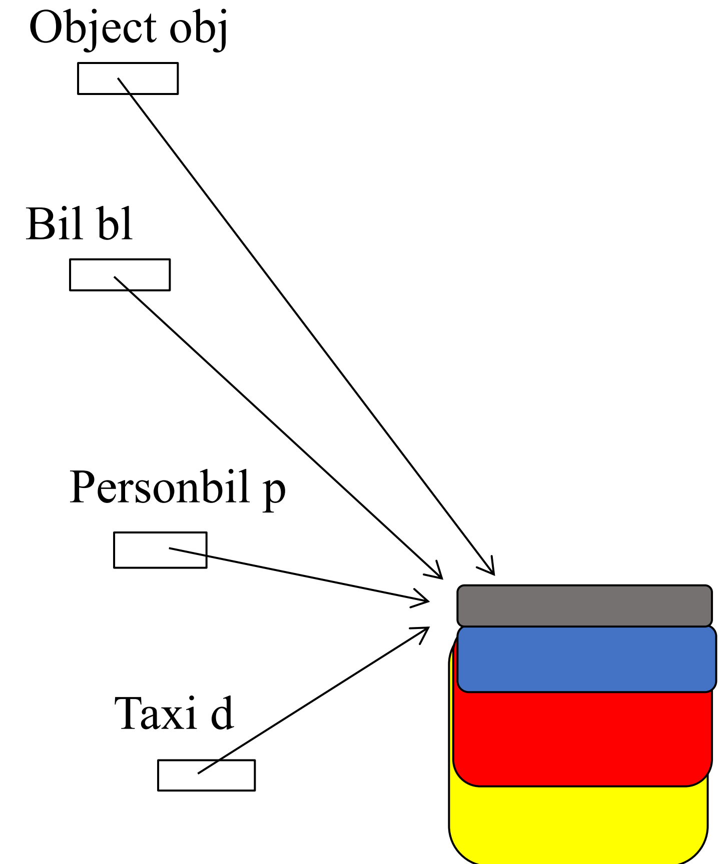


Hint: Bl.a. equals(. . .)

class Object - eksempel

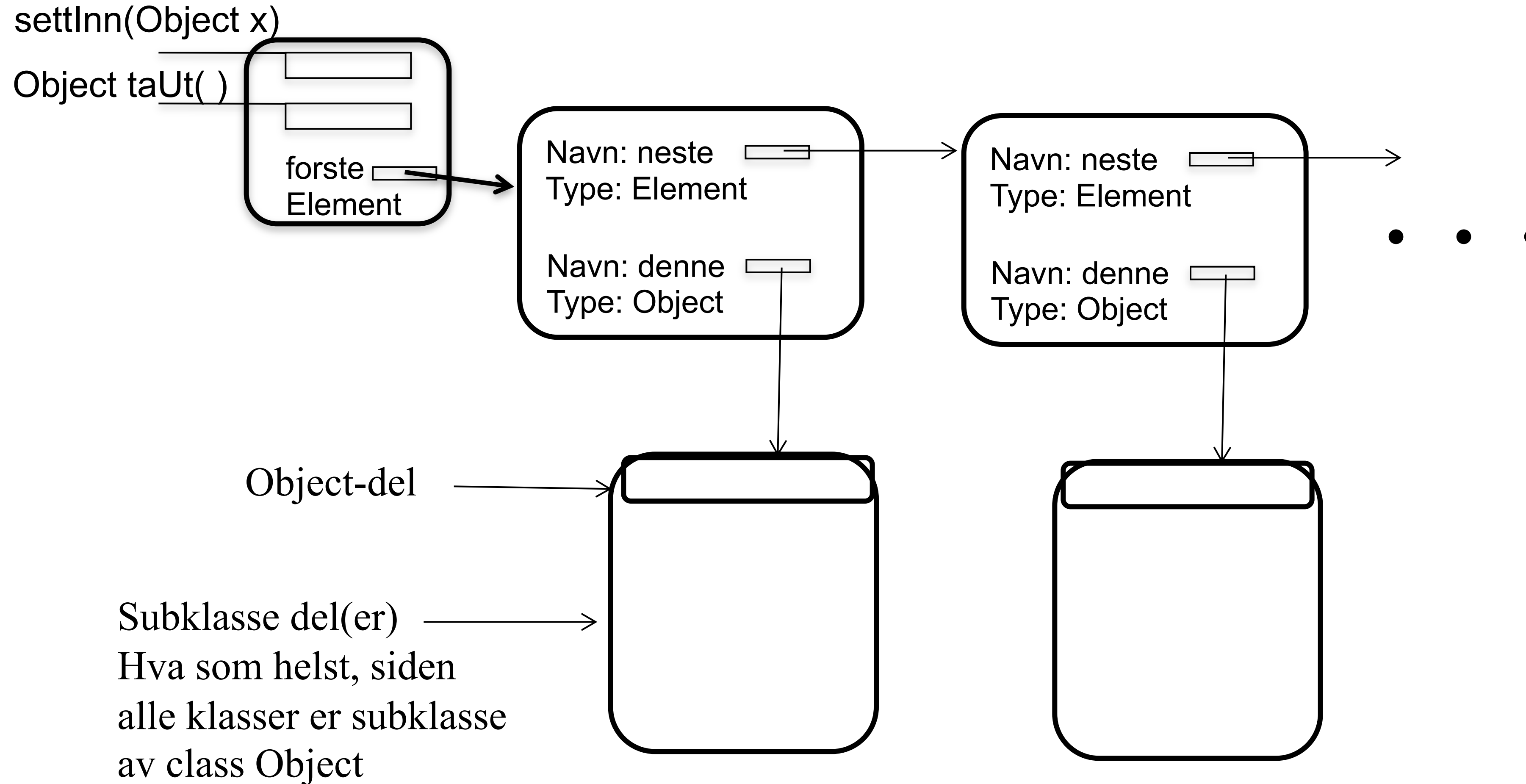
Klassehierarki:





En lenket liste med noder

Dette venter vi noen uker med



Konstanter i Java

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

```
class KonstantDemoKlasse {
    protected final int objektId;
    KonstantDemoKlasse (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.

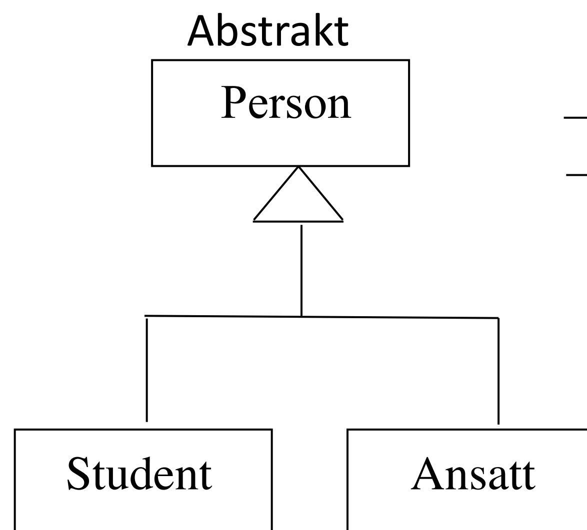
Bare når parameteren og instansvariabelen har samme navn brukes «this» i konstruktører!

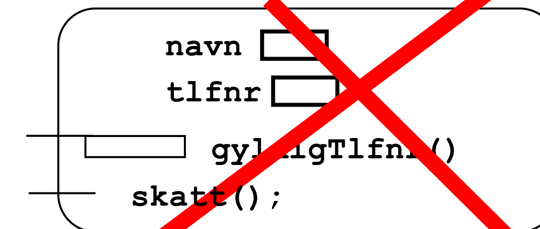
```
class KonstantDemoKlasse {
    protected final int objektId;
    KonstantDemoKlasse (int obId) {
            objektId = obId;
    }
}
```

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() { . . . }
}
```

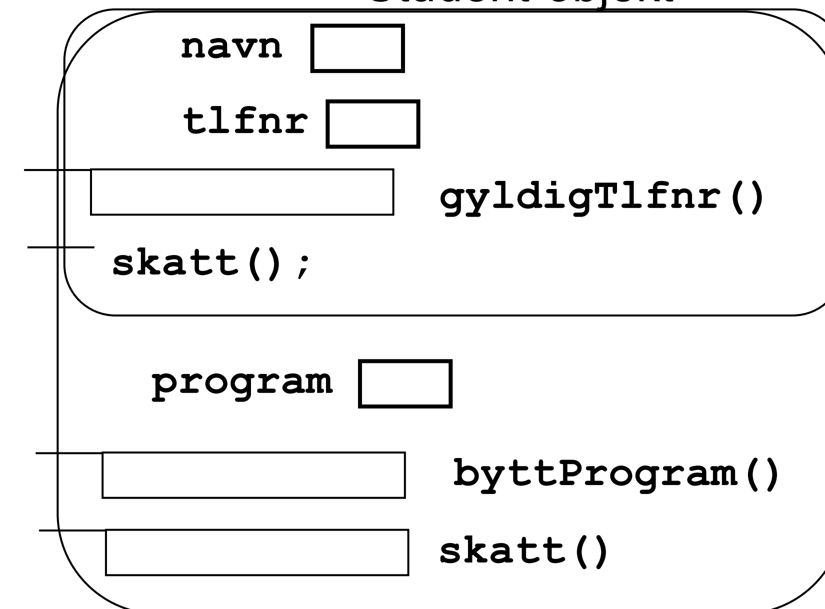




Ikke lov å si new Person() !

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

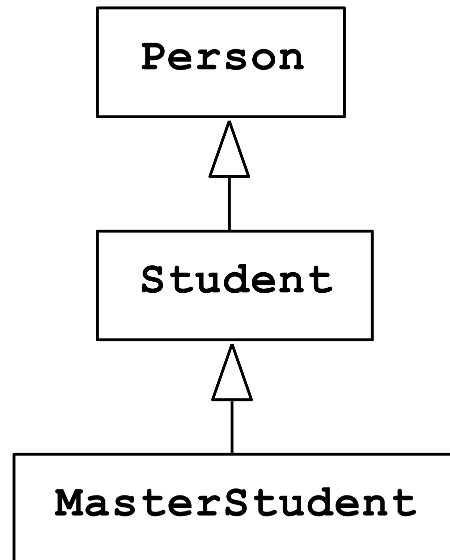
Eksempel på et Student-objekt




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

Mer forklaring de to neste ukene

Programmering med subklasser Hoved-"take away " i dag

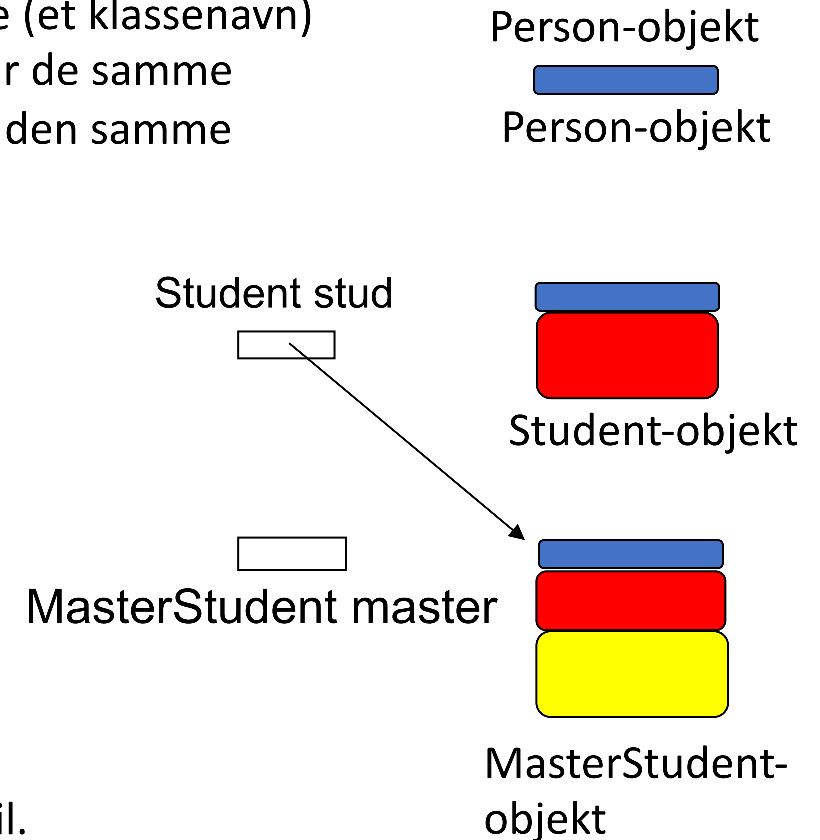


En peker (en referanseverdi) har en type (et klassenavn) og den vil bare peke på et objekt som har de samme (eller flere) egenskapene (objektet er av den samme klassen eller av en subklasse)

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

Institutt for informatikk

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