UNIVERSITETET I OSLO

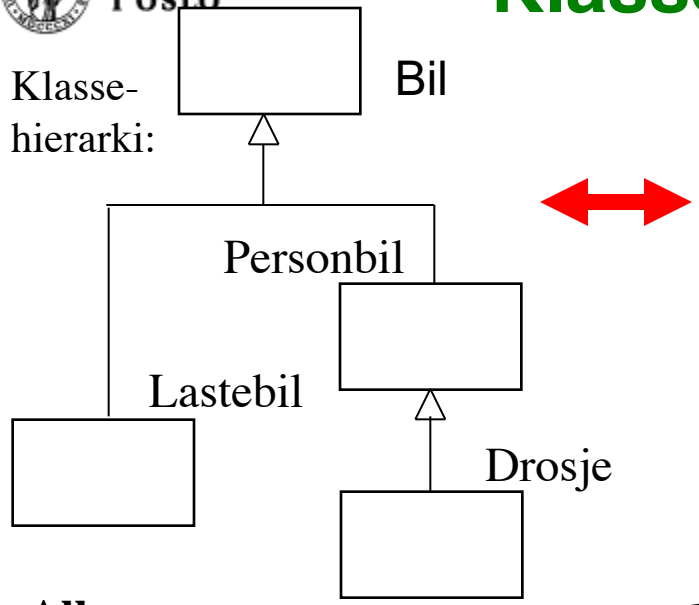
IN1010 våren 2018

Tirsdag 15. mai

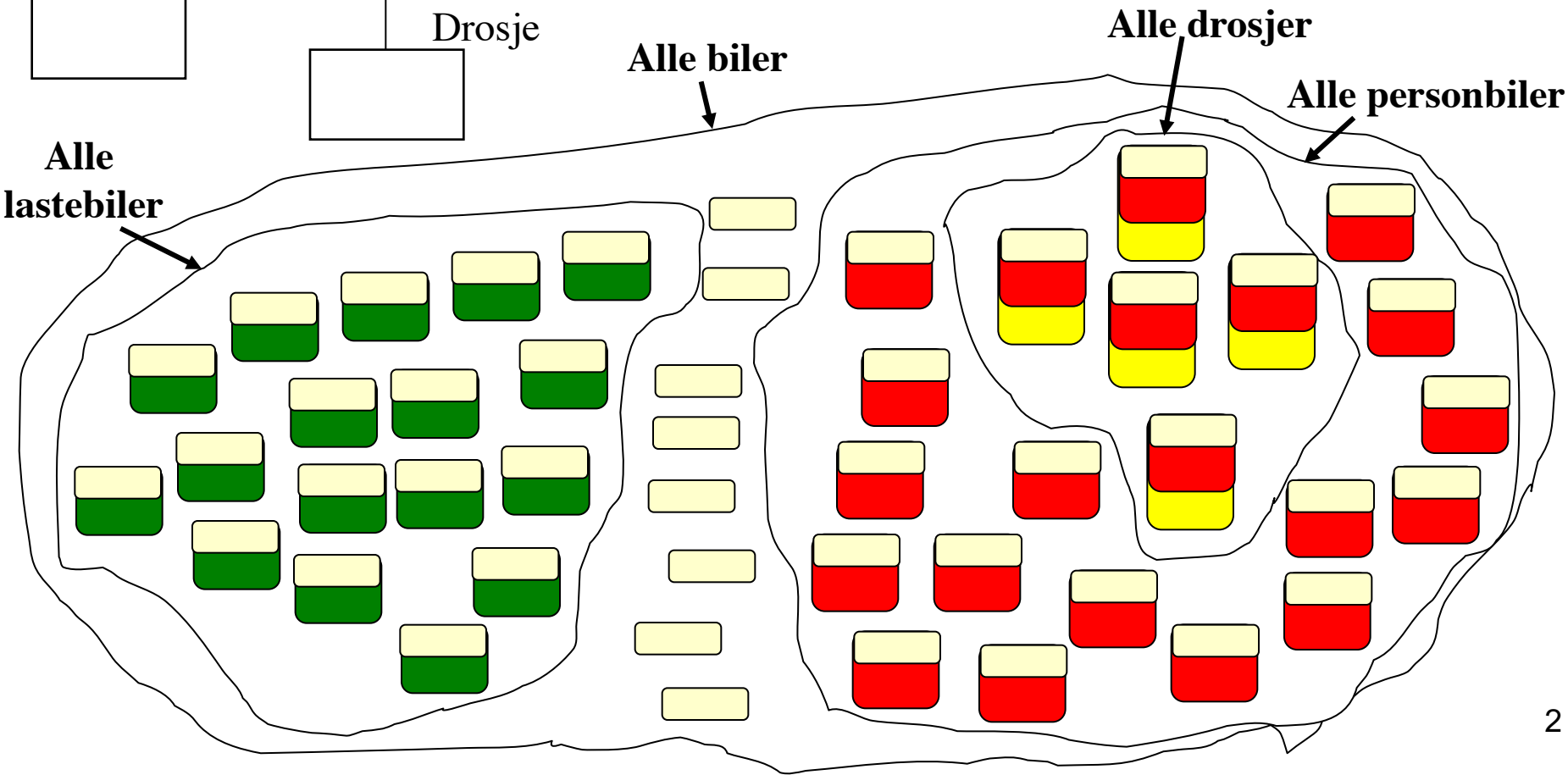
Repetisjon av subklasser og tråder

Stein Gjessing
Institutt for informatikk
Universitetet i Oslo

Klasser - Subklasser




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



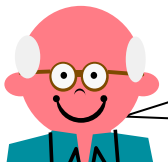
Private og public i subklasser

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;
        }
}
```

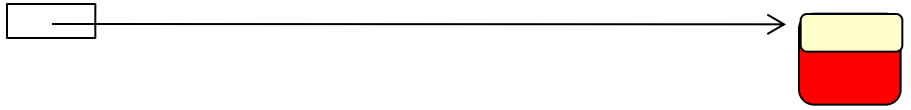


Reservert ord i Java:
protected

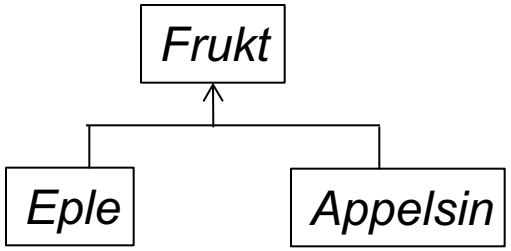
Hva slags objekt er dette?

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!");
    }
}
```

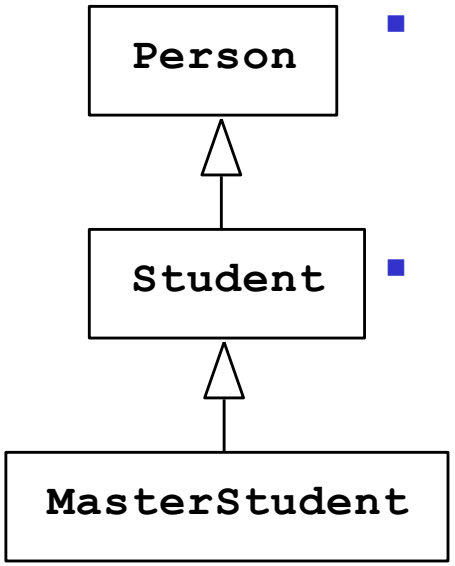


```
class Frukt { .. }
class Eple extends Frukt { .. }
class Appelsin extends Frukt { .. }
```



Konvertering mellom flere nivåer

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



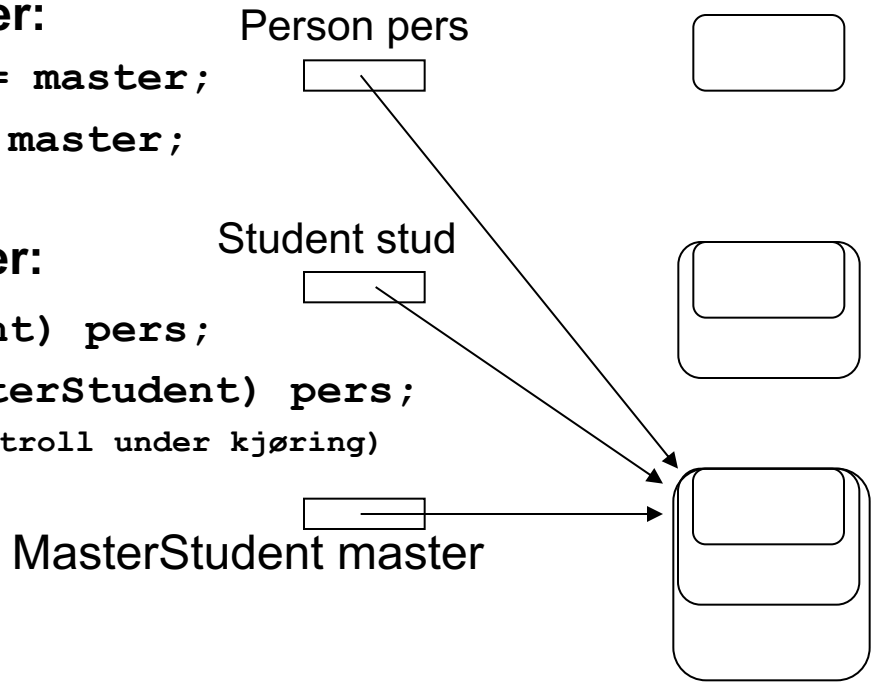
- **Konvertering oppover:**

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

- **Konvertering nedover:**

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

(fordi dette krever kontroll under kjøring)

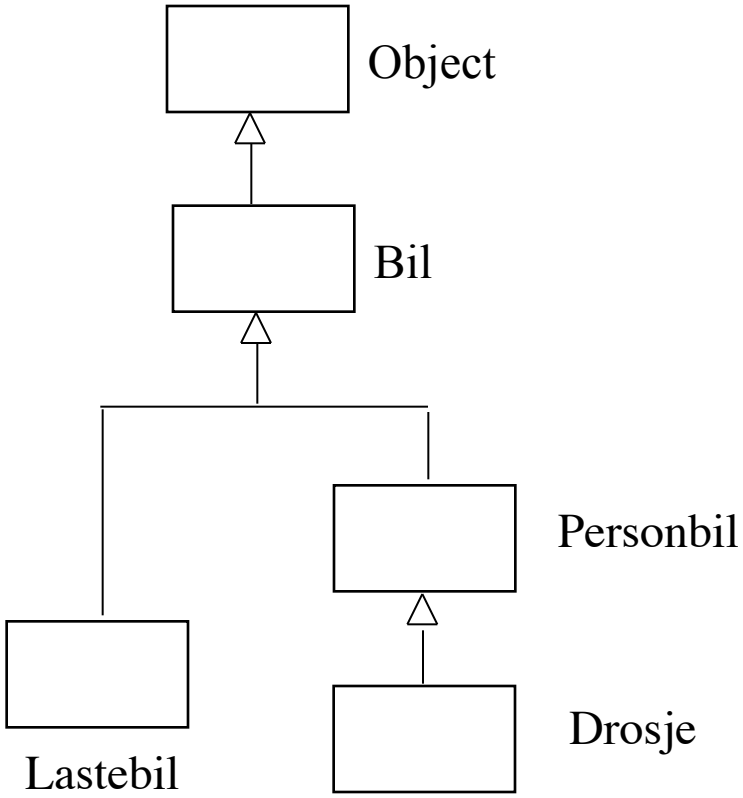


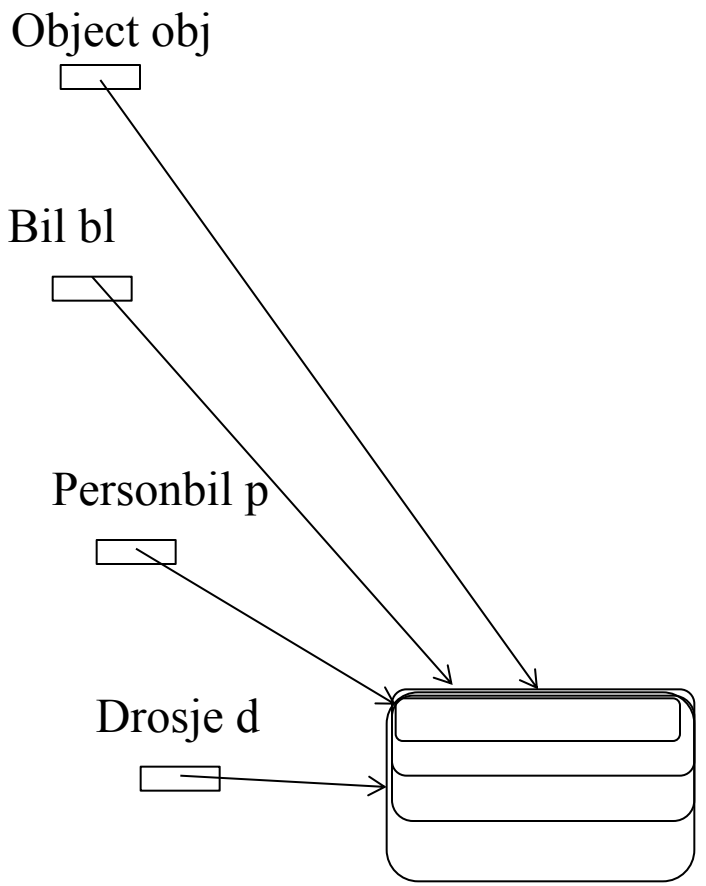
Regel: ”Alle referanser har lov til å peke bortover og nedover”

(men ikke ”oppover”)

Eksempel

Klassehierarki:

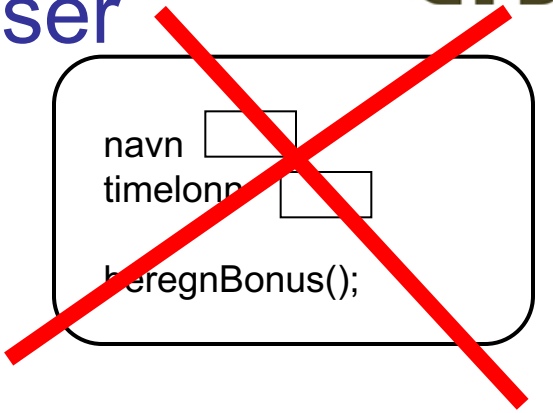


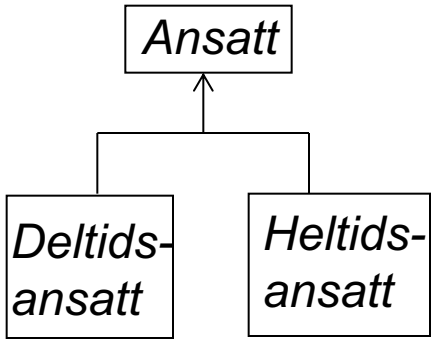


Abstrakte metoder og klasser

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

    public abstract double beregnBonus();
}
```

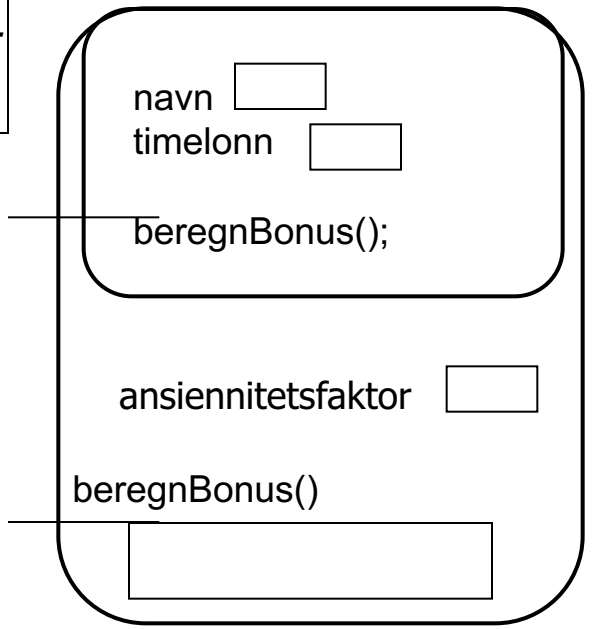
Ikke lov å si
new Ansatt() !






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

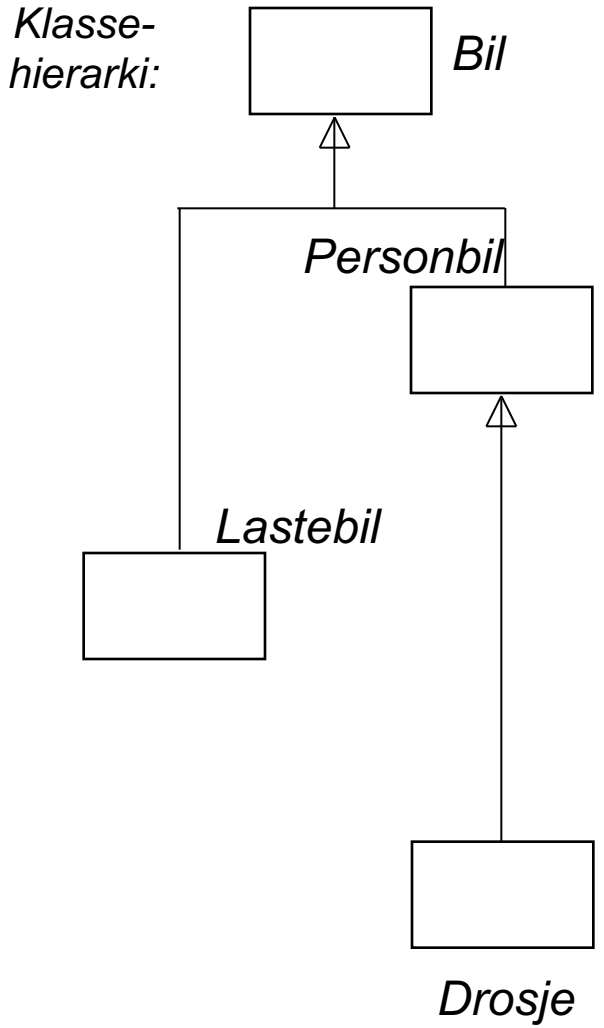
Heltidsansatt objekt



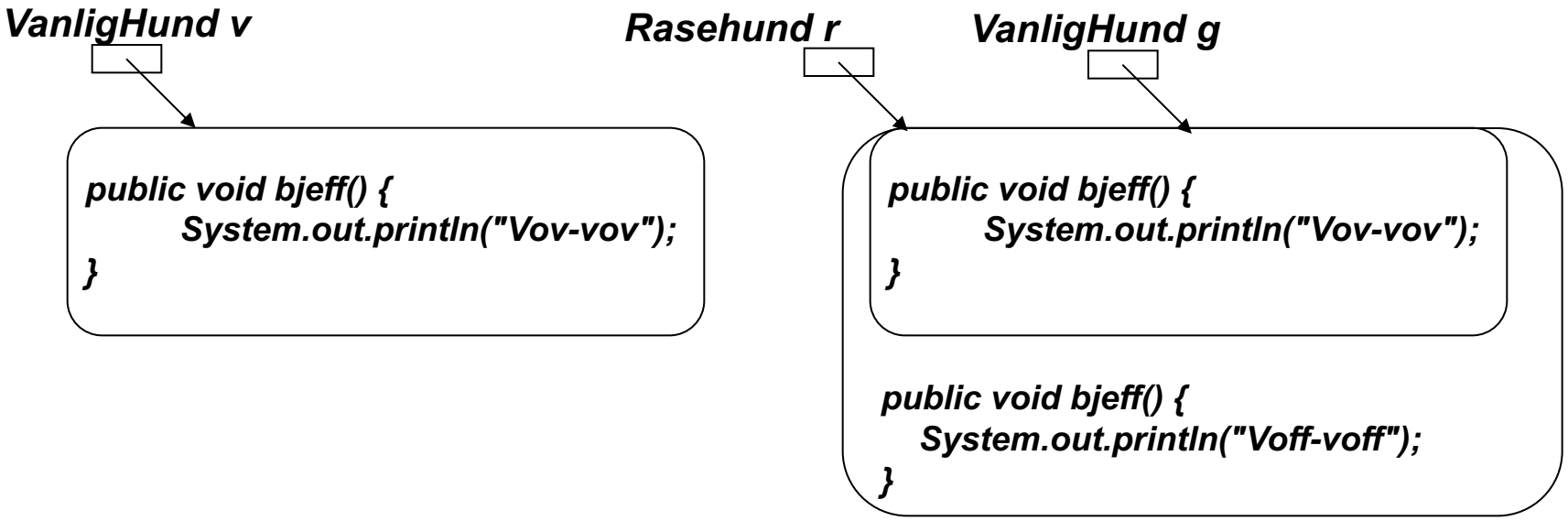

```
class Heltidsansatt extends Ansatt {
    protected int ansiennitetsfaktor;

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

Polymorfi: eksempel

```
class Bil {
    protected int pris;
    public int skatt( ) {return pris;}
}

class Personbil extends Bil {
    protected int antallPassasjer;
    public int skatt( ) {return pris * 2;}
}


class Lastebil extends Bil {
   protected double lastevekt;
   public int skatt ( ) {return pris / 2;}
}


class Drosje extends Personbil {
   protected String loyveId;
   public int skatt ( ) {return pris / 4;}
}
```



Polymorfi: eksempel



Anta dette programmet:

```
VanligHund v = new VanligHund();
Rasehund r = new Rasehund();
VanligHund g = r;
```

Hva skrives ut ved hvert av kallene:

`v.bjeff();`	Vov-vov
`r.bjeff();`	Voff-voff
`g.bjeff();`	Voff-voff

Nøkkelordet **super.**

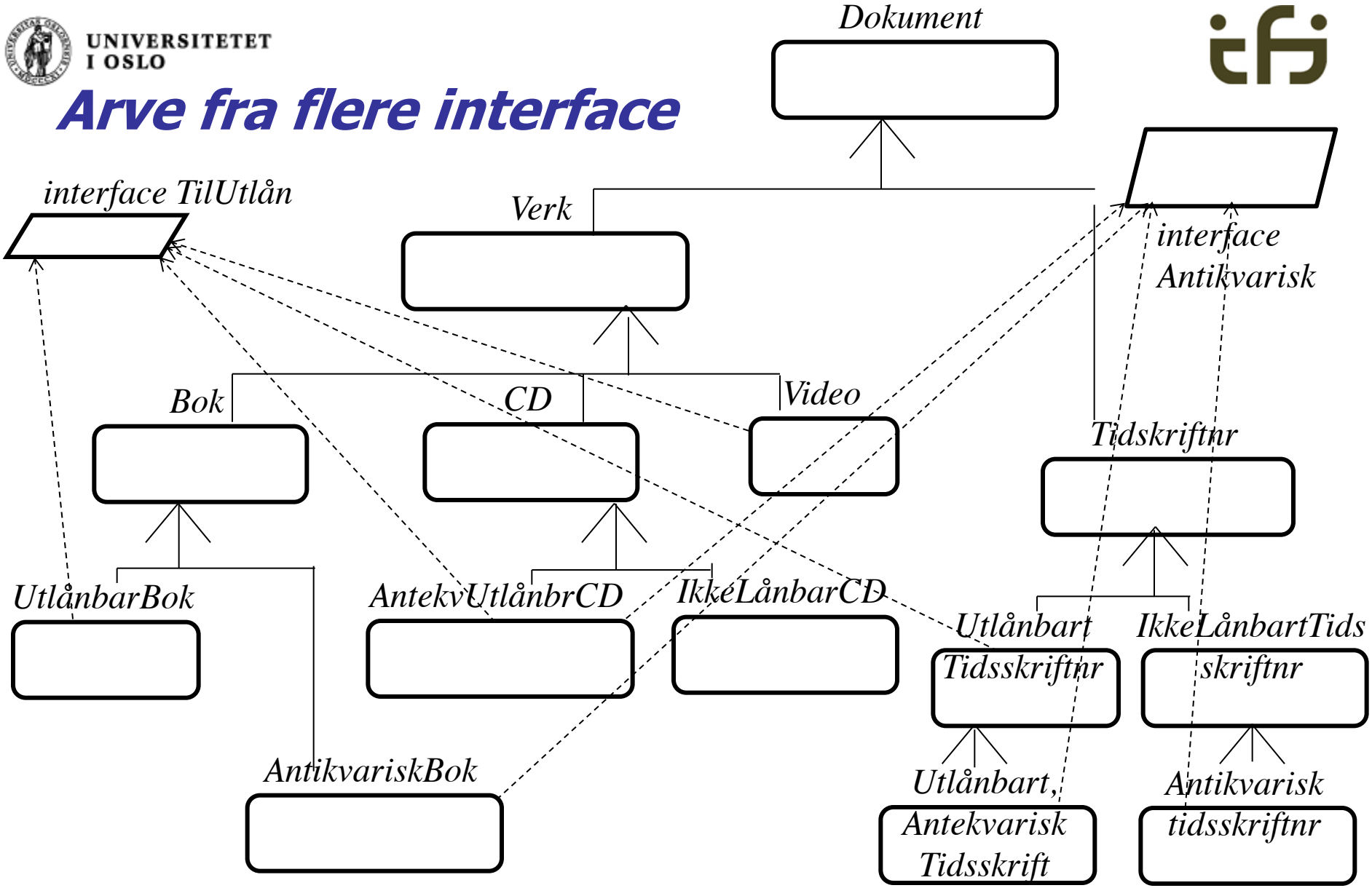
Nøkkelordet **super** brukes til å aksessere metoder i objektets superklasse. Dette kan vi bruke til å la superklassen Person ha en generell **skrivData**, som så kalles i subklassene:

```
// I klassen Person:
public void skrivData() {
        System.out.println("Navn: " + navn);
        System.out.println("Telefon: " + tlfnr);
}

// I klassen Student:
public void skrivData() {
        super.skrivData();
        System.out.println("Studieprogram: " + program);
}

// Tilsvarende i klassen Ansatt:
public void skrivData() {
        super.skrivData();
        System.out.println("Lønnstrinn: " + lønnstrinn);
        System.out.println("Timer: " + antallTimer);
}
```

Kall på super-konstruktøren

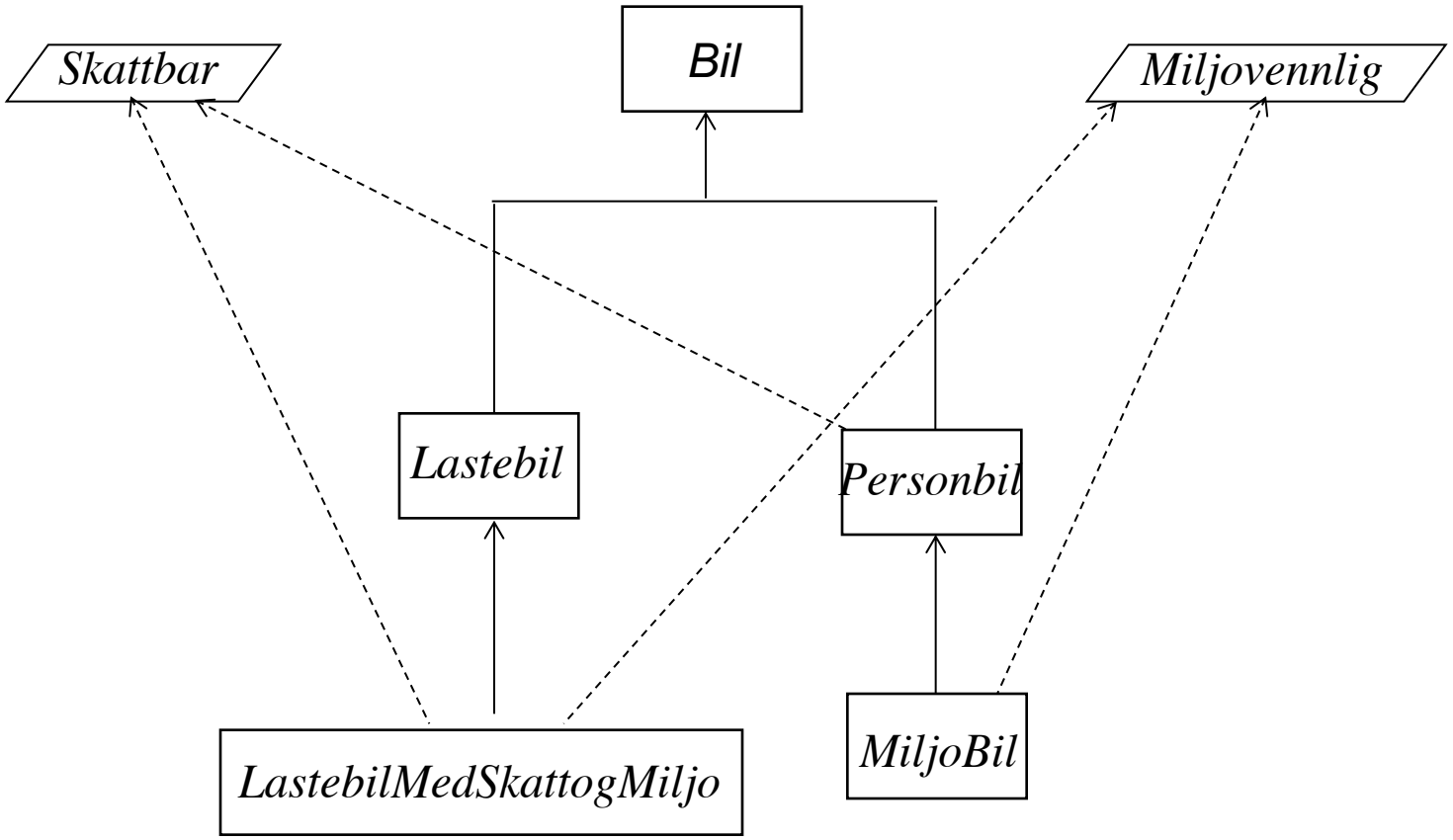
- Superklassens konstruktør kan kalles fra en subklasse ved å si:
  - `super();`
    - vil kalle på en konstruktør uten parametre
  - `super(5, "test");`
    - om vi vil kalle på en konstruktør med to parametre (int og String)
- Et kall på super **må** legges **helt i begynnelsen av** konstruktøren.
- Kaller man ikke super eksplisitt, vil Java **selv legge inn kall på super( )** helt først i konstruktøren når programmet kompileres.
- Hvis en klasse ikke har noen konstruktør, legger Java inn en tom konstruktør med kallet super();

Arve fra flere interface



- En klasse kan tilføres et ubegrenset antall interface-er
- Dvs. en klasse kan spille et ubegrenset antall roller
- Felles egenskaper på tvers av klassehierarkiet

Biler


Skattbar
Bil
Miljovennlig
Lastebil
Personbil
LastebilMedSkattogMiljo
MiljoBil

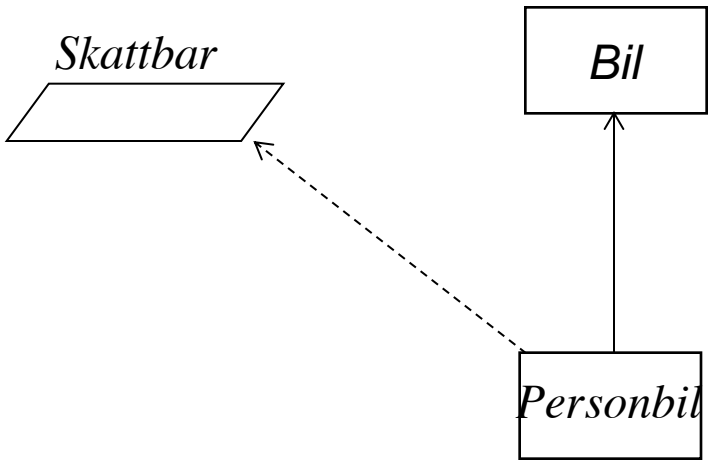
Implementere et interface: **implements**

rollen Bil (i arv) fra klassehierarkiet

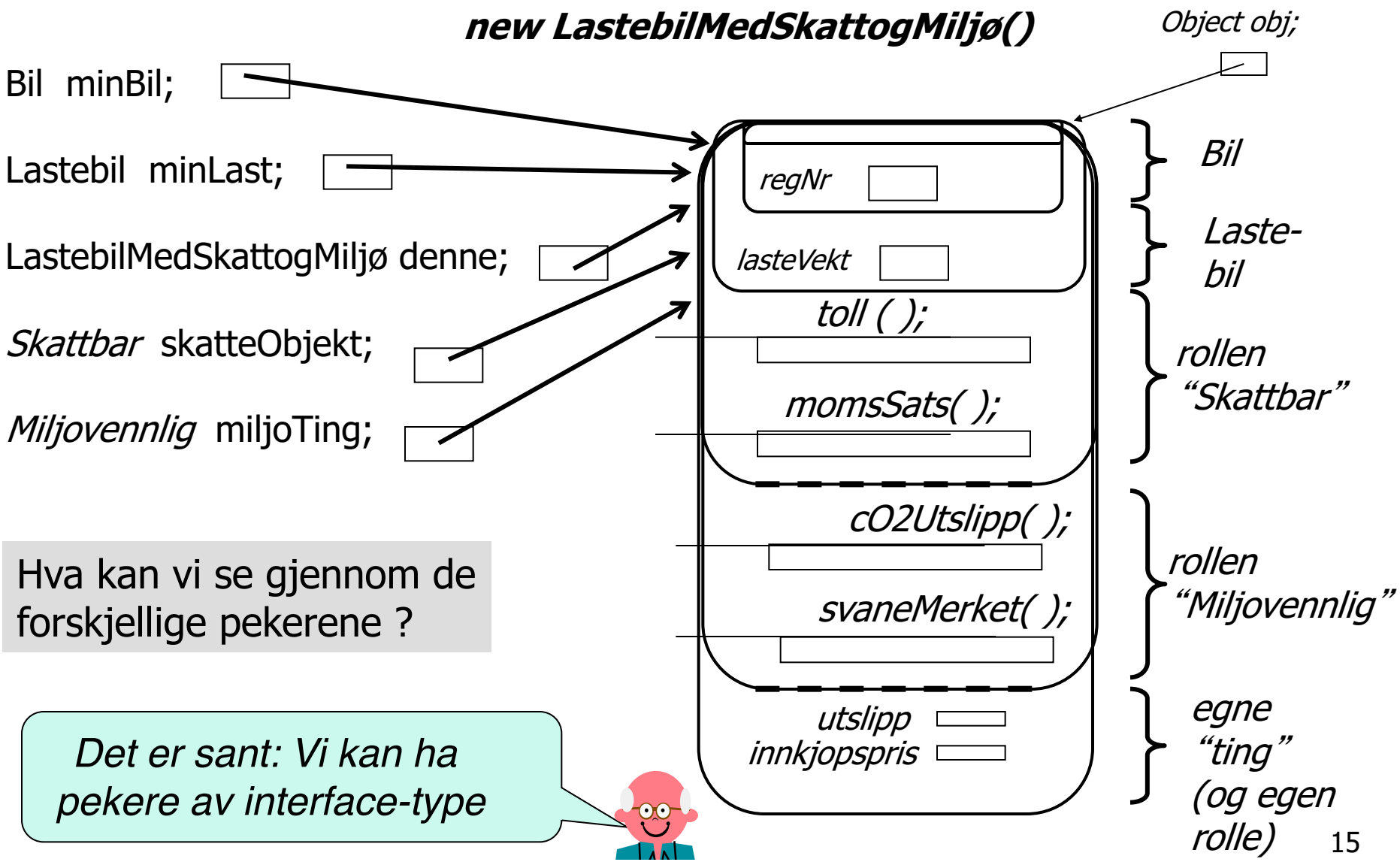
```
class Personbil extends Bil implements Skattbar {
    int antPass;
    double momsGrunnlag = 150000;
    public double toll( ){return momsGrunnlag*0.5;}
    public int momsSats( ){return 25;}
}
```

rollen "Skattbar"

```
class Bil {
    String regNr;
}

interface Skattbar {
    double  toll();
    int momsSats() ;
}
```



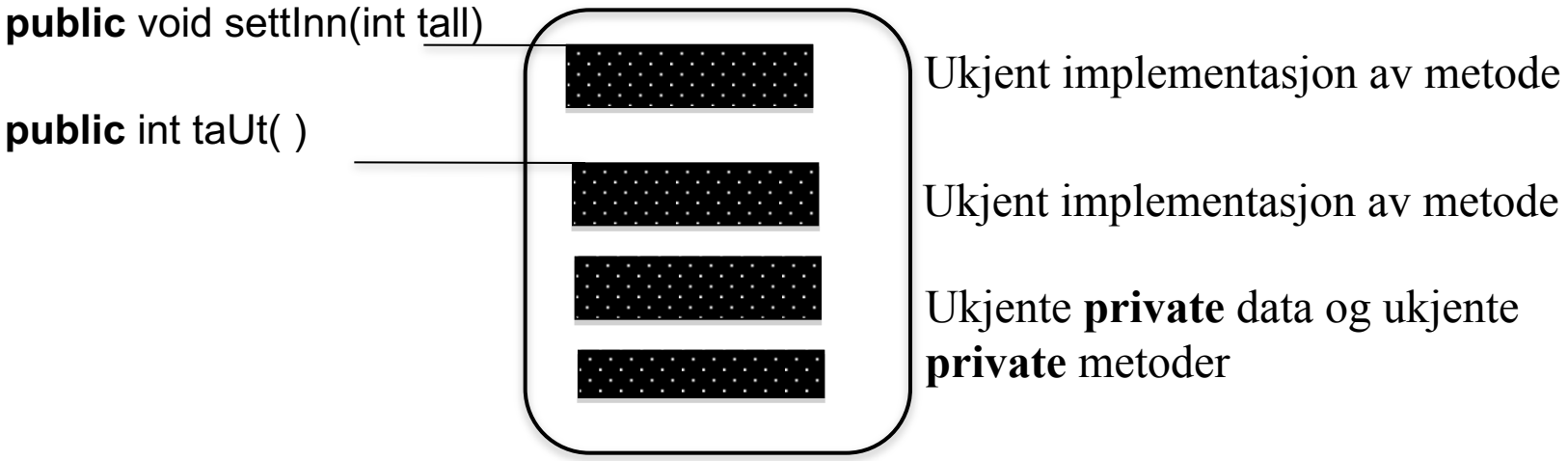
Et objekt og noen pekere



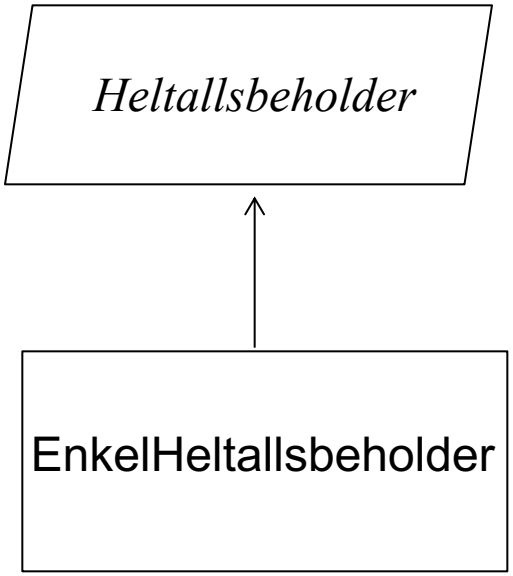
Hva kan vi se gjennom de forskjellige pekerene ?

Det er sant: Vi kan ha pekere av interface-type

Objektorientering handler om å tydeliggjøre objektenes public-metoder. Interface gjør det.



Interface: Klassehierarki og Java-kode

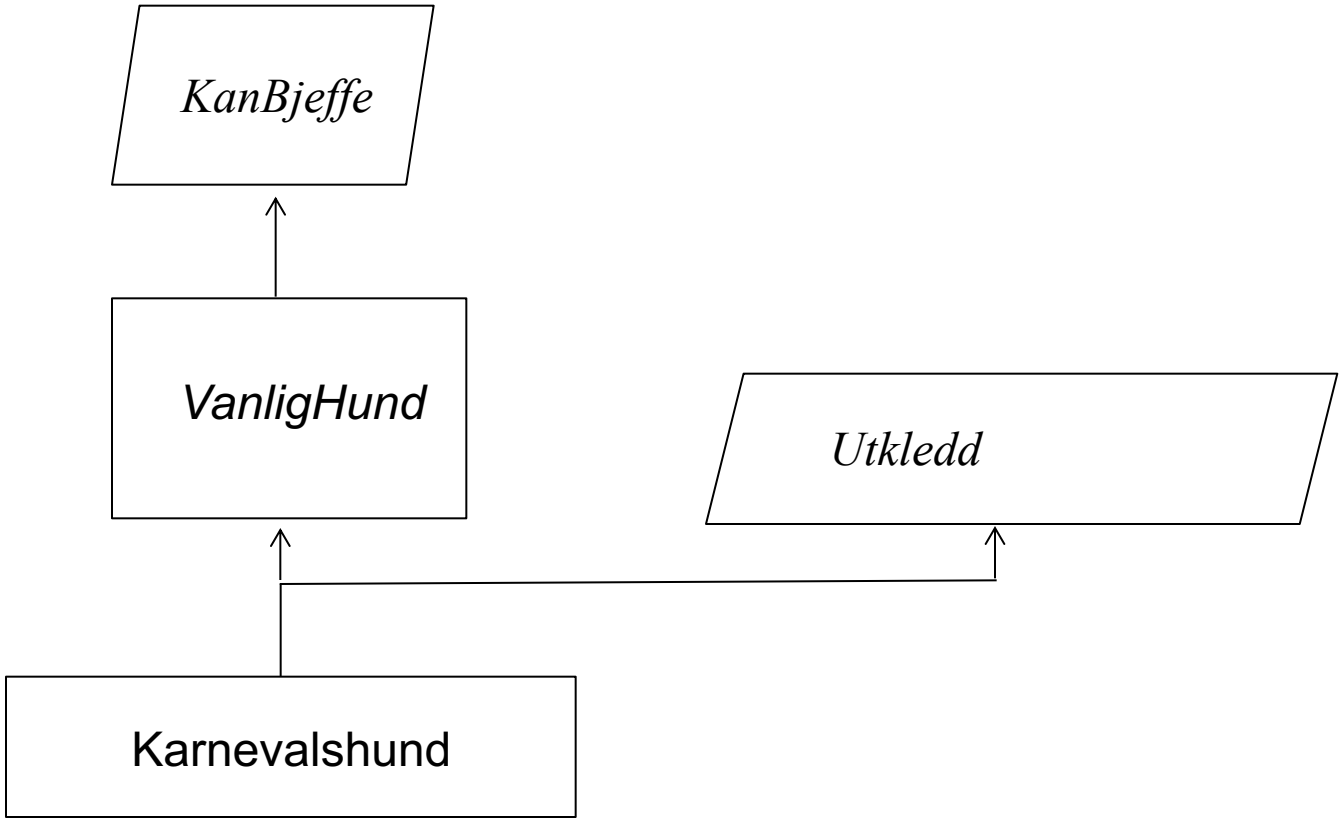



```
interface Heltallsbeholder {
        public void settInn(int tall);
        public int taUt( );
}

class EnkelHeltallsbeholder
        implements Heltallsbeholder {
    protected int [ ] tallene = new int [100];
    protected int antall;
    public void settInn(int tall) { . . . }
    public int taUt( ) { . . . }
}
```

Når en klasse implementerer et interface tegner vi det nesten på samme måte som en superklasse / subklasse. For å markere at "superklassen" ikke er det, men et interface, kan vi enten skrive "interface" i boksen, og/eller vi kan gjøre navnet på interfacet (og boksen ?) kursiv.

Engelsk: Interface ***Norsk: Grensesnitt***

Karnevalshund



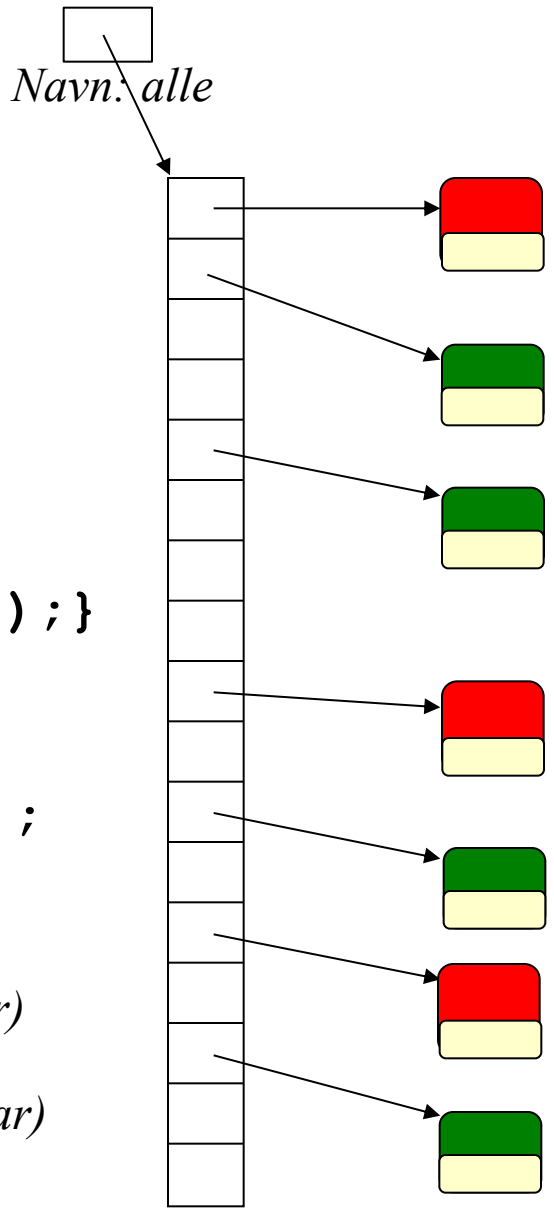
Denne figuren avspeiler
”interface”-ene og ”class”-ene på neste siden

Foto: AP

```
interface KanBjeffe{
        void bjeff();
}

interface Utkledd {
        int antallFarger();
}

class VanligHund implements KanBjeffe {
       public void bjeff() {
              System.out.println("Vov-vov");
       }
}

class Karnevalshund extends VanligHund implements Utkledd {
       private boolean farger;
       public Karnevallshund (int frg) {
              farger = frg;
       }
       public boolean antallFarger() {
              return farger;
       }
}
```

Samlet import-skatt

```
Skattbar[ ] alle = new Skattbar [100];
alle[0] = new Bil(”DK12345”, 150000);
alle[1] = new Ost(20,5000);
. . .
. . .
int totalSkatt = 0;

for (Skattbar den: alle) {
   if (den != null)
    {totalSkatt = totalSkatt + den.skatt();}
}

System.out.println(”Total skatt: ” +
                                totalSkatt);
```



Rollen Skattbar

Rollen Bil (untatt Skattbar)

Rollen Ost (untatt Skattbar)

Veldig viktig og bra eksempel. Dagens rosin.

Generiske interface
Inteface med parametre

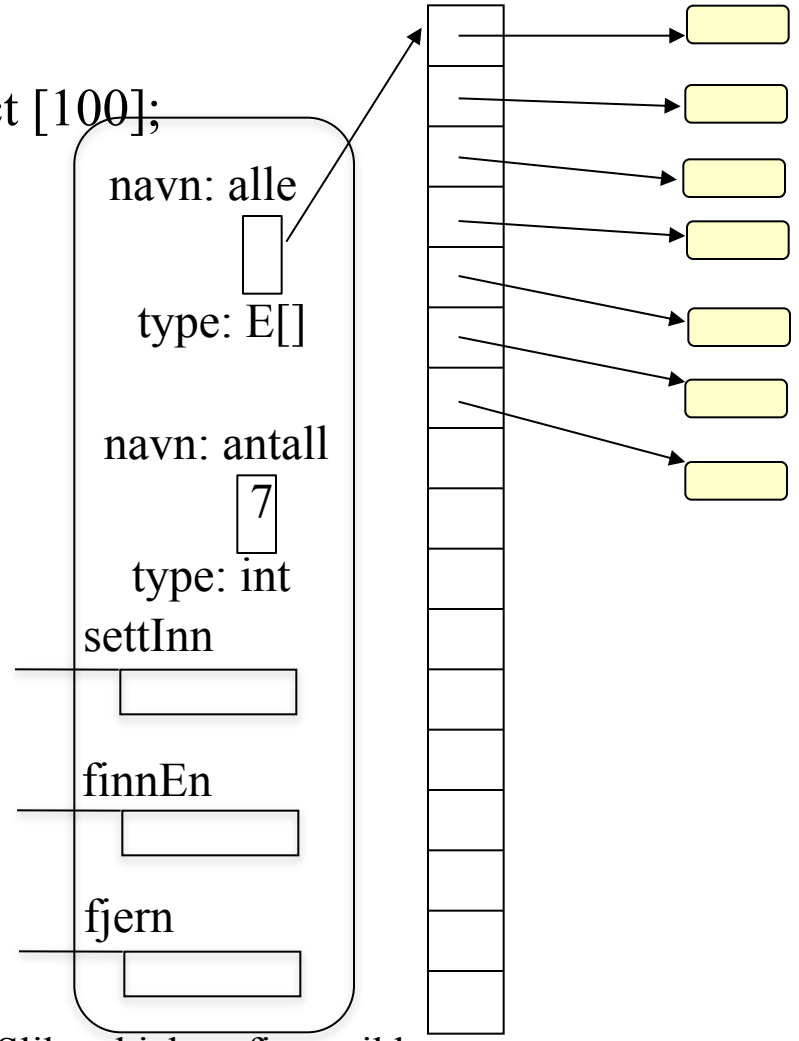
- På samme måte som klasser, kan interface lages med parametre.
- class GeneriskBeholderTilEn <E> { . . . }
- *interface Beholder <E> { . . . }*

new GeneriskBeholderTilEn<Bil>( );

~~*new Beholder<Bil>( );*~~

```
interface Beholder <T> {
        public void settInn (T  en);
        public T  taUt ( );
}


class GeneriskBeholderTilEn <T> implements  Beholder <T> {
        T   denne;
        public void settInn (T  en) { denne = en;}
        public T  taUt ( ) {return denne;}
}
```

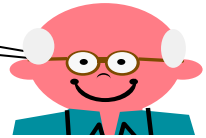
```
class KonkretEnkelStorBeholder <E>
                    implements EnkelStorBeholder<E>
{
        private E [ ] alle = (E [ ] ) new Object [100];
        private int antall = 0;
        public boolean settInn(E  elem) {
                . . .
                . . .
                . . .
                . . .
        }
        public E finnEn( ) {
                . . .
                . . .
        }
        public void  fjern( ) {
                . . .
        }
}
```



Slike objekter finnes ikke
(må gi E en verdi først)

Konklusjon interface

*Hva brukes **interface** til?*



Vet hjelp av interface kan forskjellige klasser og objekter ha det samme grensesnittet. Dette er en fordel når vi skal beskrive objekter med felles egenskaper.

Et interface kalles gjerne også en **rolle** (som en subklasse)

- Noen objekter kan spille flere forskjellig roller (snart: multippel arv)
- Forskjellige objekter kan implementere samme rolle på forskjellige måter
  - innkapsling = skjuling av detaljer