



UNIVERSITETET  
I OSLO



Institutt for informatikk

# IN1010 våren 2020

Onsdag 12. februar

## Interface - Grensesnitt

Stein Gjessing

Institutt for informatikk

## Dagens hovedtema

- Engelsk: Interface (også et Java-ord)
- Norsk: Grensesnitt
  
- Les notatet “Grensesnitt i Java” av Stein Gjessing
  
- To motivasjoner for interface
  - 1) Tydeliggjøre klassens public-metoder
  - 2) Multippel arv

## Multippel arv: Om å arve fra flere

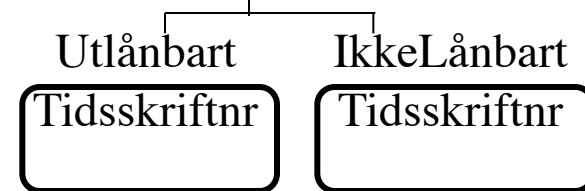
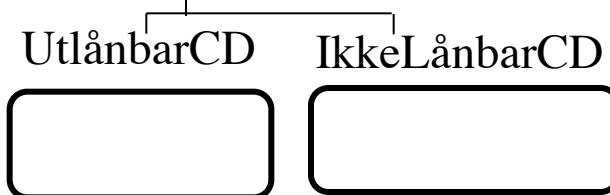
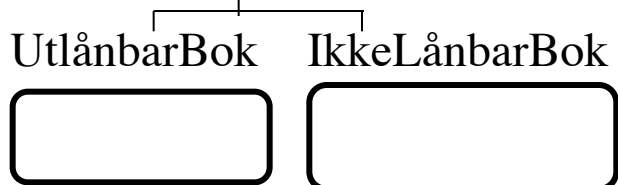
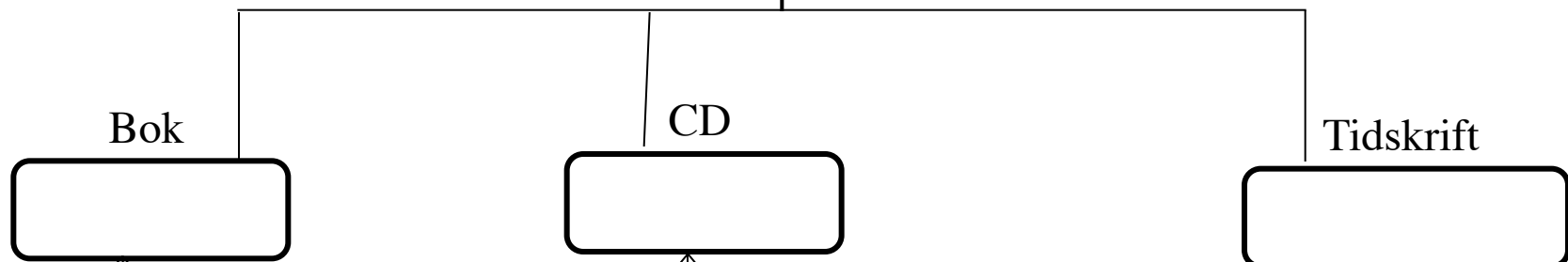
- I Java kan en klasse bare arve egenskapene til **én** annen klasse (en superklasse).
  - Dette gjør språket sikrere å bruke
- Hva skal vi gjøre hvis vi ønsker at et objekt skal inneholde mange forskjellige egenskaper fra forskjellige “superklasser” ?
- På de neste sidene:
  - Begrepshierarkiet i et bibliotek



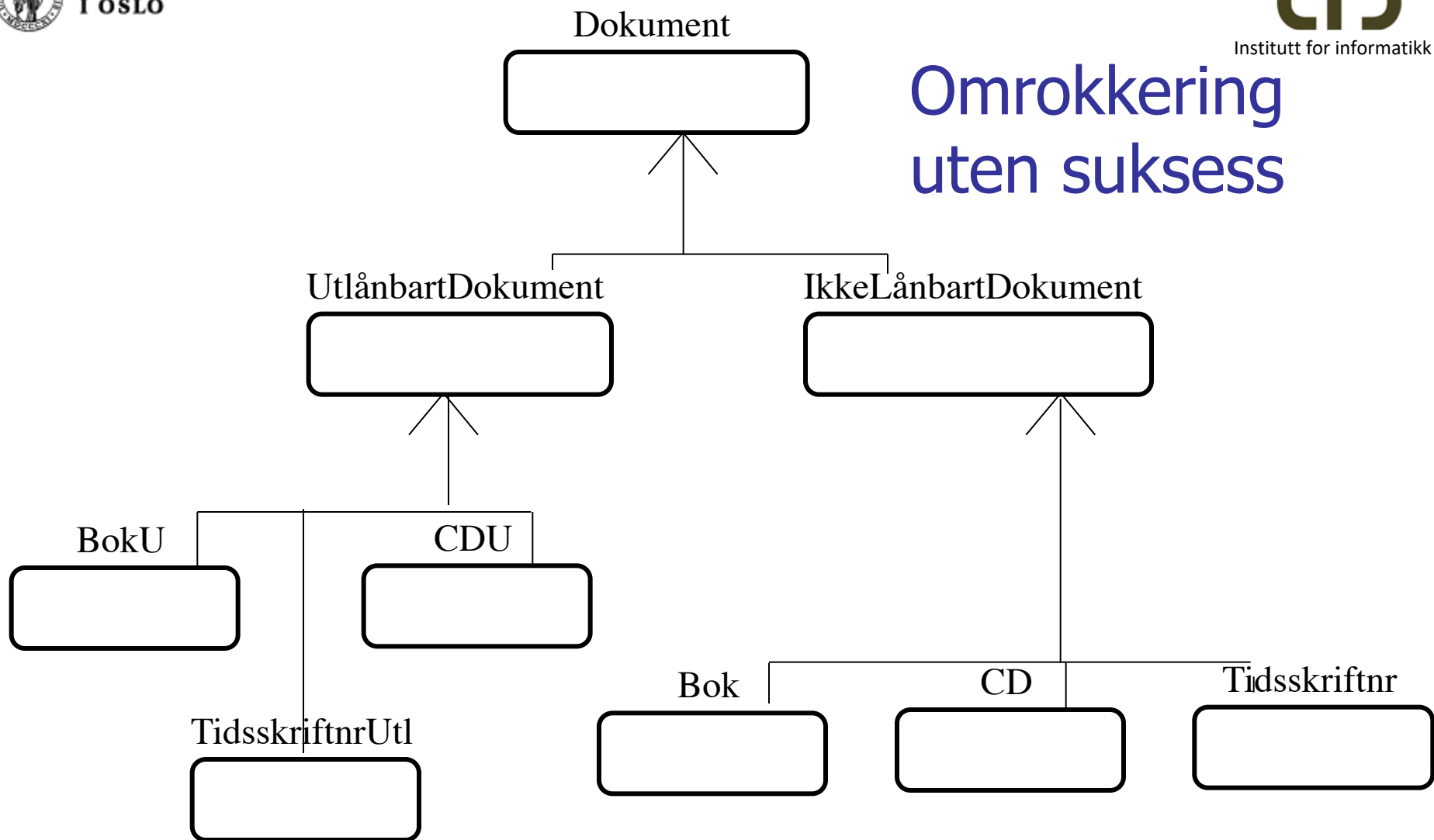
# Motivasjon for begrepet interface: Analyse av bibliotek

- Bøker, tidsskrifter, CDer, videoer, mikrofilmet materiale, antikvariske bøker, flerbindsverk, oppslagsverk, upubliserte skrifter, ...
- En del felles egenskaper
  - antall eksemplarer, hylleplass, identifikasjonskode (Dokument)
  - for det som kan lånes ut: Er utlånt ? , navn på låner, ... (TilUtlån)
  - for det som er antikvarisk: Verdi, forskringssum, ... (Antikvarisk)
- Spesielle egenskaper:
  - Bok: Forfatter, tittel, forlag
  - Tidsskriftnummer: Årgang, nummer, utgiver
  - CD: Tittel, artist, komponist, musikkforlag

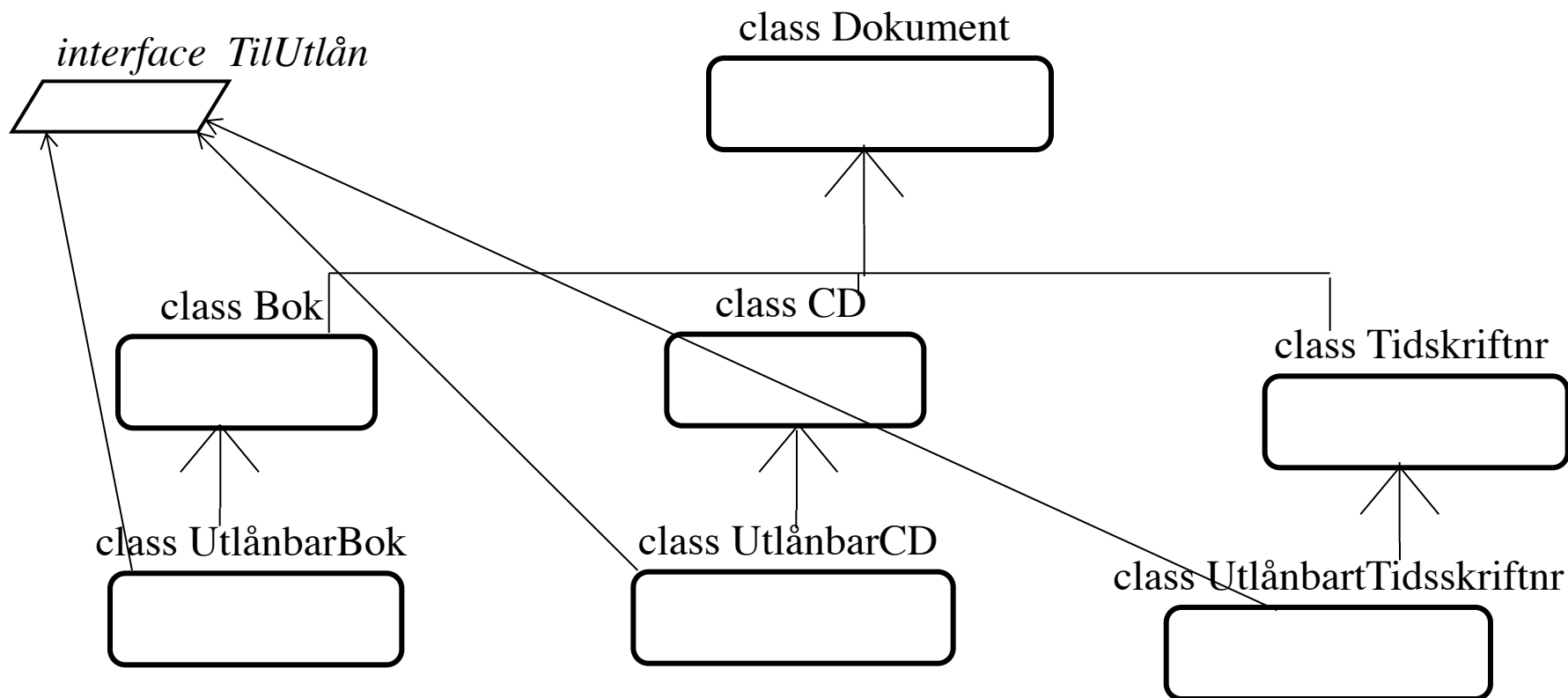
# Tvilsomt begrepshierarki



# Omrokking uten suksess



# Samle lik oppførsel: bruk **interface**



- En klasse kan tilføres et (eller flere) interface
  - i tillegg til å arve egenskapene i klassehierarkiet
- Dvs. en klasse kan spille to (eller flere) **roller**



## Et interface (grensesnitt) er:

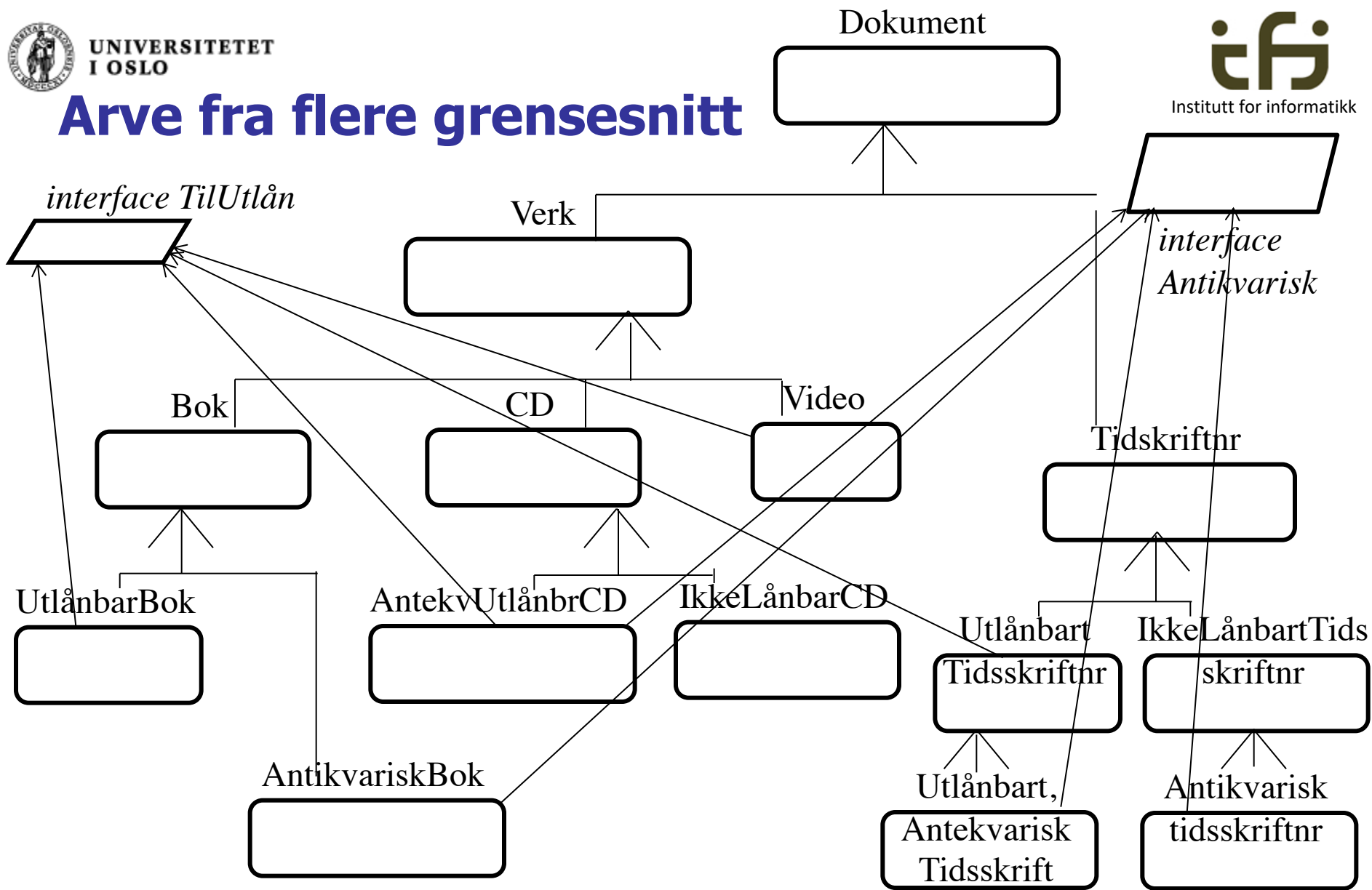
- En samling egenskaper (en rolle) som ikke naturlig hører hjemme i et arve-hierarki
- En samling egenskaper som mange forskjellige “ting” av forskjellige typer kan anta
- En klasse kan arve egenskapene til mange grensesnitt (men bare en klasse)
- For eksempel
  - Kan delta i konkurranse (startnummer, resultat, ..  
Mennesker, biler, hester kan delta i konkurranser)
  - Svømmedyktig (mennesker, fugler er svømmedyktige)
  - Her: Antikvarisk (møbler, bøker, .... )  
Kan lånes ut (biler, bøker, festklær, ... )
  - Sammenlignbar (Comparable)
  - ....



## Hva er et interface ?

- Et interface ligner en abstrakt klasse
- **Alle** metodene i et interface er abstrakte og polymorfe
- En interface inneholder **ingen** variable eller annen datastruktur (men litt annet som vi ikke bruker i IN1010)
- En klasse som arver egenskapene til et interface må selv putte inn kode i alle de abstrakte metodene (og deklare passende variable som disse metodene bruker for å gjøre jobben sin).
- En klasse kan arve egenskapene til mange grensesnitt (men bare en klasse)
  
- Å arve (en samling metoder) = å spille en rolle

# Arve fra flere grensesnitt



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

# Nytt interface-eksempel

(Vi kommer tilbake til biblioteket senere)

Hvis vi ønsker at noen objekter også skal kunne spille rollene (ha / arve egenskapene) “**Skattbar**” (en ting vi må skatte av) og “**Miljøvennlig**” (en ting som er miljøvennelig) kan vi ha:

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

interface istedenfor class

; istedenfor innmat i  
metodene

```
interface Miljøvennlig {  
    int cO2Utslipp ();  
    boolean svaneMerket ();  
}
```

Nytt Java nøkkelord:  
**interface**



# Vi tegner et interface slik

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



```
interface Miljovennlig {  
    int cO2Utslipp ();  
    boolean svaneMerket ();  
}
```



# Enkelt eksempel med bil-hierarkiet

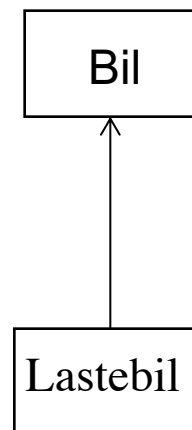
```
class Bil { protected String regNr; }  
class Lastebil extends Bil {  
    protected double lasteVekt;  
}
```

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

*Skattbar*

```
interface Miljovennlig {  
    int cO2Utslipp ( ) ;  
    boolean svaneMerket ( ) ;  
}
```

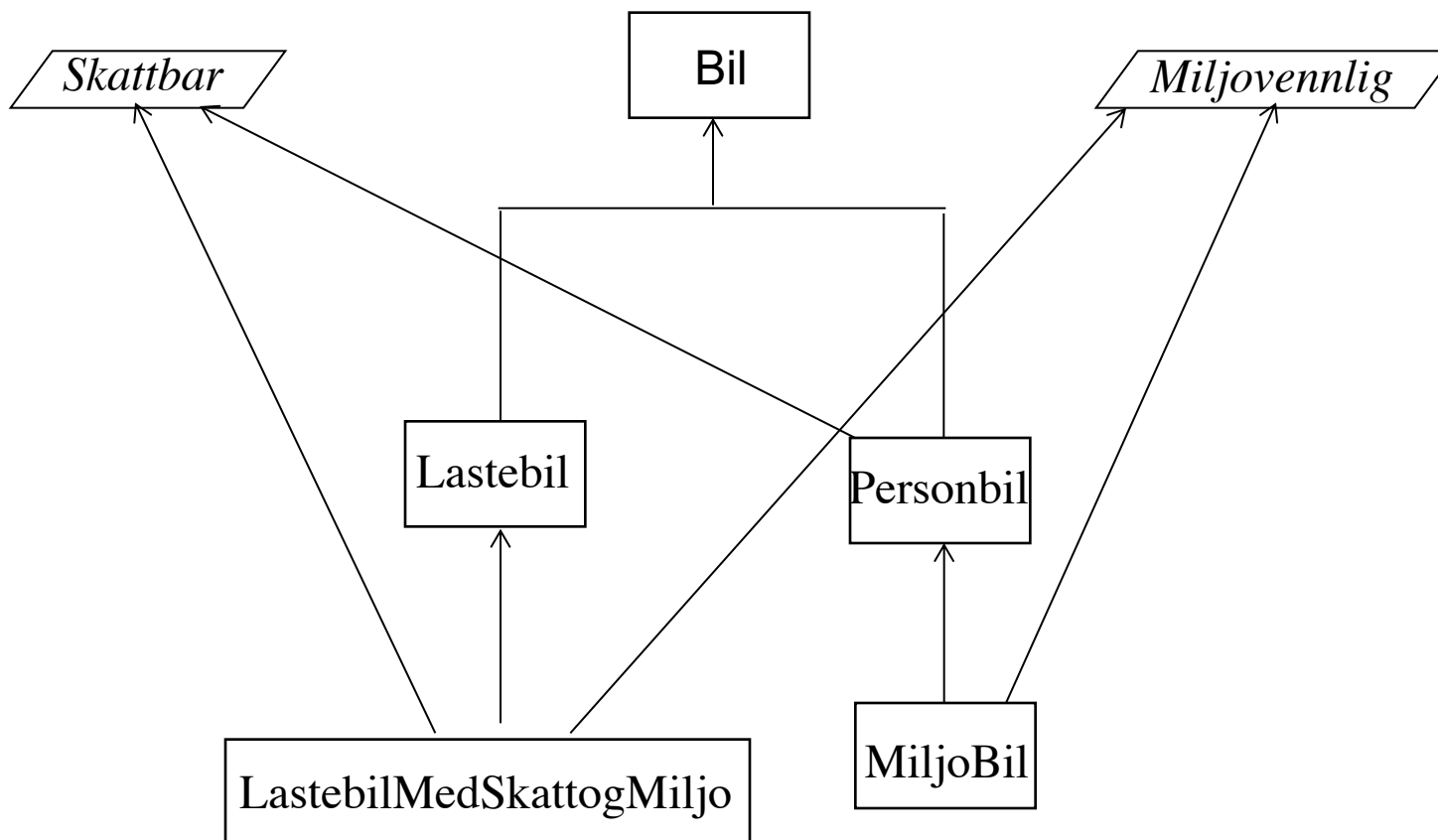
*Miljovennlig*



Metodene i et grensesnitt er veldig "abstrakte"



## Tre nye klasser som kan spille mange roller



Men metodene må (dessverre) skrives på nytt hver gang de brukes



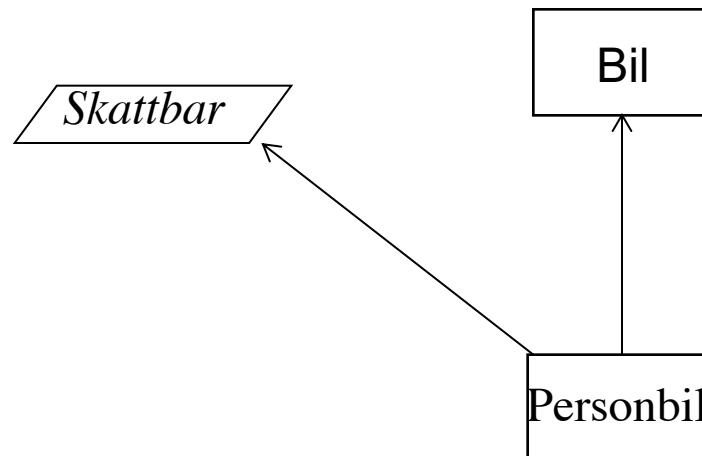
# Rerservert Java-ord: implements (1)

rollen Bil (i arv) fra klassehierarkiet

```
class Personbil extends Bil implements Skattbar {  
    protected int antPass = 5;  
    protected double momsGrunnlag = 150000;  
    @Override  
    public double toll( ) {return momsGrunnlag*1.0;}  
    @Override  
    public int momsSats( ) {return 25;}  
}
```

rollen  
"Skattbar"

```
class Bil {  
    String regNr;  
}  
  
interface Skattbar {  
    double toll();  
    int momsSats();  
}
```





Rollene i (arv fra) klassehierarkiet

```
class MiljoBil extends Personbil implements Miljovennlig {  
    protected int utslipp = 100;  
    @Override  
    public int cO2Utslipp(){return utslipp;}  
    @Override  
    public boolean svaneMerket(){return false;}  
}
```

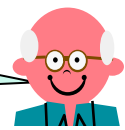
} rollen "Miljovennlig"

```
class LastebilMedSkattogMiljo extends Lastebil implements Skattbar,  
Miljovennlig {  
    protected double innkjopspris = 200000;  
    protected int utslipp = 400;  
    @Override  
    public double toll(){return innkjopspris*0.1;}  
    @Override  
    public int momsSats(){return 25;}  
    @Override  
    public int cO2Utslipp(){return utslipp;}  
    @Override  
    public boolean svaneMerket(){return false;}  
}
```

} rollen "Skattbar"

} rollen "Miljovennlig"

MiljoBil arver rollen Skattbar fra Personbil





# Et objekt og noen pekere

**new LastebilMedSkattogMiljø()**

Bil minBil;

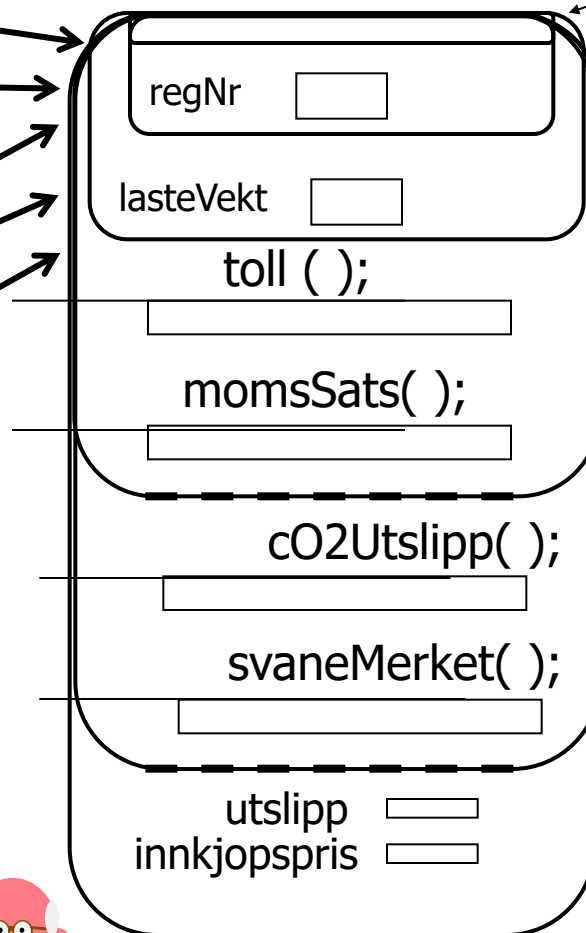
Lastebil minLast;

LastebilMedSkattogMiljø denne;

Skattbar skatteObjekt;

Miljøvennlig miljøTing;

Object obj;



Hva kan vi se gjennom de forskjellige pekerene ?

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



# Mer om grensesnitt (interface)

- Navnet på et interface kan brukes som typenavn når vi lager referanser (det så vi på forrige side)
- Vitsen med et interface er å spesifisere **hva** som skal gjøres (ikke hvordan)
- Vanligvis er det flere implementasjoner av et interface (flere klasser implementerer det).
- Vi vet: En klasse kan implementere (flere) interface samtidig som klassen også er subklasse av (bare) én annen klasse.
- En implementasjon (av et interface) skal kunne endres uten at resten av programmet behøver å endres.
  
- Vi har en ny type som er interfacenavn:  
«Skattbar» og «Miljøvennlig» er referansetyper

# Grensesnitt (interface) lærdom

- Et interface har bare
  - metodenavn med parametre, men ikke kode (husk ;)
- Bruker 'interface' i steden for 'class' før navnet
- Definerer en 'type' / 'rolle' som andre må implementere
- Meget nyttig, brukes mye ved distribuerte systemer og generelle programbiblioteker som Javas eget
- Ulempe: Koden/implementasjonen må gjøres mange ganger
- **Mer generelt kjent under navnet ADT =Abstrakt Data Type** ,  
Vi definerer ***hva*** en ny datatype skal gjøre, ***ikke hvordan*** dette gjøres.
  - Det kan være mange mulige implementasjoner (=måter å skrive kode på) som lager en slik datatype.
  - Hva som er beste implementasjon må avgjøres etter hvilken bruk vi har.



# Ekstra eksempler: Mer om Biler og Lastebiler: Legg til metoder for å skrive ut på skjerm:

```
class Bil {  
    protected String regNr;  
    public void skriv(){  
        System.out.println("Registreringsnummer: " + regNr);  
    }  
}
```

```
class Lastebil extends Bil {  
    double lasteVekt;  
    @Override  
    public void skriv () {  
        super.skriv();  
        System.out.println("Lastevekt: " + lasteVekt);  
    }  
}
```



```

class LastebilMedSkattOgMiljo extends Lastebil implements Skattbar, Miljovennlig {
    protected double innkjopspris = 200000;
    protected int utslipp = 400;
    @Override
    public double toll() { return innkjopspris * 0.1; }
    @Override
    public int momsSats() {return 20;}
    public void skrivSkatt()
        { System.out.println("Innkjøpspris " + innkjopspris); }
    @Override
    public int cO2Utslipp () {return utslipp; }
    @Override
    public boolean svaneMerket () { return false; }
    public void skrivMiljo() { System.out.println("Utslipp " + utslipp); }
    @Override
    public void skriv() {
        System.out.println("Lastebil med skatt og miljø: ");
        super.skriv(); skrivSkatt(); skrivMiljo();
    }
}

```

(Som før)

(Skattbar og Miljovennlig som før)

Det er ikke naturlig at Skatt og Miljo skal **kreve** en "skriv"-metode (?)



# Skriv i LastebilMedSkattogMiljo

Object obj;

## LastebilMedSkattogMiljø-objekt

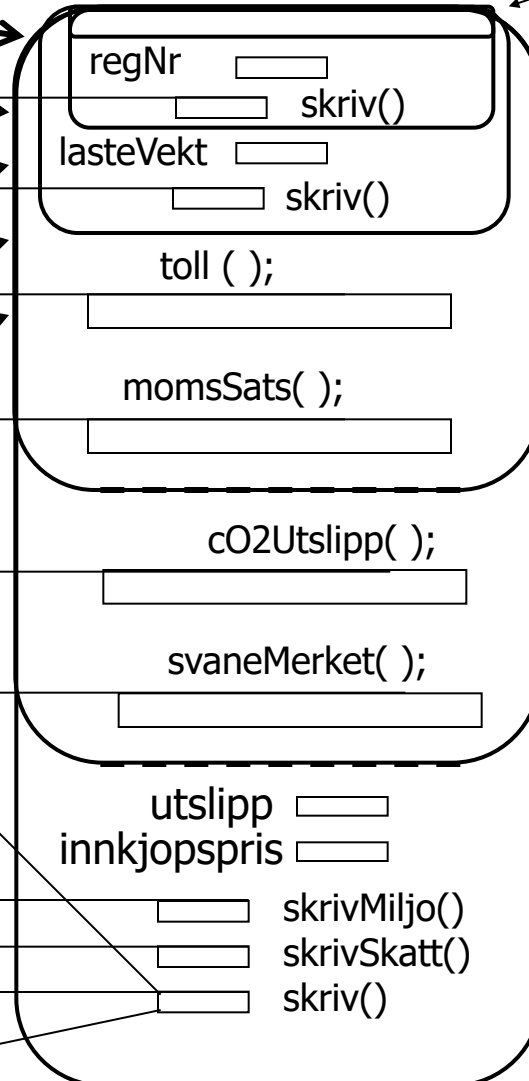
Bil minBil;

Lastebil minLast;

LastebilMedSkattogMiljø denne;

Skattbar skatteObjekt;

Miljøvennlig miljøTing;



```

public void skriv( ) {
    System.out.println
        ("Lastebil med skatt og miljø: ");
    super.skriv( );
    skrivSkatt();
    skrivMiljo();
}
  
```

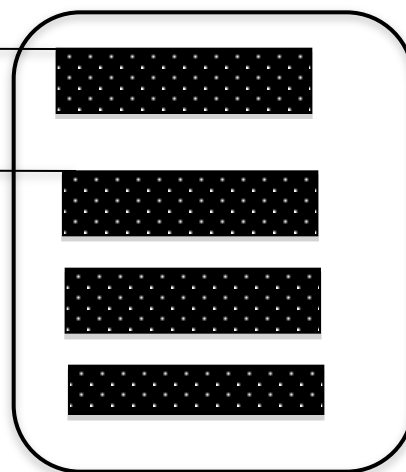
(egne  
"ting" og)  
**egen**  
**rolle**

# Objektorientering handler om å tydeliggjøre objektene public-metoder

Husker dere forelesingen om enhetstesting:

**public** void settInn(int tall)

**public** int taUt( )



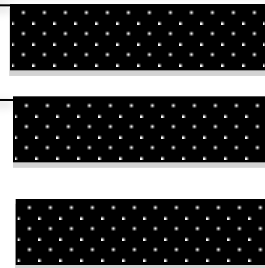
Ukjent implementasjon av metode

Ukjent implementasjon av metode

Ukjente **private** data og ukjente **private** metoder

```
public void settInn(int tall)
```

```
public int taUt( )
```



*Med Java koden under kan vi senere lage objekter med slik oppførselen*

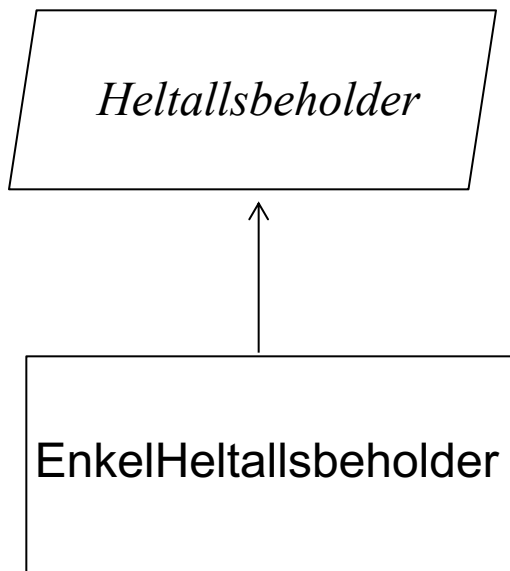
```
interface Heltallsbeholder {  
    public void settInn(int tall);  
    public int taUt( );  
}
```

Java  
kode

~~new Heltallsbeholder()~~



# Interface: Klassehierarki og Java-kode



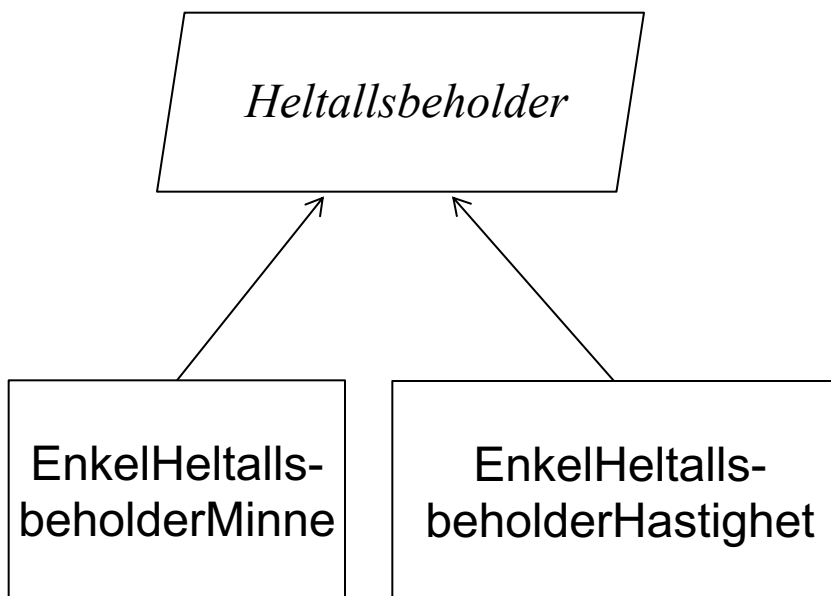
```
interface Heltallsbeholder {
    public void settInn(int tall);
    public int taUt( );
}
```

```
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 boksen og navnet på interfacet kursiv.

# VIKTIG: Ett interface

## Flere forskjellige implementasjoner



”Ikke-funksjonelle” forskjeller på implementasjonene. For eksempel hastighet, minnebruk, . . .



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

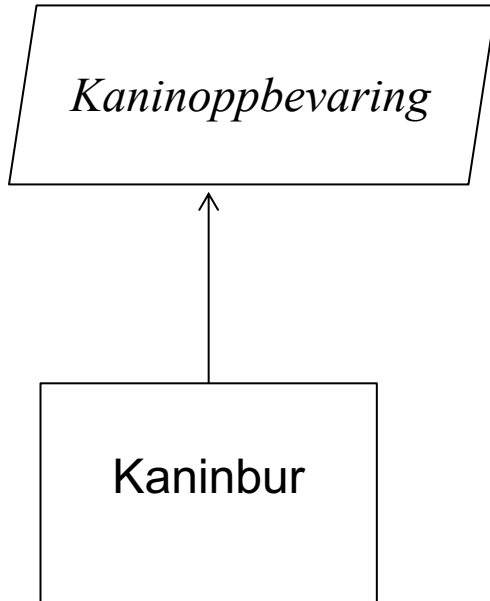
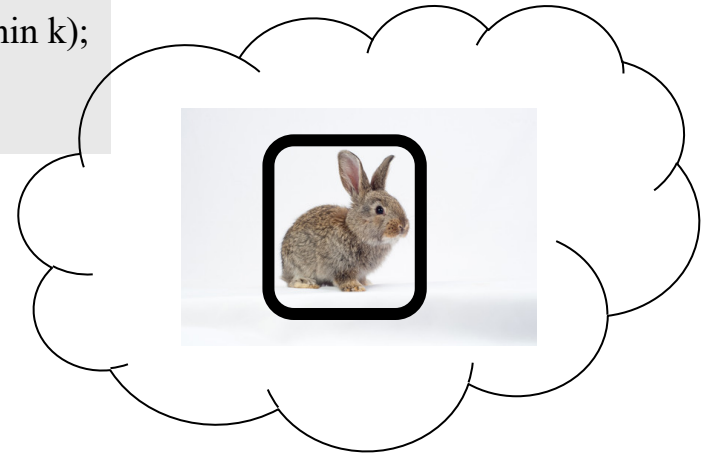
```
class EnkelHeltallsbeholderMinne
    implements Heltallsbeholder {
    protected int [ ] tallene = new int [100];
    protected int antall;
    public void settInn(int tall) { . . . }
    public int taUt( ) { . . . }
}
```

```
class EnkelHeltallsbeholderHastighet
    implements Heltallsbeholder {
    protected ArrayList . . . . .

    public void settInn(int tall) { . . . }
    public int taUt( ) { . . . }
}
```

# Vi kan se på et kaninbur som et sted for kaninoppbevaring

```
interface KaninOppbevaring {  
    public boolean settInn(Kanin k);  
    public Kanin taUt();  
}
```



# Kaninbur som et sted for kaninoppbevaring

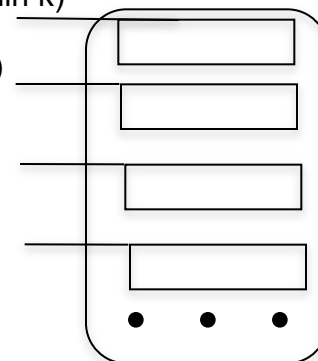
```
interface KaninOppbevaring {  
    public boolean settInn(Kanin k);  
    public Kanin taUt( );  
}
```

```
class Kaninbur implements KaninOppbevaring {  
    private Kanin denne = null;  
    @Override  
    public boolean settInn(Kanin k) {  
        ...  
    }  
    @Override  
    public Kanin taUt( ) {  
        ...  
    }  
}
```



```
public boolean settInn(Kanin k)
```

```
public Kanin taUt( )
```



Et objekt av en klasse som  
implementerer grensesnittet  
KaninOppbevaring

```
class Kanin {  
    private String navn;  
    public Kanin(String nv) {navn = nv;}  
}
```

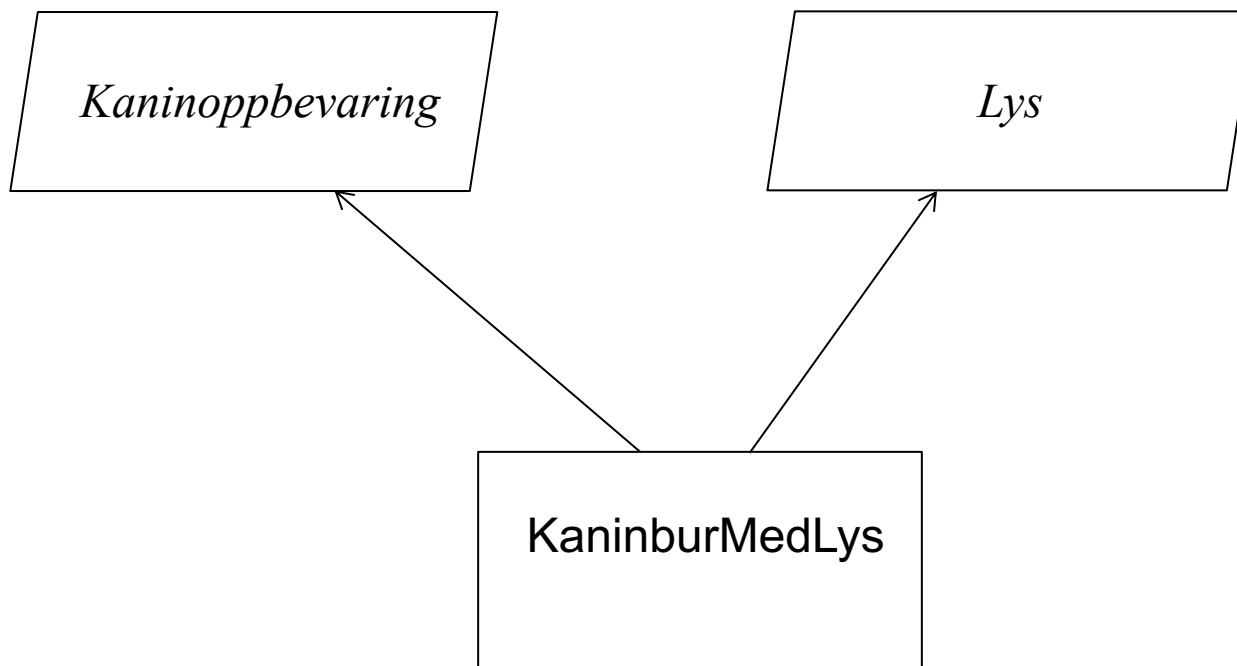
```
interface KaninOppbevaring {  
    public boolean settInn(Kanin k);  
    public Kanin taUt();  
}
```



```
class Kaninbur implements KaninOppbevaring {  
    private Kanin denne = null;  
    @Override  
    public boolean settInn(Kanin k) {  
        if (denne == null) {  
            denne = k;  
            return true;  
        }  
        else return false;  
    }  
    @Override  
    public Kanin taUt() {  
        Kanin k = denne;  
        denne = null;  
        return k;  
    }  
}
```

# KaninburMedLys

Når en kanin vil ha lys på om natten:





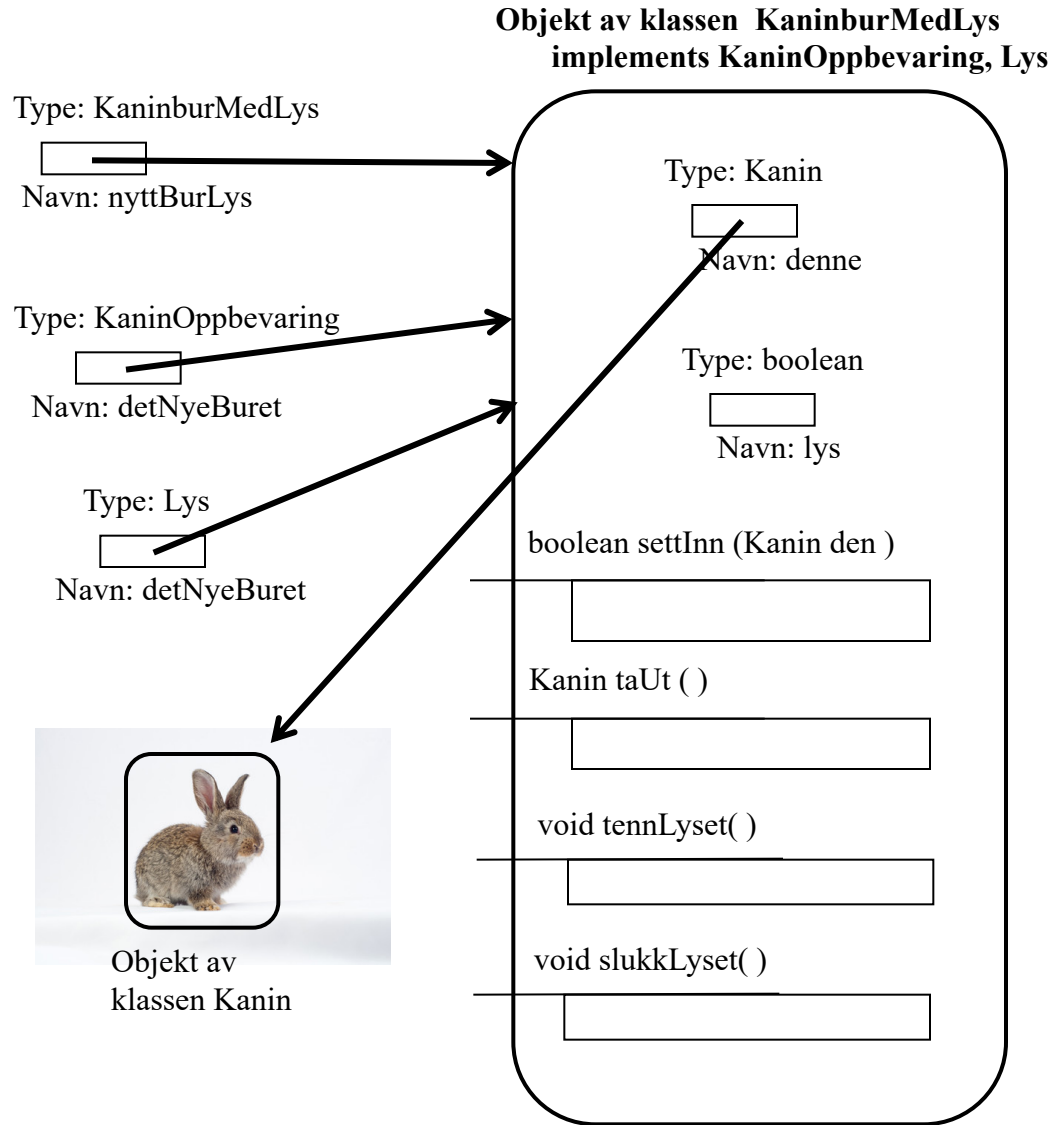
## Vi kan lage kassen KaninburMedLys på denne måten: Én klasse – to grensesnitt

```
class KaninburMedLys implements KaninOppbevaring, Lys {  
    private boolean lys = false;  
    private Kanin denne = null;  
    @Override  
    public boolean settInn(Kanin k) {  
        ...  
        ...  
    }  
    @Override  
    public Kanin taUt( ) {  
        ...  
        ...  
    }  
    @Override  
    public void tennLyset ( ) {lys = true;}  
    @Override  
    public void slukkLyset ( ) {lys = false;}  
}
```

```
interface KaninOppbevaring {  
    public boolean settInn(Kanin k);  
    public Kanin taUt( );  
}
```

```
interface Lys {  
    public void tennLyset ( );  
    public void slukkLyset ( );  
}
```

# Étt objekt– to grensesnitt – tre briller



```
interface KaninOppbevaring {  
    public boolean settInn(Kanin k);  
    public Kanin taUt( );  
}
```

```
interface Lys {  
    public void tennLyset ( );  
    public void slukkLyset ( );  
}
```

Vi kan se på objektet både med KaninburMedLys-briller og med KaninOppbevaring-briller og med Lys-briller



Forskjellige briller = forskjellige roller





```
class KaninburMedLys implements KaninOppbevaring, Lys {
```

```
    private boolean lys = false;
```

```
    private Kanin denne = null;
```

```
    @Override
```

```
    public boolean settInn(Kanin k) {
```

```
        if (denne == null) {
```

```
            denne = k;
```

```
            return true;
```

```
        }
```

```
        else {return false;}
```

```
    }
```

```
    @Override
```

```
    public Kanin taUt( ) {
```

```
        Kanin k = denne;
```

```
        denne = null;
```

```
        return k;
```

```
    }
```

```
    @Override
```

```
    public void tennLyset ( ) {lys = true;}
```

```
    @Override
```

```
    public void slukkLyset ( ) {lys = false;}
```

```
}
```

```
interface KaninOppbevaring {  
    public boolean settInn(Kanin k);  
    public Kanin taUt( );  
}
```

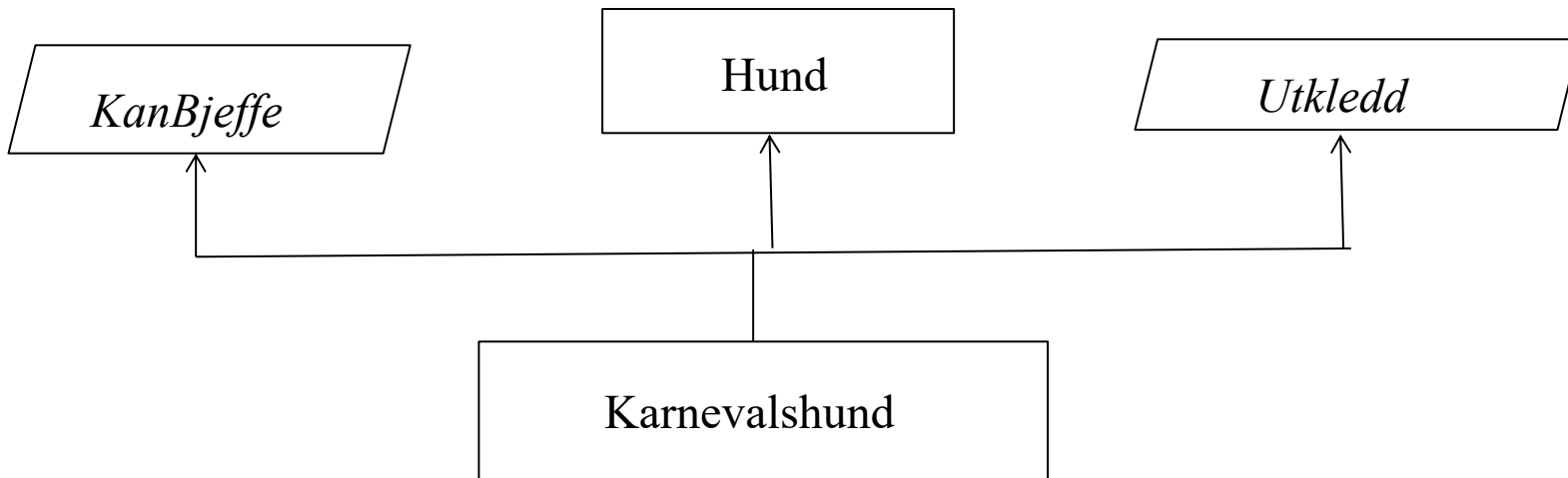
```
interface Lys {  
    public void tennLyset ( );  
    public void slukkLyset ( );  
}
```

# Karnevalshund

To (eller flere) grensesnitt  
= to (eller flere) roller



Foto: AP



# Flere eksempler: En klasse – mange grensesnitt

```
class Hund {protected double vekt;}
```

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

```
interface Utkledd {  
    int antallFarger();  
}
```

```
class Karnevalshund extends Hund implements KanBjefte,Utkledd {  
    protected int farger;  
    public Karnevalshund (int frg) { farger = frg; }  
    @Override  
    public void bjeff( ) {  
        System.out.println("Voff - voff");  
    }  
    @Override  
    public int antallFarger() {  
        return farger;  
    }  
}
```

Foto: AP

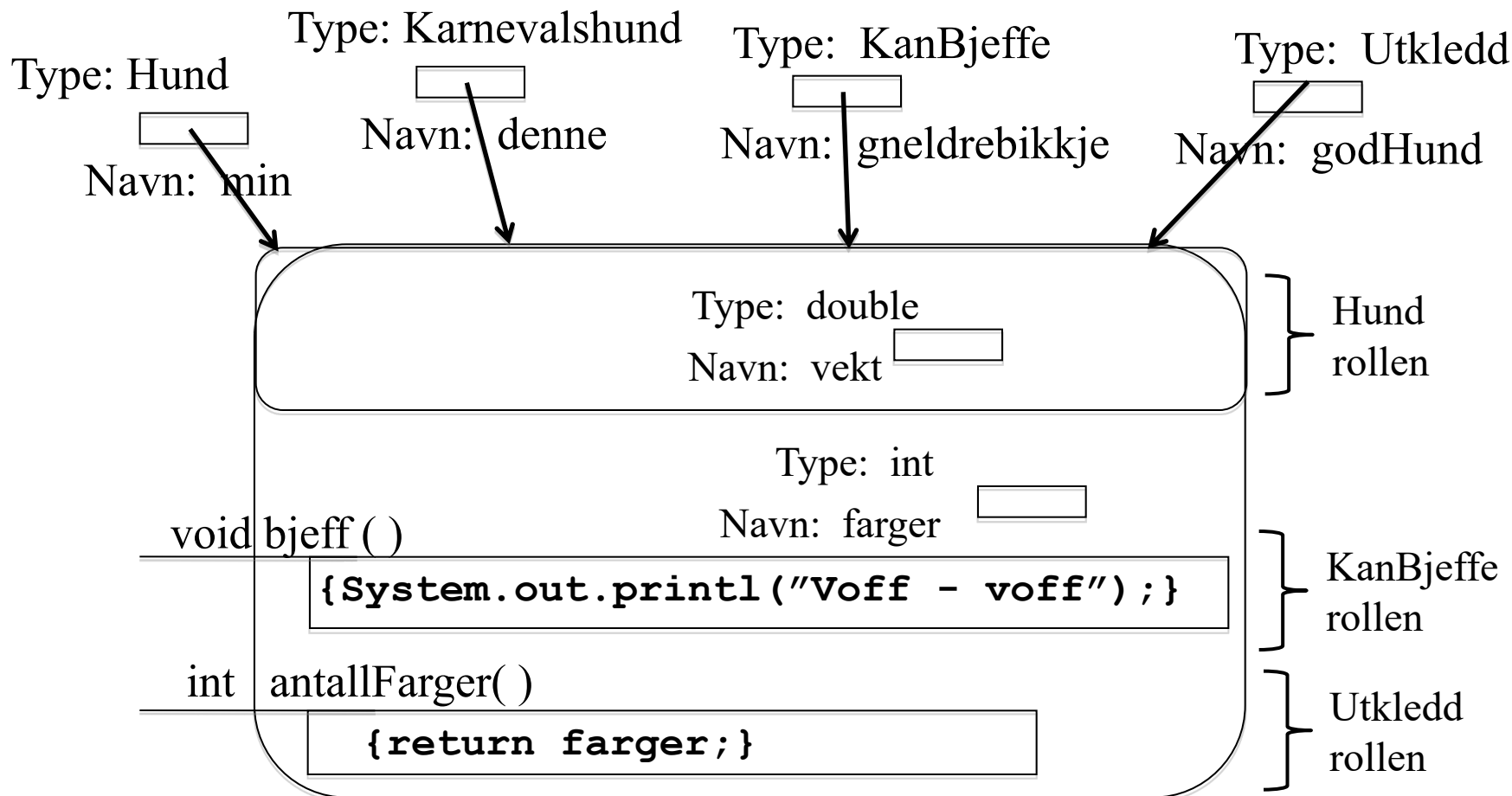




```

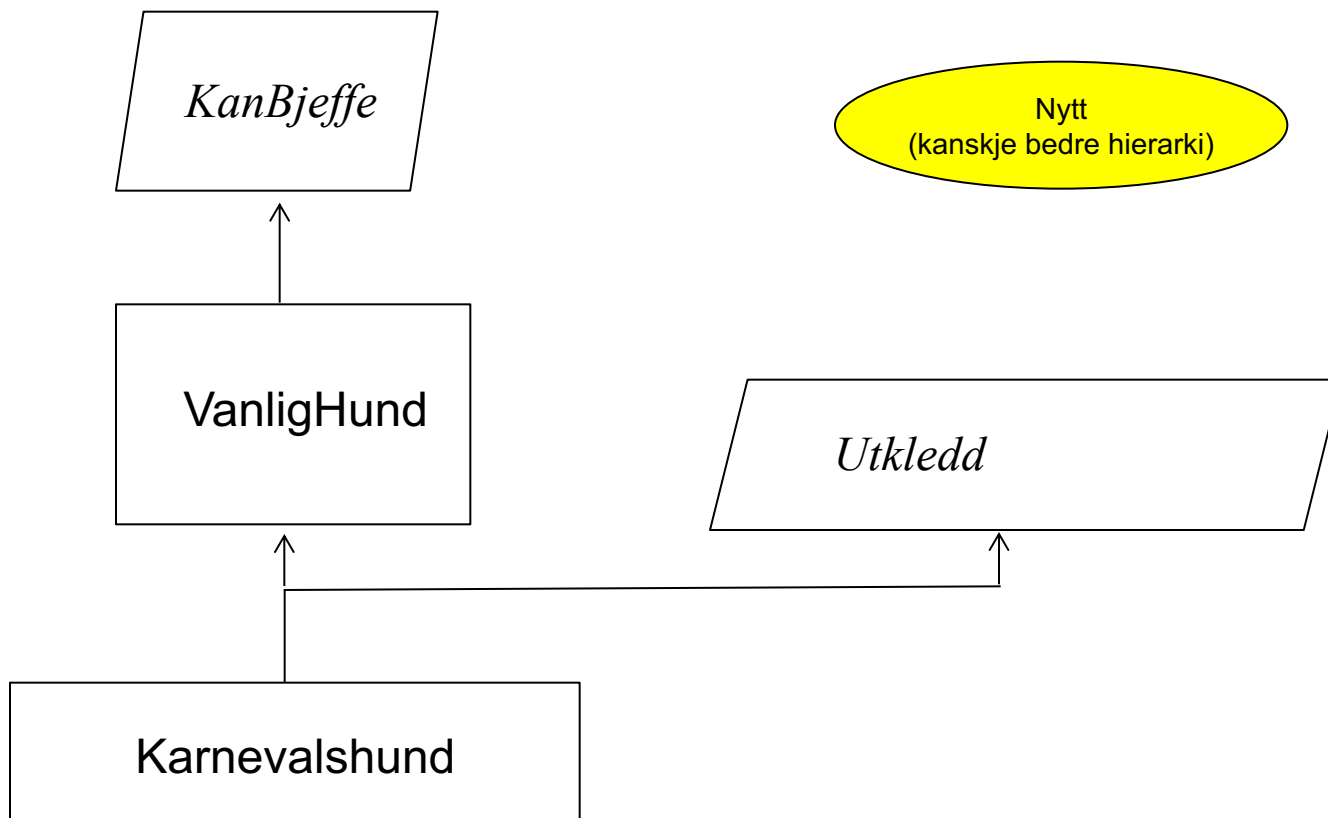
Karnevalshund passopp = new Karnevalshund( );
Hund min           = passopp;
KanBjeffe  gneldrebikkje = passopp;
Utkledd    godHunden    = passopp;

```



Objekt av klassen Karnevalshund

## Eller kanskje bedre:



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



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

```
interface Utkledd {  
    int antallFarger();  
}
```

```
class VanligHund implements KanBjefte {  
    @Override  
    public void bjeff() {  
        System.out.println("Vov-vov");  
    }  
}
```

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



Foto: AP



## Enda et eksempel :

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

interface Svingermor{
    boolean oKPaaBesok();
}

class NorskSvingermor extends KanBjefte, Svingermor {
    private boolean hyggelig = false;
    @Override
    public void bjeff( ) {
        System.out.println("Uff - uff");
    }
    @Override
    public boolean oKPaaBesok() {
        return hyggelig;
    }
}
```

Oppgave: Tegn opp et objekt av klassen NorskSvingermor og tre pekere av forskjellig type.

Hvilke roller kan dette objektet spille ?

Hva ser vi ved hjelp av de forskjellige pekerene?

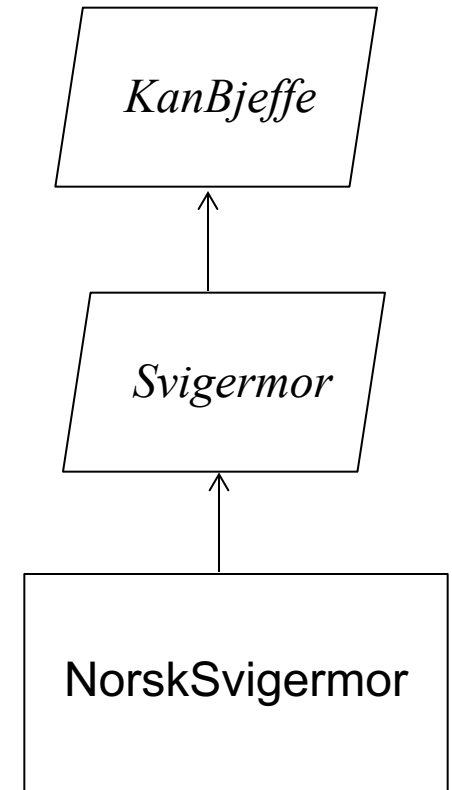


# Alternativ svigemor: Arv fra grensesnitt til grensesnitt

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

interface Svigermor extends KanBjeffe {
    boolean oKPaaBesok();
}

class NorskSvigermor implements Svigermor {
    private boolean hyggelig = false;
    @Override
    public void bjeff( ) {
        System.out.println("Uff - uff");
    }
    @Override
    public boolean oKPaaBesok() {
        return hyggelig;
    }
}
```







## Eksempel: Både biler og ost skal skattlegges

```
interface Skattbar{                                     // Skatt på importerte varer
    int skatt();
}

class Bil implements Skattbar { // Bil: 100% skatt
    private . . .
    private int importpris;
    public Bil (. . . ) {
        . . .
    }
    @Override
    public int skatt( ){return importpris;}
    . . .
}

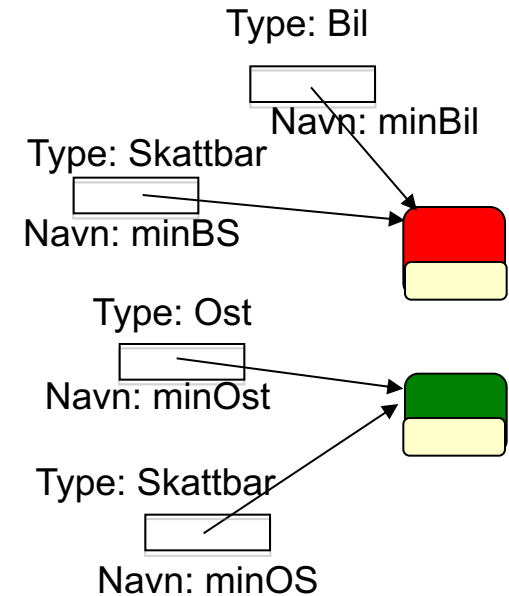
class Ost implements Skattbar { // Ost: 200% skatt
    private int importprisPrKg;
    private int antKg;
    public Ost (. . . ) {
        . . .
    }
    @Override
    public int skatt( ){return importprisPrKg*antKg*2.00;}
}
```

- Legg merke til at metoden skatt er implementert på forskjellige måter i Bil og Ost.

```
Bil minBil = new Bil ("BP12345", 100000);  
Skattbar minBS = minBil;  
Ost minOst = new Ost(100, 2);  
Skattbar minOS = minOst;
```

```
int bilskatt = minBil.skatt();  
int osteskatt = minOst.skatt();  
int skatt = bilskatt + osteskatt;
```

```
// bedre:  
int totalSkatt = 0;  
totalSkatt = totalSkatt + minBS.skatt();  
totalSkatt = totalSkatt + minOS.skatt();
```



 Rollen Skatt

 Rollen Bil (untatt Skatt)

 Rollen Ost (untatt Skatt)

# Samlet import-skatt

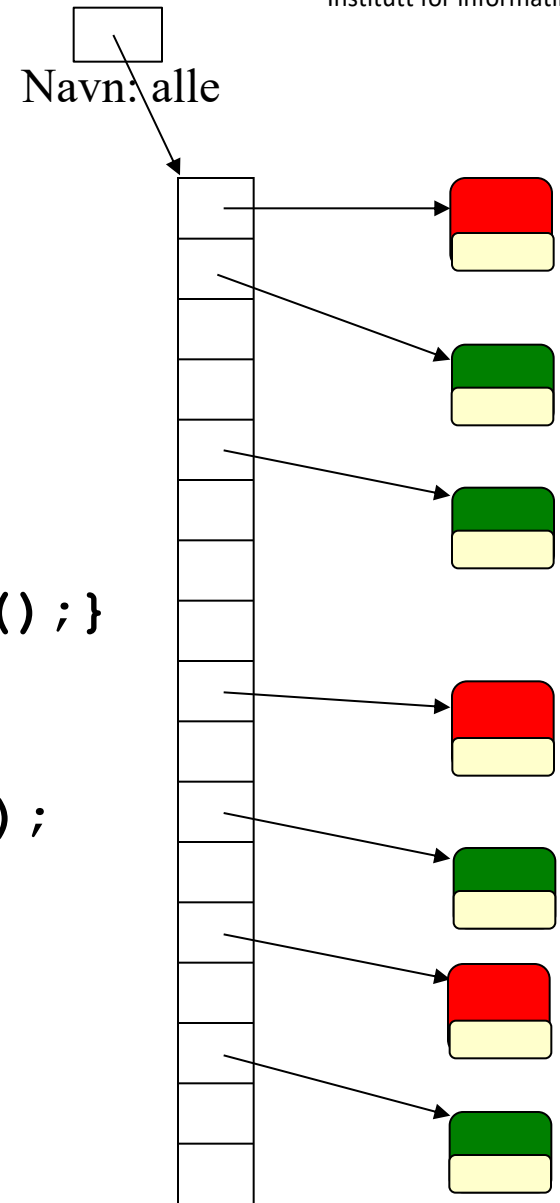
Type: Skattbar [ ]

```
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 Skatt

 Rollen Bil (untatt Skatt)

 Rollen Ost (untatt Skatt)



Veldig viktig og bra eksempel. Dagens rosin.





# Bil og Ost – Full kode

```
interface Skattbar{                                     // Skatt på importerte varer
    int skatt();
}

class Bil implements Skattbar { // Bil: 100% skatt
    private String regNr;
    private int importpris;
    public Bil (String reg, int imppris) {
        regNr = reg; importpris = imppris;
    }
    @Override
    public int skatt( ){return importpris;}
    public String hentRegNr( ) {return regNr;}
}

class Ost implements Skattbar { // Ost: 200% skatt
    private int importprisPrKg;
    private int antKg;
    public Ost (int kgPris, int mengde) {
        importprisPrKg = antKg; antKg = mengde;
    }
    @Override
    public int skatt( ){return importprisPrKg*antKg*2.00;}
}
```

## 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 forskjellige roller (multippel arv)
- Forskjellige objekter kan implementere samme rolle på forskjellige måter
  - innkapsling = skjuling av detaljer

MYE MER I EKSEMPLER