UNIVERSITETET I OSLO

IN1010 våren 2020

Onsdag 13. mai

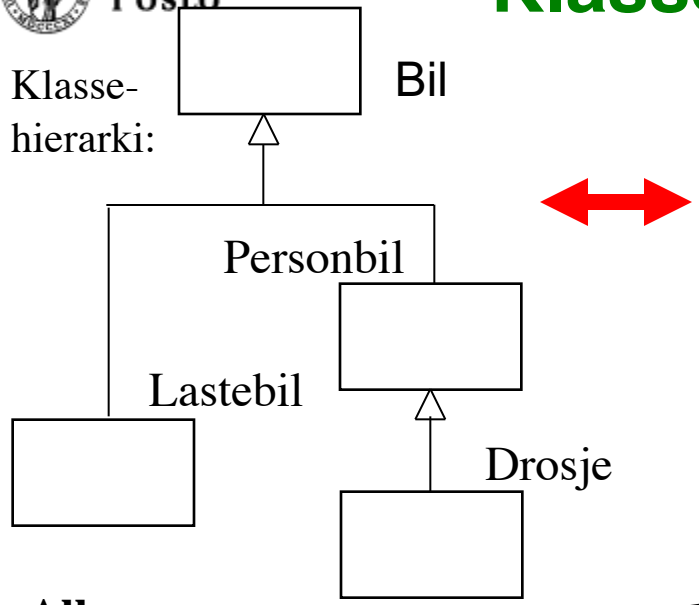
Repetisjon

Stein Gjessing
Institutt for informatikk
Universitetet i Oslo

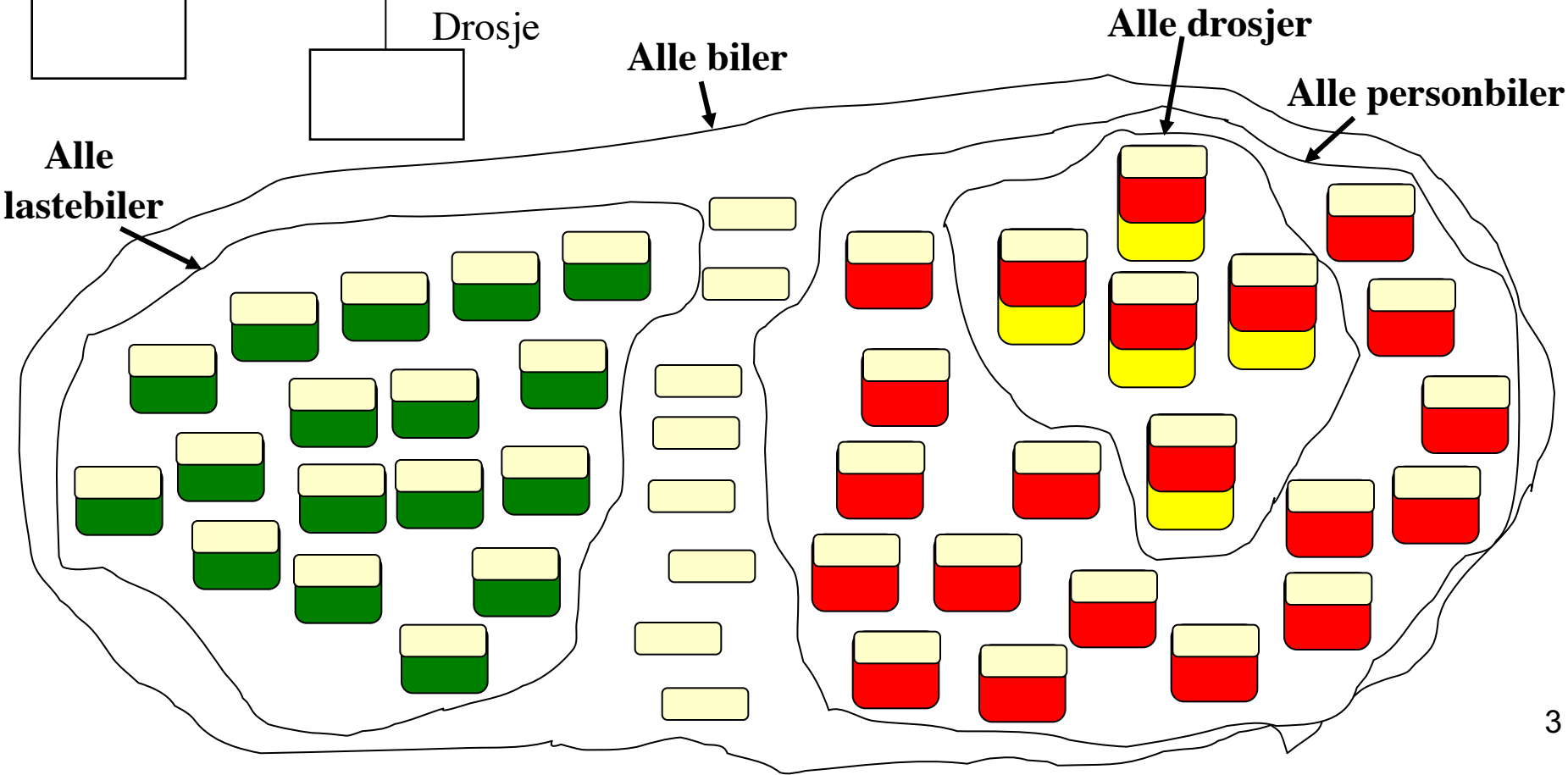
Innledning

- Dette er 51 lysark som det ikke er mulig å gå gjennom på én time
- Disse lysarkene representerer det viktigste i pensum
- Jeg kommer til å peke på noen få viktige poeng, og så er meningen at dere kan bruke lysarkene når dere skal repetere stoffet på egenhånd

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> }
```



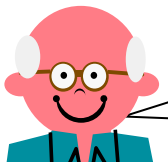
Private og public i subklasser

Private i en klasse gjør at ingen utenom denne klassen (og heller ikke subklasser) kan se denne egenskapen

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

Public er som før

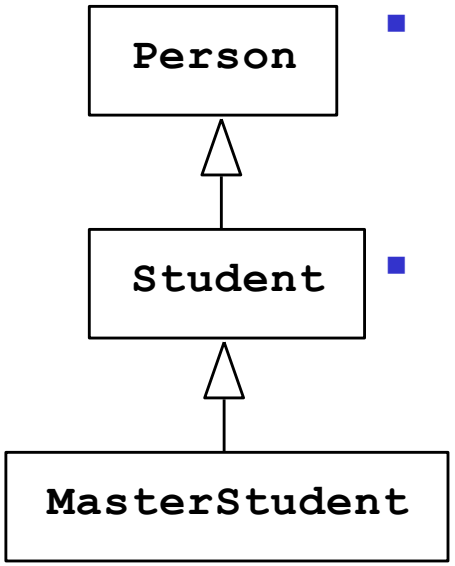
```
class Person {
        protected String navn;
        protected int tlfnr;

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



Nytt reservert ord i Java:
protected

Konvertering mellom flere nivåer

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



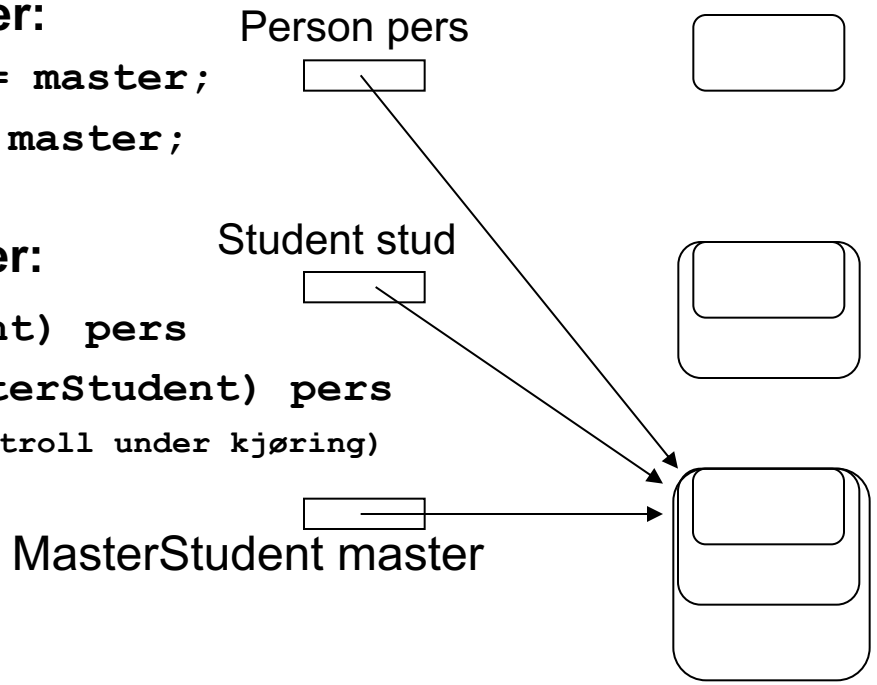
- **Konvertering oppover:**

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

- **Konvertering nedover:**

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

(fordi dette krever kontroll under kjøring)

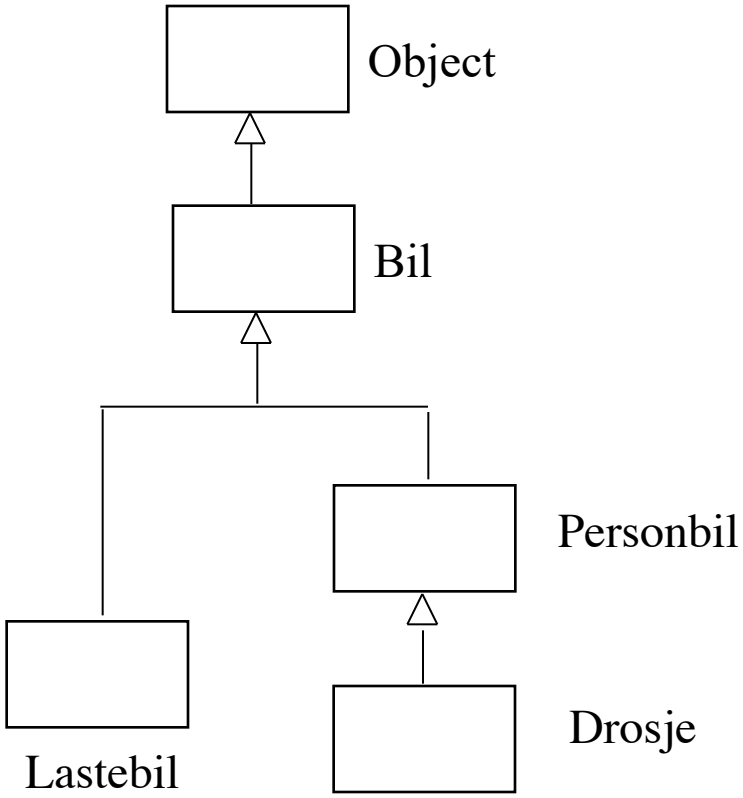


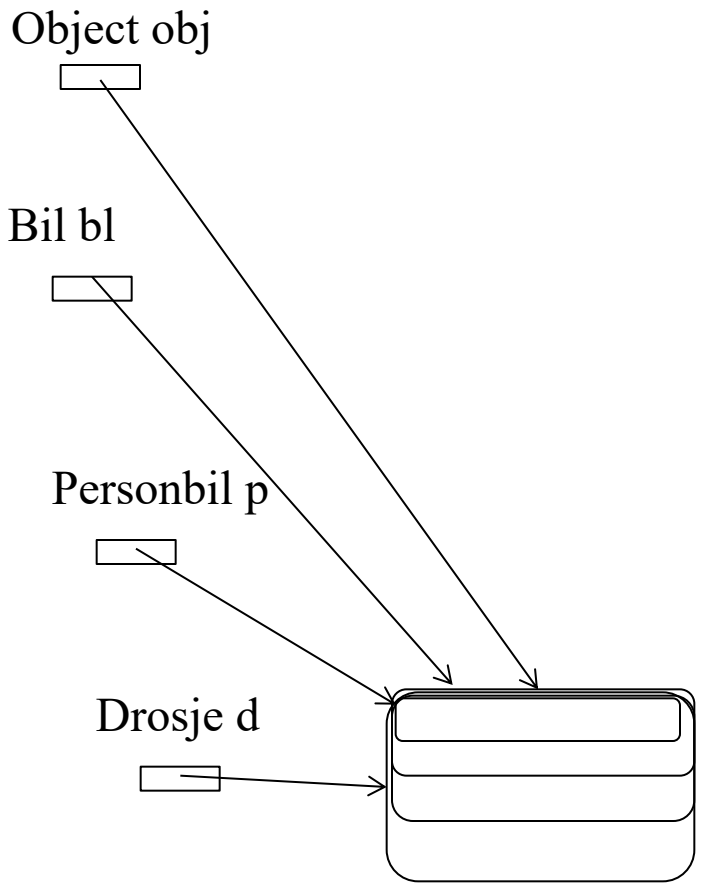
Regel: ” Alle referanser har lov til å peke bortover og nedover”

(men ikke ”oppover”)

Eksempel

Klassehierarki:

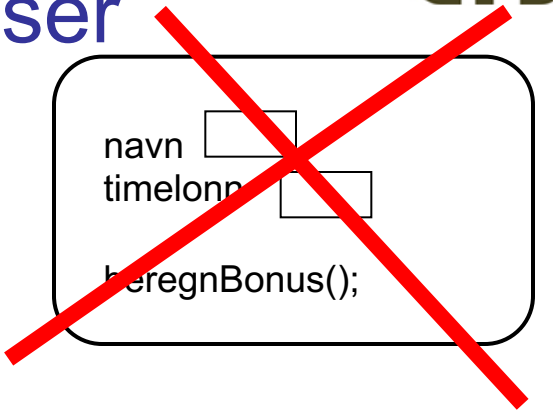


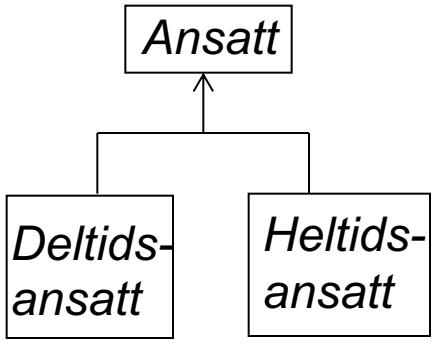


Abstrakte metoder og klasser

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

    public abstract double beregnBonus();
}
```

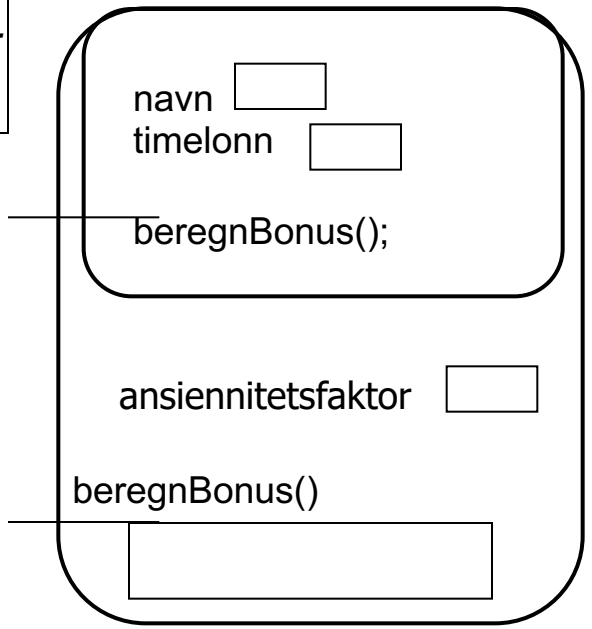
Ikke lov å si
new Ansatt() !






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

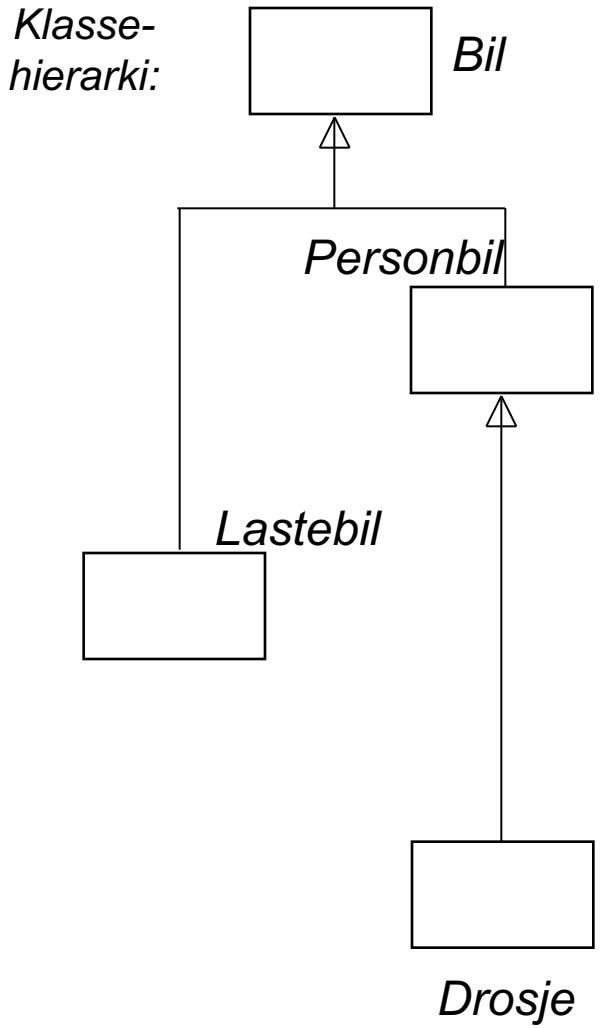
Heltidsansatt objekt




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

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

Polymorfi

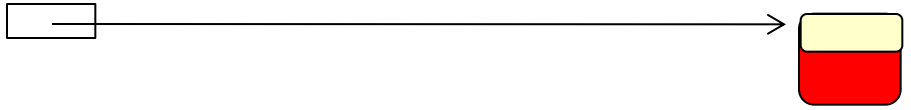
```
class Bil {
    protected int pris;
    public int skatt( ) {return pris;}
}

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;}
}
```



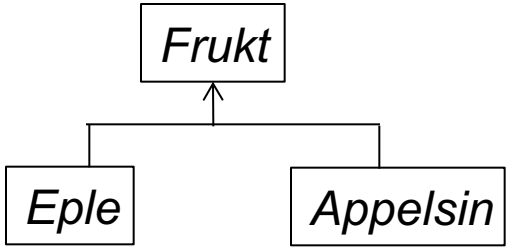
Hva slags objekt er dette?

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 { .. }
```



Nøkkelordet **super.**

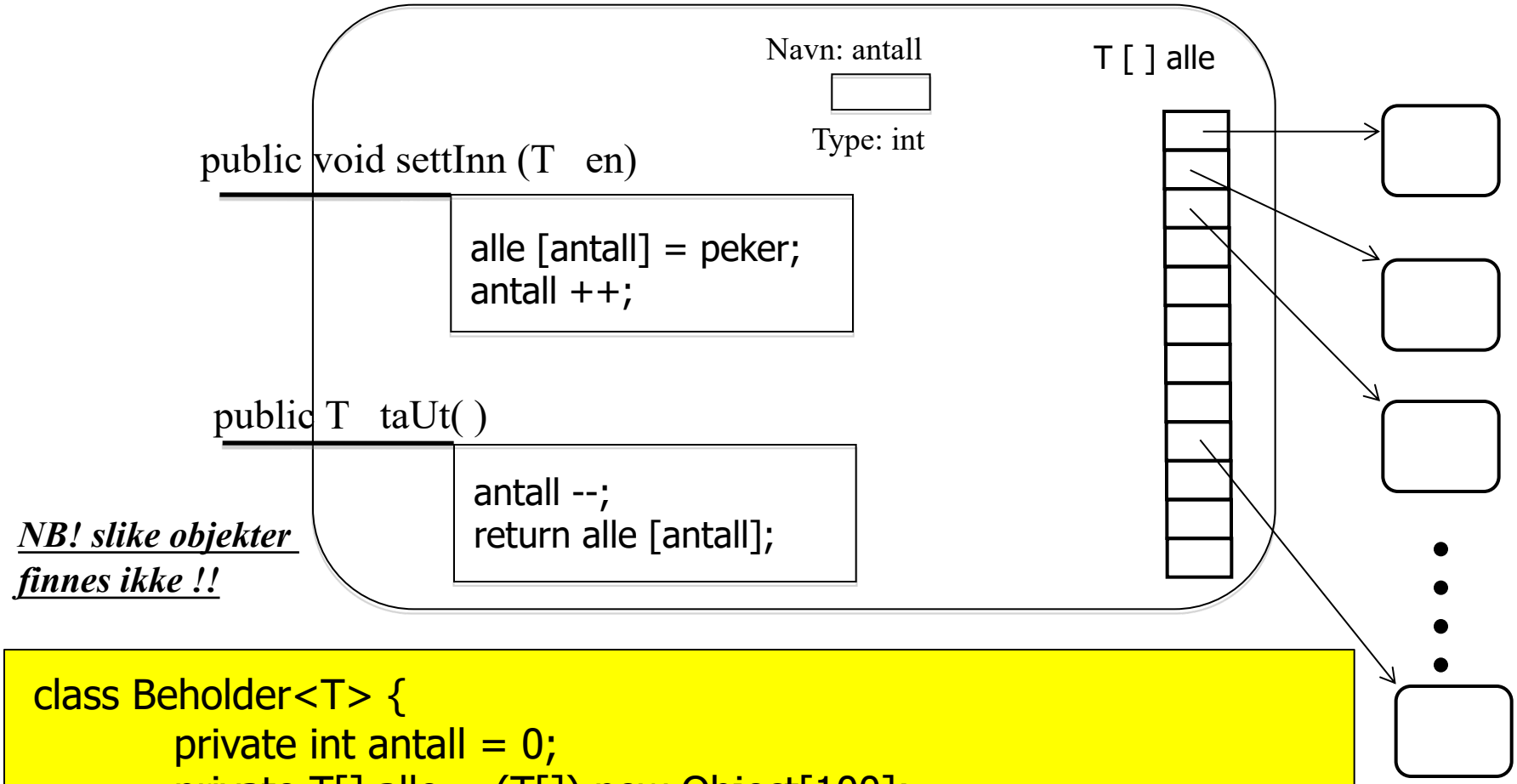
Nøkkelordet **super** brukes til å aksessere metoder i objektets superklasse. Dette kan vi bruke til å la superklassen Person ha en generell **skrivData**, som så kalles i subklassene:

```
// 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();

En generell stor beholder:




```
class Beholder<T> {
        private int antall = 0;
        private T[] alle = (T[]) new Object[100];

        public void settInn(T peker) {alle [antall] = peker; antall ++; }

        public T taUt() { antall --; return alle[antall]; }
}
```

"unchecked cast"

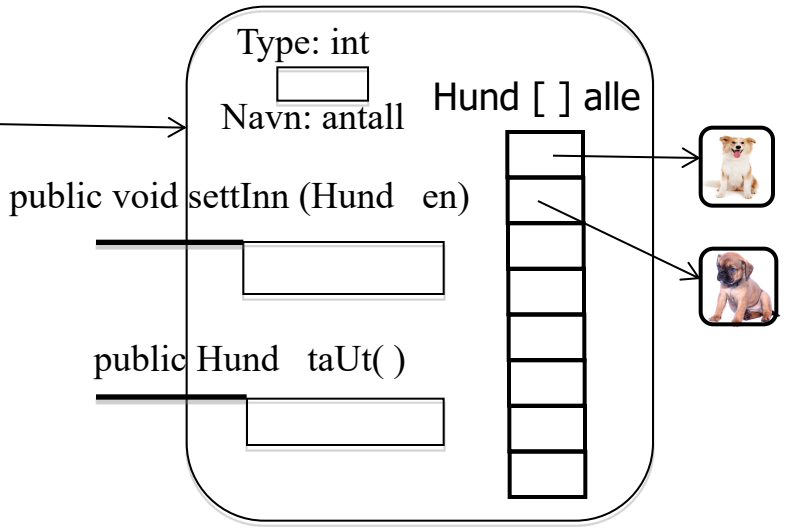
```
Terminal — bash — 83×11
ammoniake:programmer steing$ javac Hoved.java
Note: Hoved.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
ammoniake:programmer steing$ javac -Xlint:unchecked Hoved.java
Hoved.java:23: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: T[]
          T[] alle = (T[]) new Object[100];
                           ^
1 warning
ammoniake:programmer steing$
```

Ikke bry deg om dette.
Grunnen er at under kjøring vet Java ikke hva slags klasse som brukes inne i objekter av den generiske typen.
Dette er en "feil" i Javas kjøresystem.

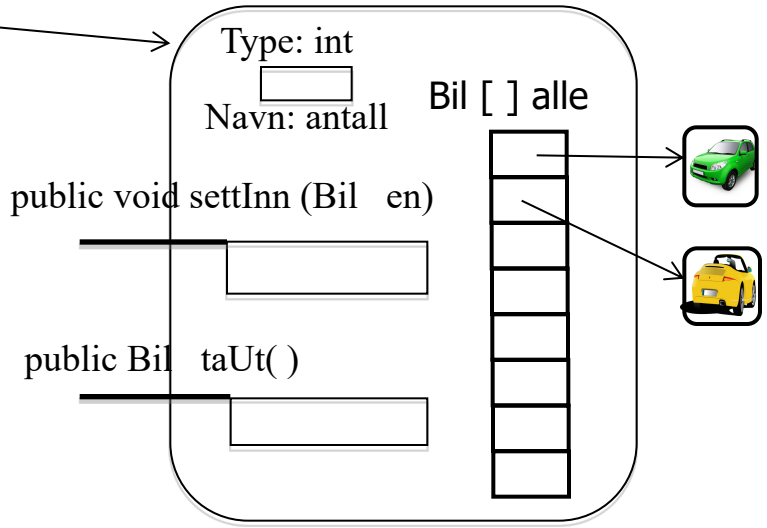
Litt mer om dette senere i semesteret.

Da kan vi f.eks. skrive:

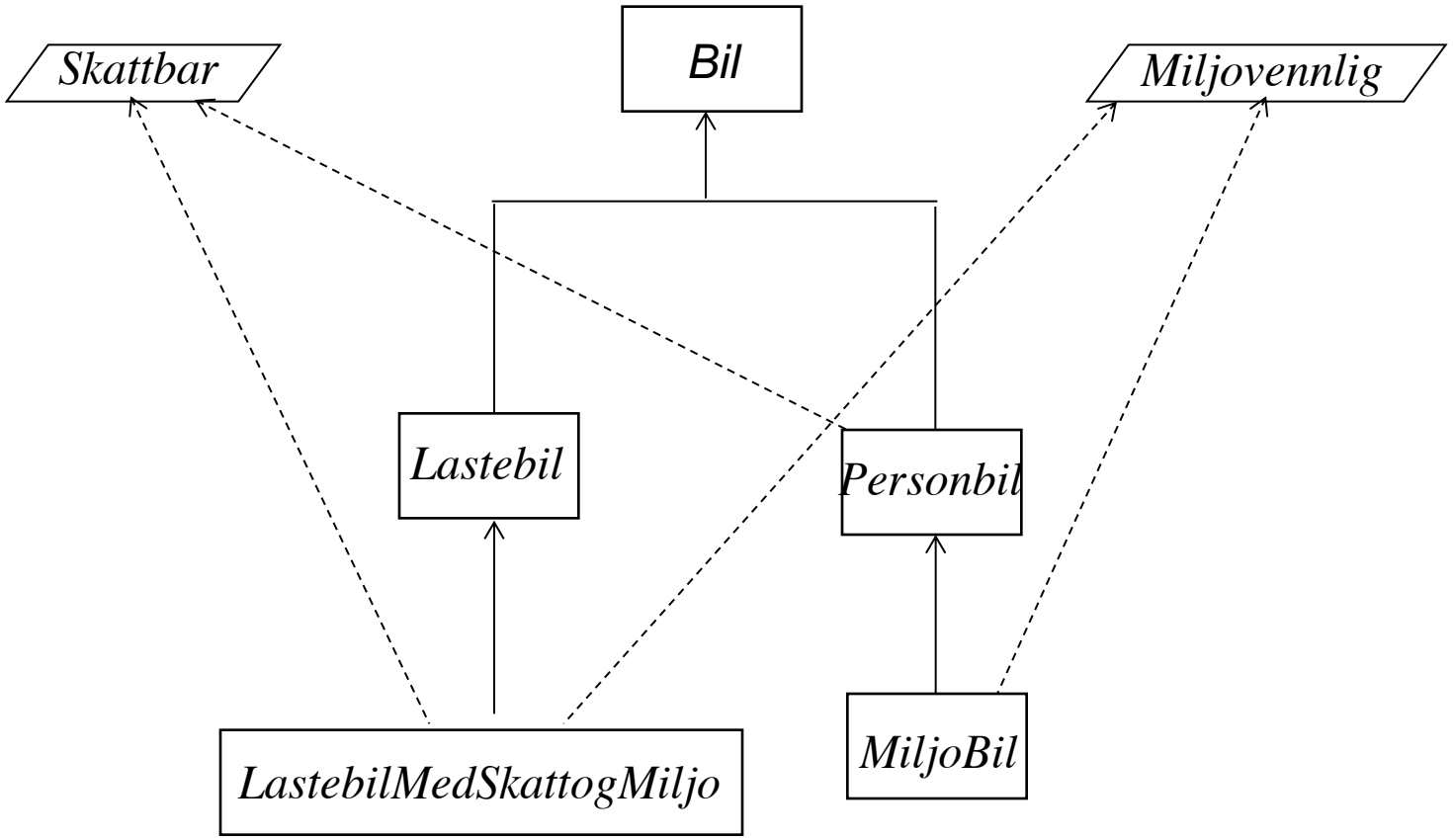
```
Beholder<Hund> minHundegard = new Beholder<Hund> ( );

Hund passopp = new Hund("Passopp");
Hund trofast   = new Hund("Trofast");
minHundegard.settInn(passopp);
minHundegard.settInn(trofast);
Hund drittbikkje;
drittbikkje = minHundegard.taUt();
System.out.println(" Drittbikkja heter: "
                        + drittbikkje.navn);
```




```
Beholder<Bil> fellesGarasjen = new Beholder<Bil> ( );

Bil dinBil  = new Bil("AD98764");
Bil minBil = new Bil("DK12365");
fellesGarasjen.settInn(minBil);
fellesGarasjen.settInn(dinBil);
fellesGarasjen.taUt();
Bilen denBilen = fellesGarasjen.taUt();
System.out.println(" Bilen som kjørte sist ut var: "
                        + denBilen.regNr);
```



Interface


Skattbar
Bil
Miljovennlig
Lastebil
Personbil
LastebilMedSkattogMiljo
MiljoBil

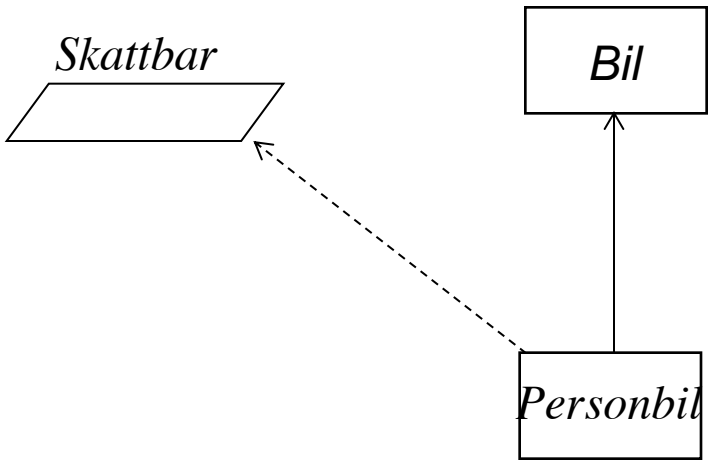
Implementere et interface: **implements**

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;}
}
```

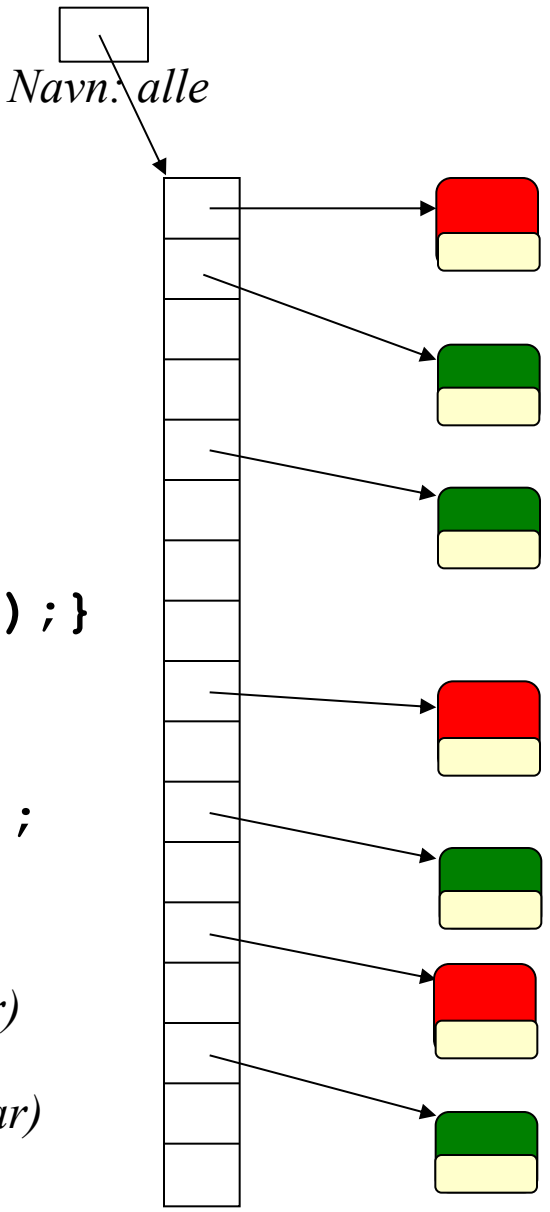
rollen "Skattbar"

```
class Bil {
    String regNr;
}

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



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);
```



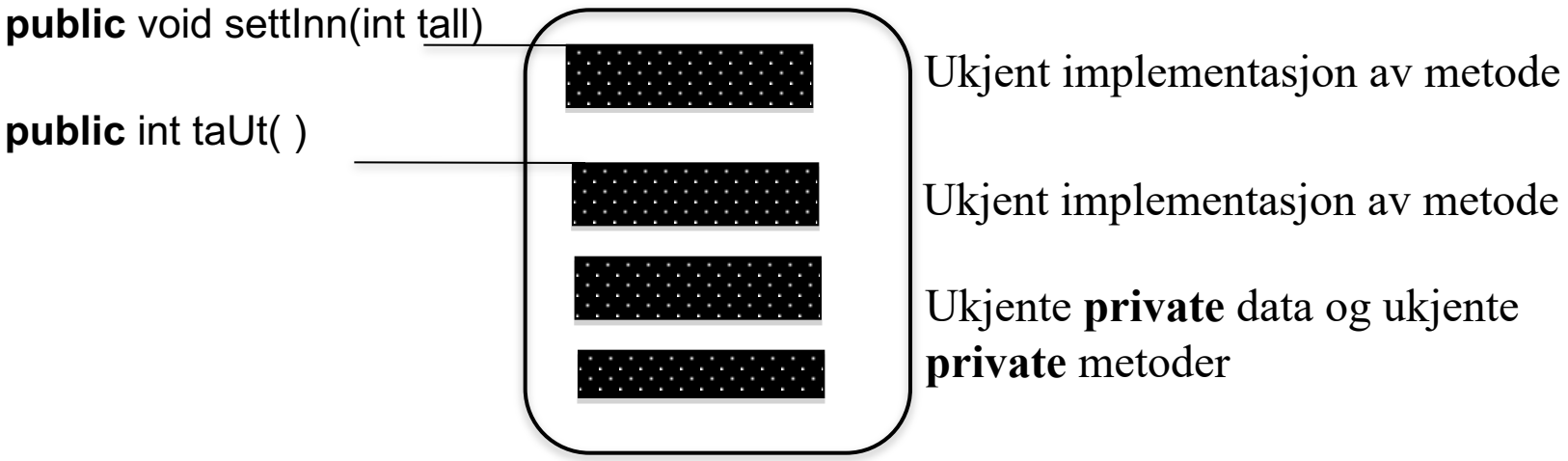
Rollen Skattbar

Rollen Bil (untatt Skattbar)

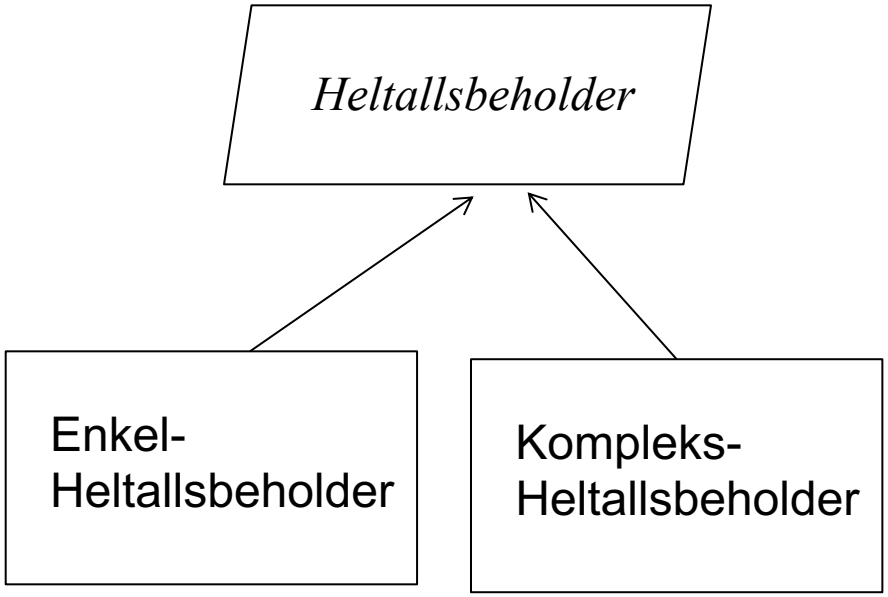
Rollen Ost (untatt Skattbar)

Veldig viktig og bra eksempel. Dagens rosin.

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



Interface: Klassehierarki og Java-kode



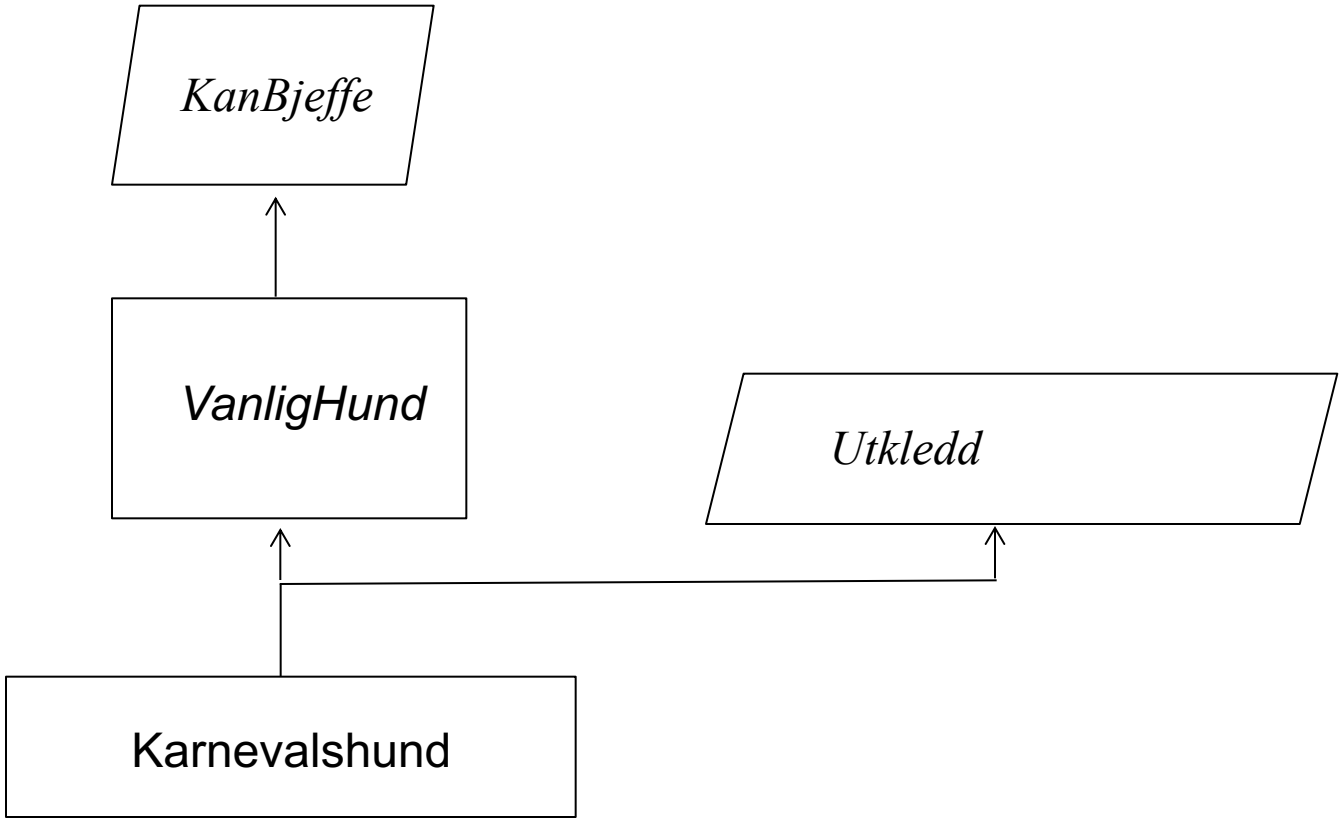

```
interface Heltallsbeholder {
        public void settInn(int tall);
        public int taUt( );
}
```

```
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;
       }
}
```

Generiske interface
Inteface med parametre

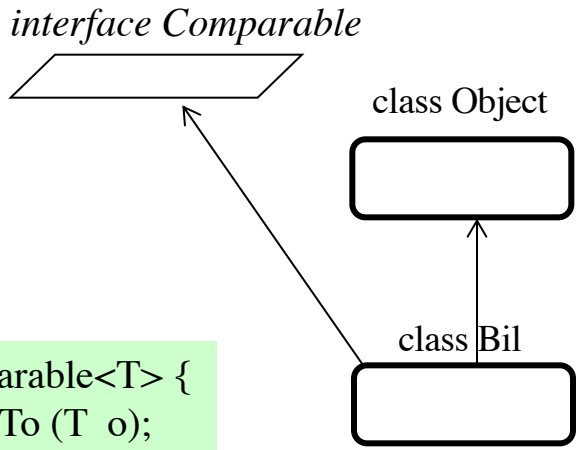
- På samme måte som klasser, kan interface lages med parametre.
- interface Beholder <E> { . . . }
- class GeneriskBeholderTilEn <E> { . . . }

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;}
}
```

Mest vanlig bruk




```
interface Comparable<T> {
      int compareTo (T  o);
}
```

```
class Bil implements Comparable <Bil> {
   private String regNr;
   private int vekt;
   public int compareTo(Bil b) {return  vekt – b.vekt; }
}
```

Oppsummering

*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)

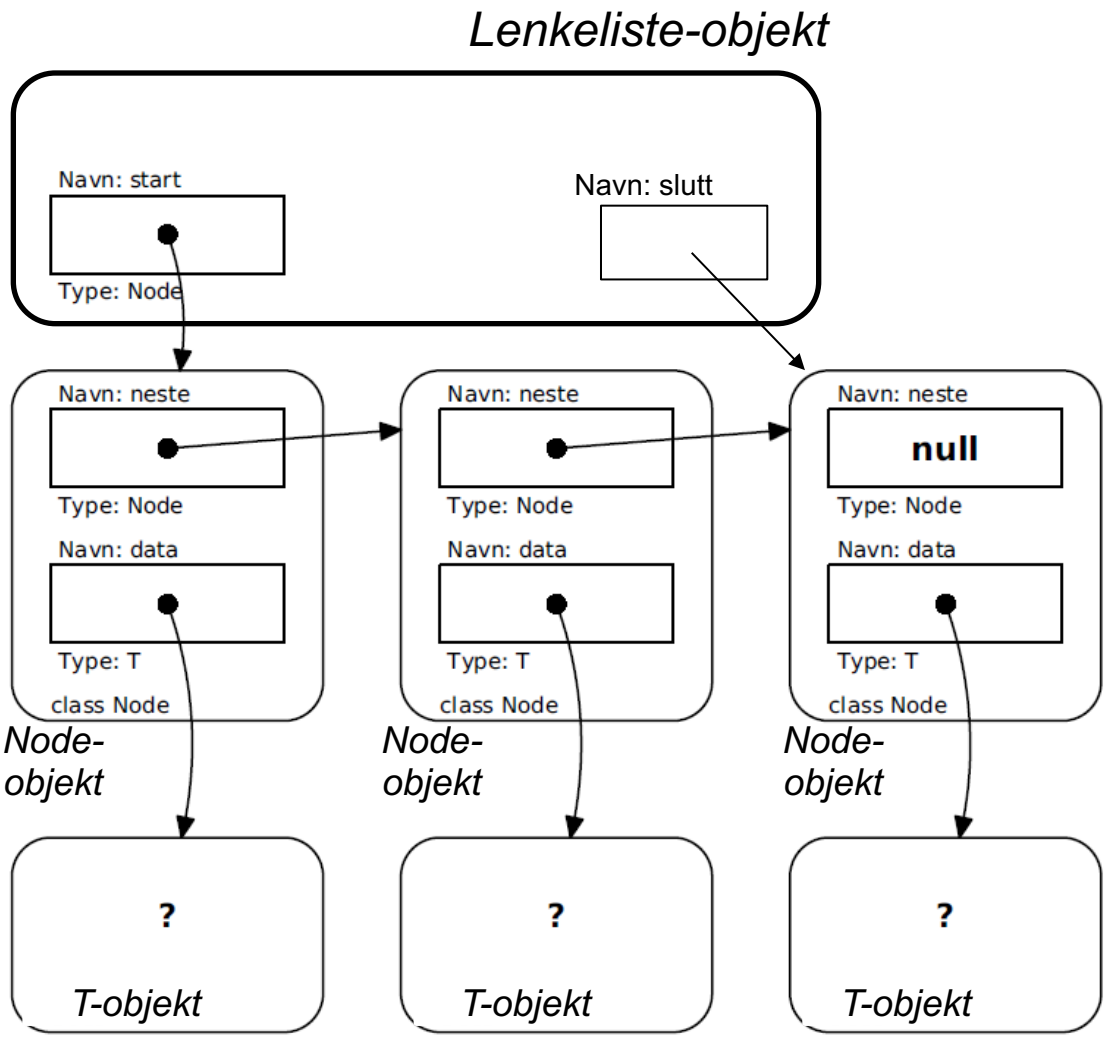
- Forskjellige objekter kan implementere samme rolle på forskjellige måter
  - innkapsling = skjuling av detaljer

Enkel-kjedete lister

Start / først
slutt / sist

Må ta ut foran

Sett inn foran = stakk
Sett inn bak = FIFO

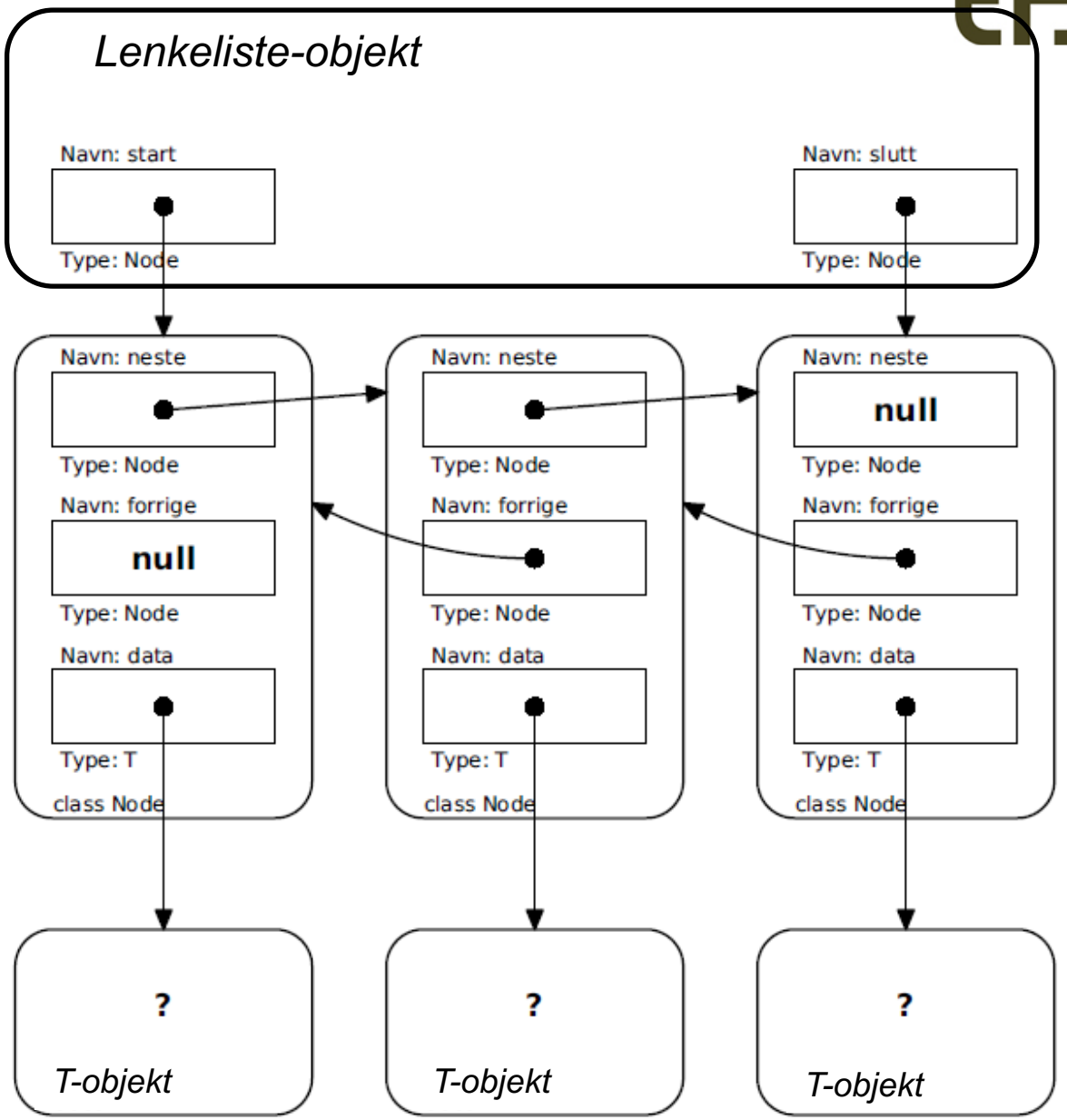


Husk: Datastruktur-tegninger

Toveisliste

- *Slipper å lete oss frem til element som skal fjernes*
- *Sett inn og fjerne i begge ender*

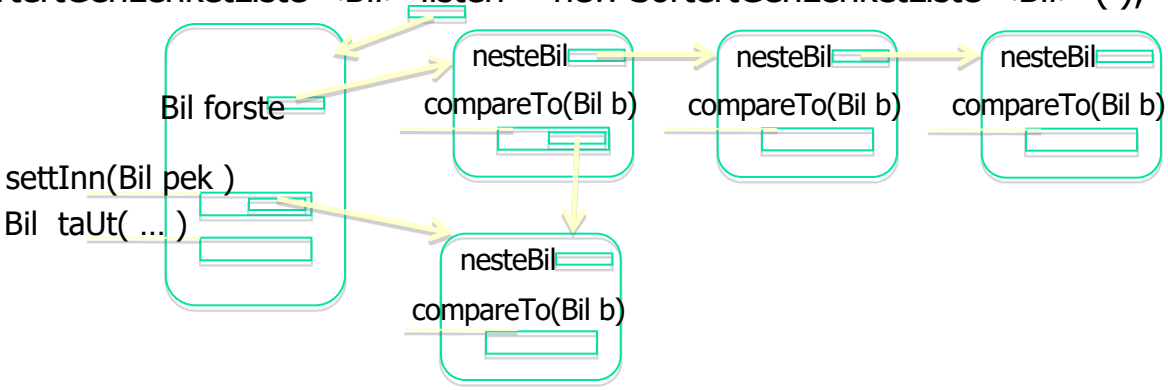
:



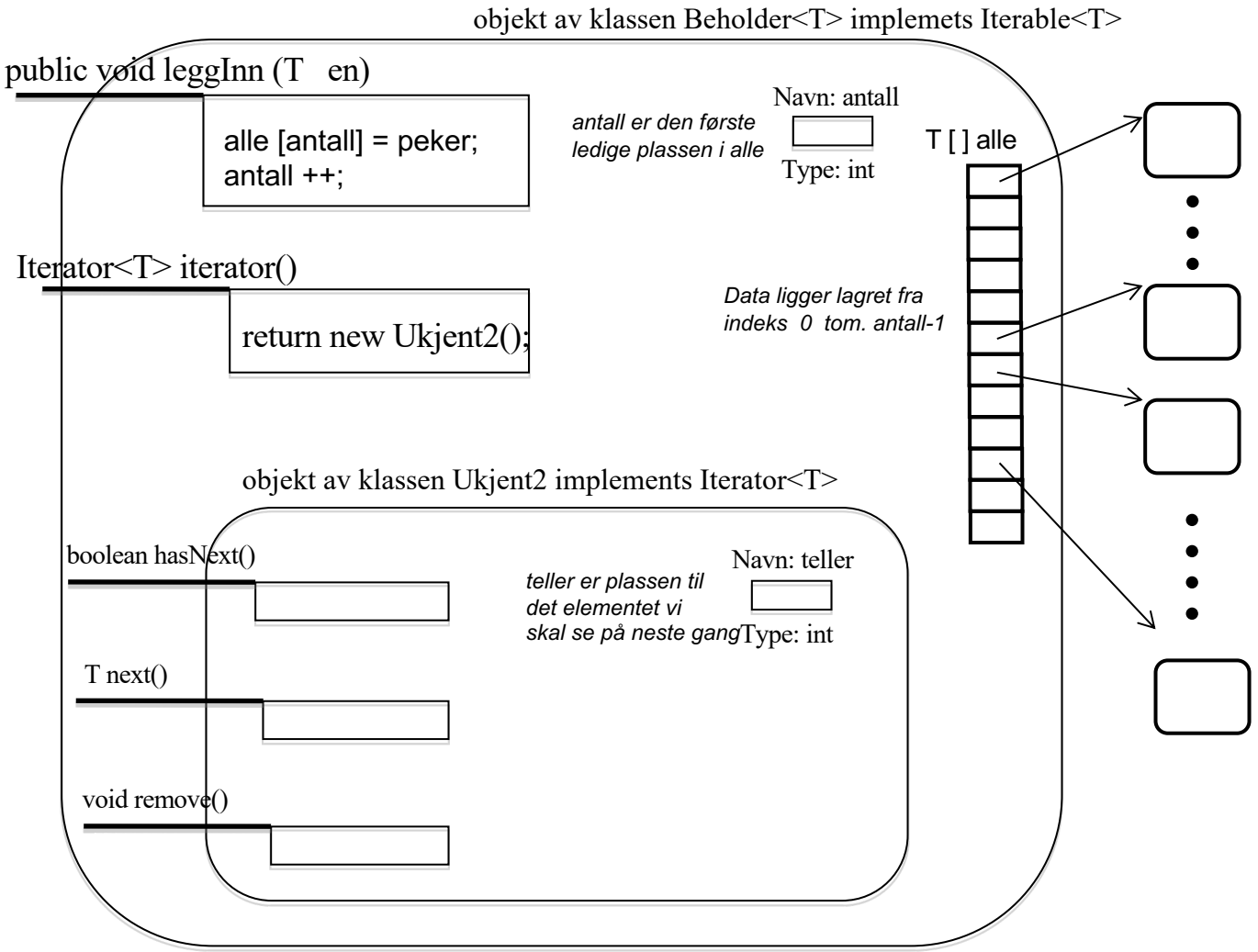
```
class SortertGenLenketListe<T extends Comparable <T>> {
    T forste;
    settInn (T pek) { T denne;
        int verdi = forste.compareTo(pek);
        . . .
     }
    T taut( . . .) { . . . }
}
```

```
class Bil implements Comparable<Bil>
{
    Bil nesteBil;
    <mer datastruktur>
    public int compareTo (Bil b) { . . . }
}
```

```
SortertGenLenketListe <Bil> listen = new SortertGenLenketListe <Bil> ( );
```



Grensesnittene Iterable og Iterator



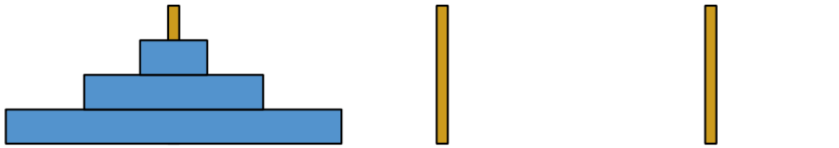
Rekursive metoder

- gaa() i labyrinten
  - Kall på gaa() i nabo-nodene
    - Vakkert og enkelt + med polymorfi (sorte ruter)
- Hanois tårn

Hanois tårn med 3 ringer

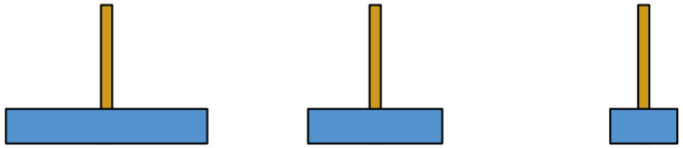
Steg

1

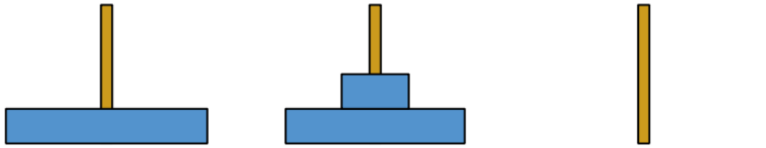


2

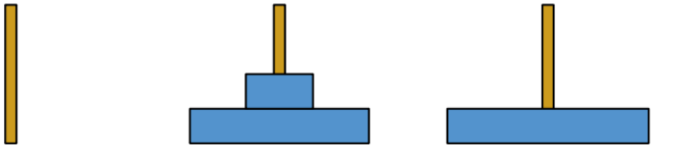
3



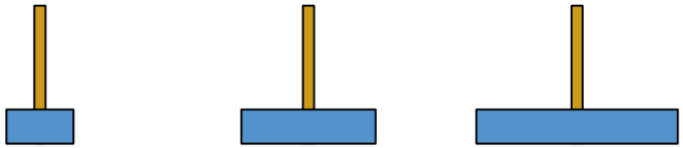
4



5

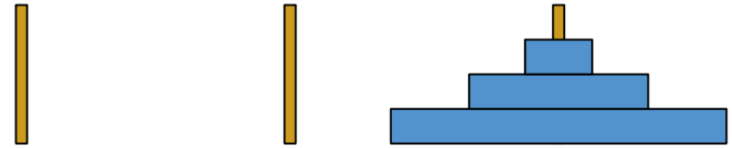


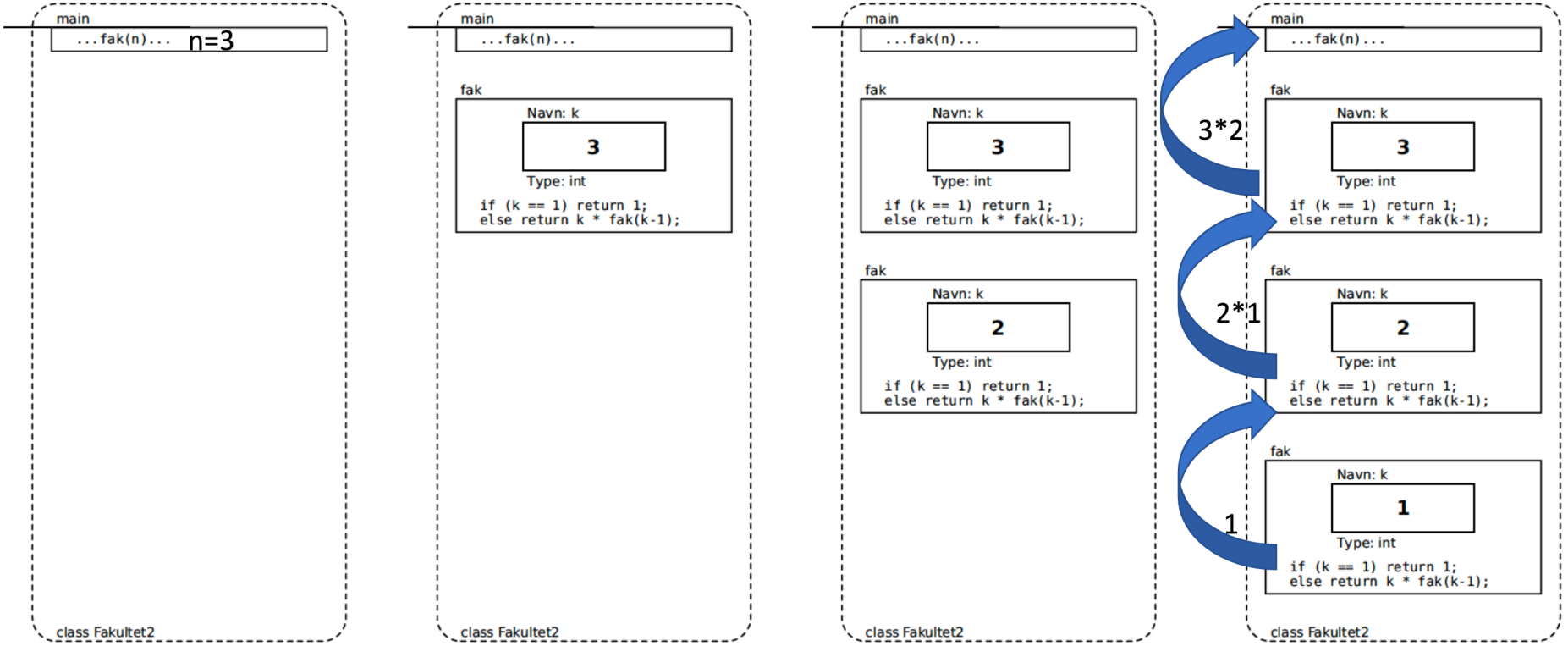
6



7

8





Fra Dag Langmyhr

Invarianter i **løkker**

Eksempel: **Finne minste verdi i tabell**

minstTilNaa inneholder minste verdi i området fra og med tabell [0] til og med tabell[indeks]

= *Inv(indeks):*

Induksjons-basis: indeks = 0

Induksjons-skritt: indeks = indeks + 1

Resultat: indeks = 999:

minstTilNaa inneholder minste verdi i området fra og med tabell [0] til og med tabell[999]

UNIVERSITETET I OSLO

For-betingelser
Bak-betingelser

// Vi vet ingenting annet enn at tabell [0] til og med
// tabell[999] inneholder tall. Vi skal finne det minste

For-betingelse. Engelsk: Pre-condition

Bak-betingelse.
Engelsk: Post-condition (Post-assertion)

// minstTilNaa inneholder minste verdi i området
// fra og med tabell[0] til og med tabell[999] !!!!!!

UNIVERSITETET I OSLO

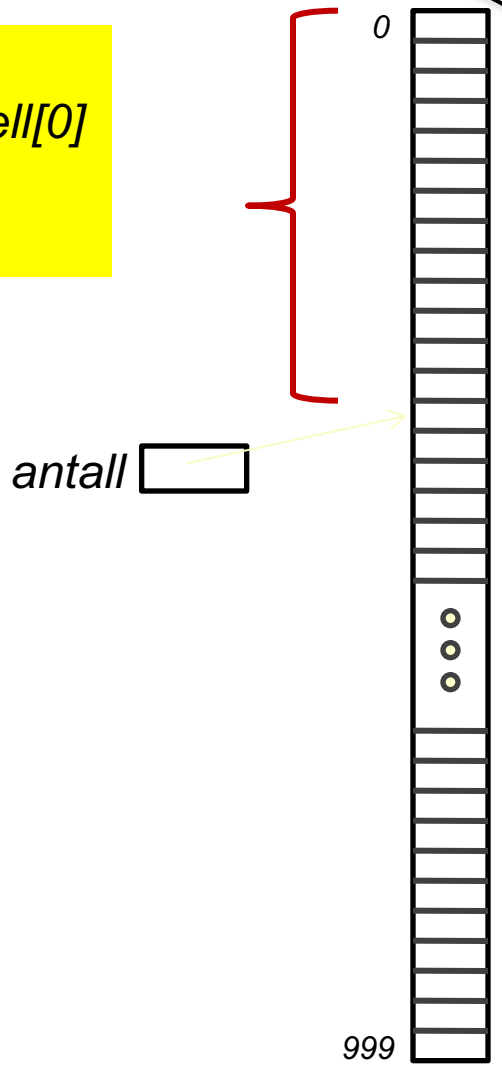
Invarianter i objekter (class invariantes)

Uten tråder

Invariant:
Alle dataene vi lagrer ligger i tabell[0] til og med tabell [antall – 1] og
0 <= antall <= 1000

```
settInn(x) {
    if (antall == 1000) return ;
    antall ++;
    tabell[antall-1] = x;
}
```

```
taUt ( ) {
    if (antall == 0) return null;
    antall --;
    return (tabell[antall]);
}
```



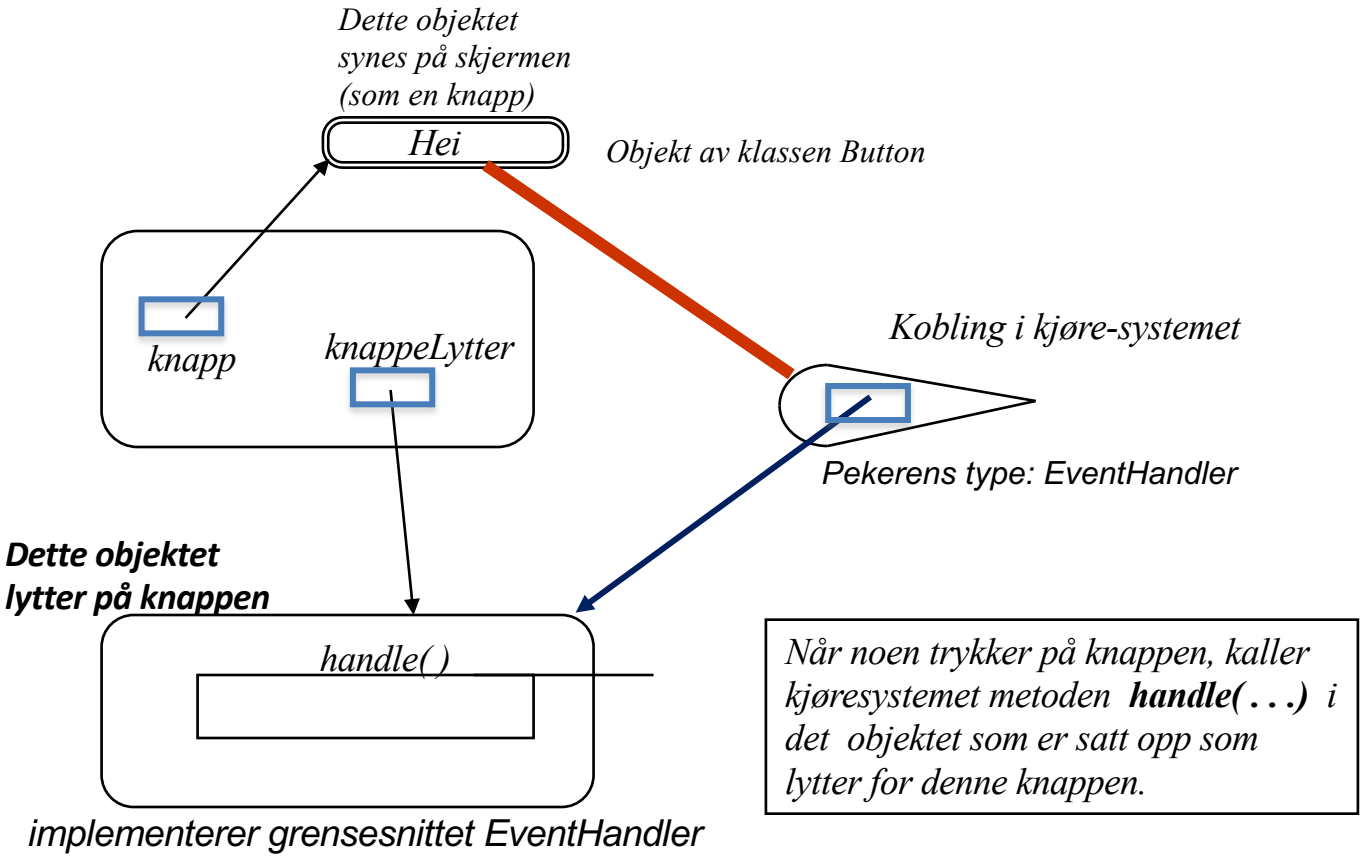
Overbevis deg (og andre) om at metodene bevarer Invarianten.
(og at den er sann ved oppstart)

Hendelseshåndtering / GUI

```
class KnappeHandterer implements EventHandler<ActionEvent> {
        @Override
         public void handle(ActionEvent e) { <Gjør det som trengs>}
 }
```

KnappeHandterer knappeLytter = new KnappeHandterer ();
Button knapp = new Button(”Hei");
knapp.setOnAction(knappeLytter);

GUI: Sette opp en knappelytter. Hva skjer ved et knappetrykk?

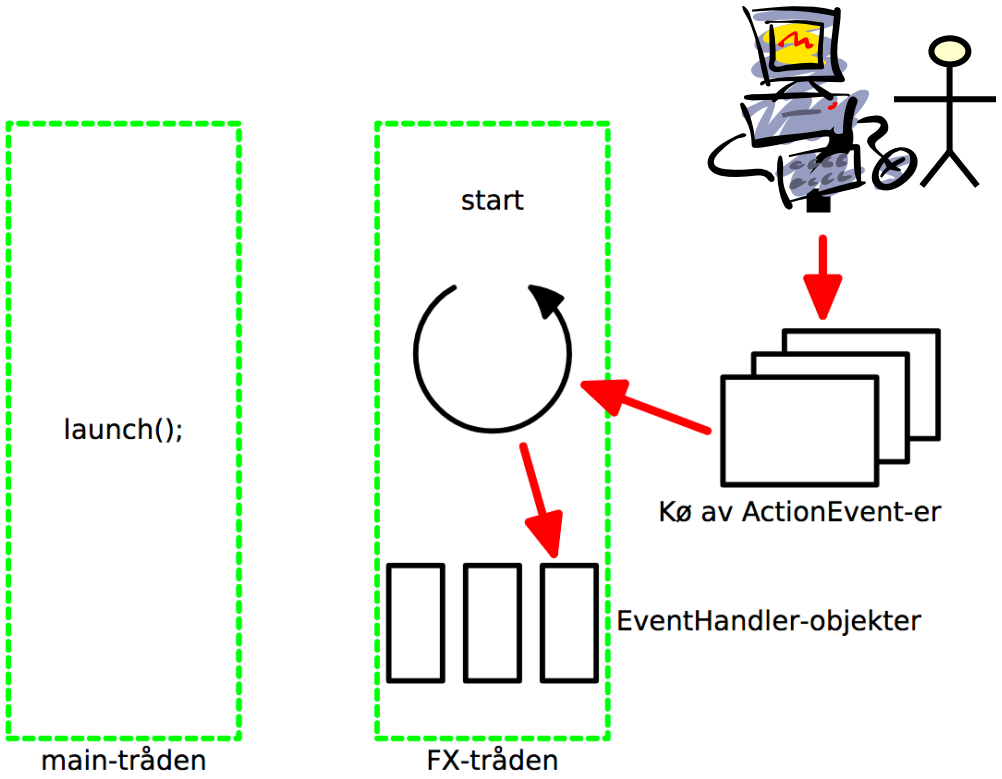


Når noen trykker på knappen, kaller kjøresystemet metoden ***handle(. . .)*** *i det objektet som er satt opp som lytter for denne knappen.*

Håndtere hendelser

Hver hendelse som inntreffer, resulterer i et ActionEvent-objekt i køen.

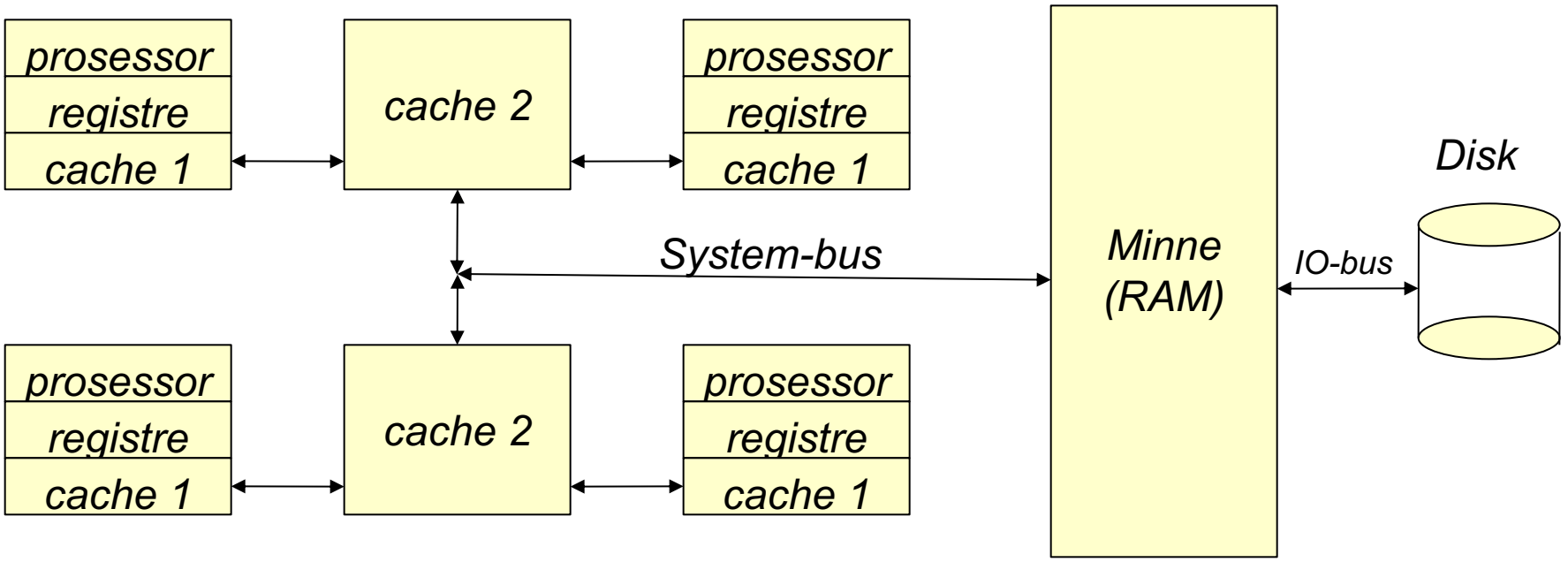
Hendelsesløkken i FX-tråden vil ta ActionEvent-ene etter tur, og den korrekte EventHandler-en vil bli kalt.



NB! FX-tråd**en**

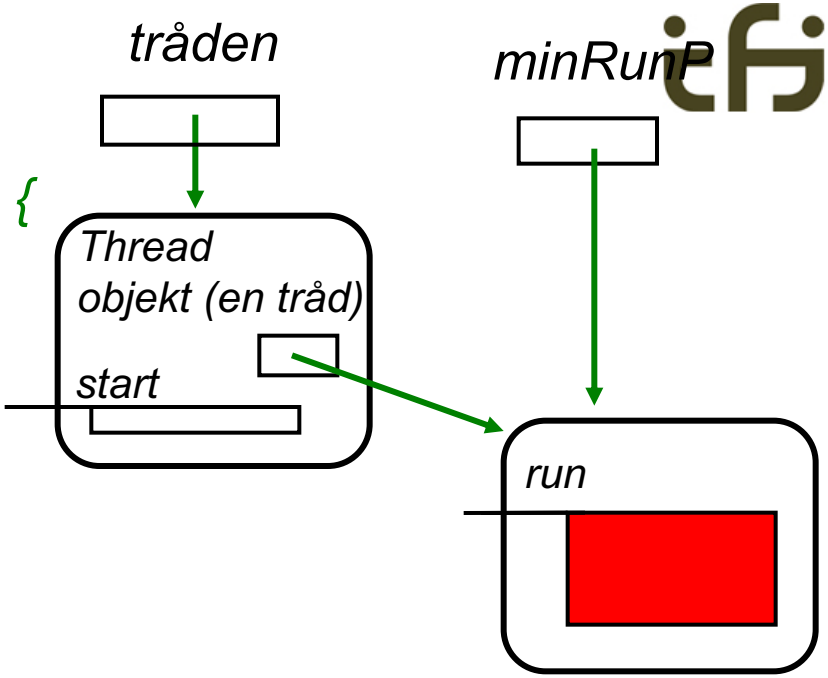
Tråder

Datamaskinarkitektur



UNIVERSITETET I OSLO

Tråder i Java



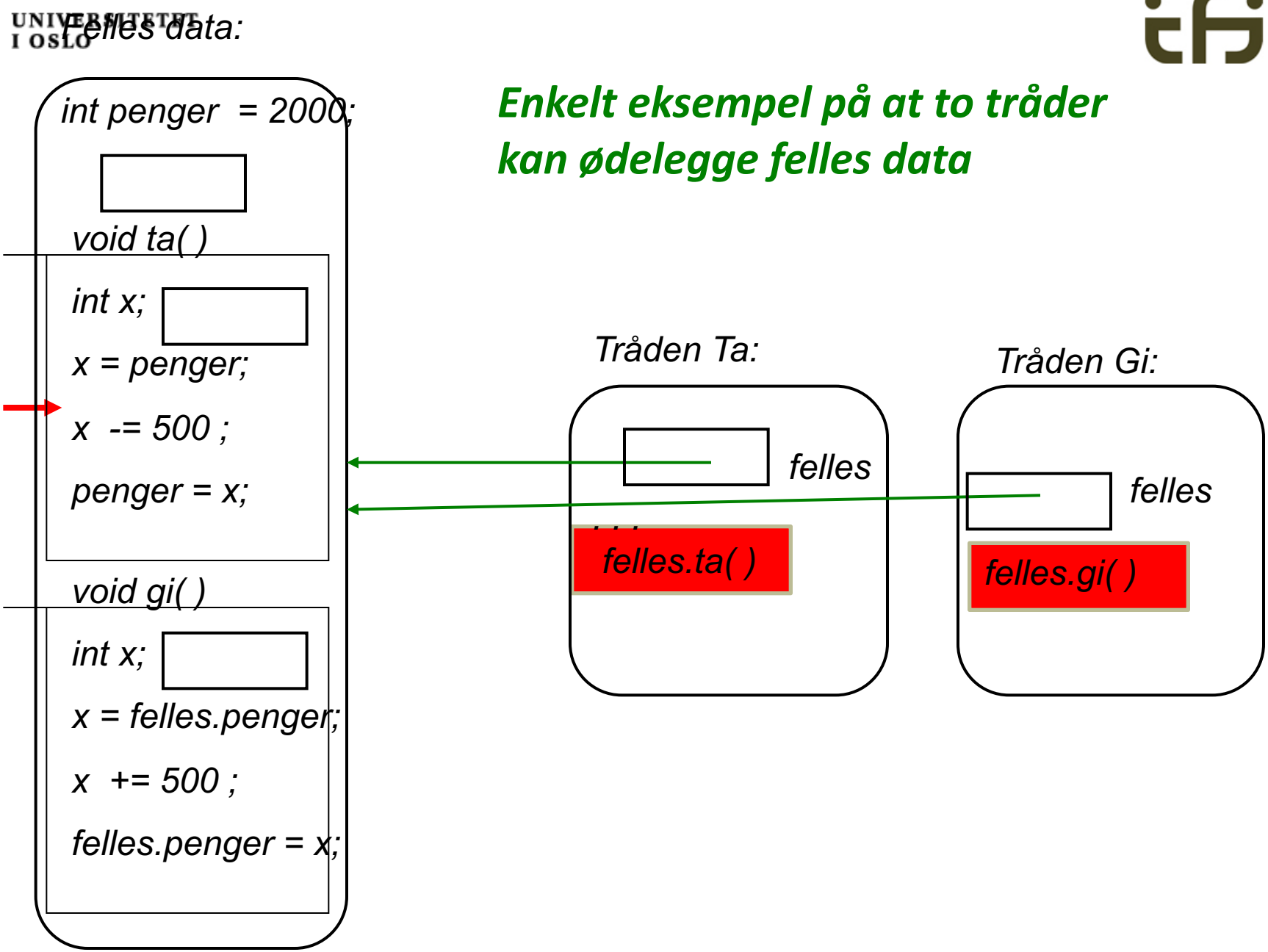

```
class MinRun implements Runnable {
    <datastruktur>
    public void run( ) {
       while (<mer å gjøre>) {
            <gjør noe>;
            . . .
       }
   }
}
```

En tråd lages og startes opp slik:

```
Runnable minRunP = new MinRun();
Thread tråden = new Thread(minRunP);
tråden.start( );
```

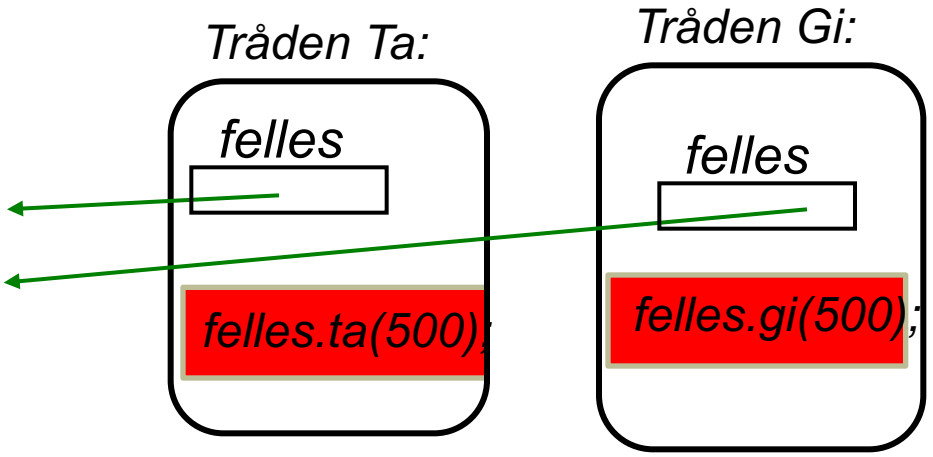
start( ) er en metode i Thread som må kalles opp for å få startet tråden. start-metoden vil igjen kalle metoden run (som vi selv programmerer).

Enkelt eksempel på at to tråder kan ødelegge felles data



Kritiske regioner / monitor

```
Lock laas = new ReentrantLock();

Int penger;  2000

public void gi ( int verdi ) throws InterrupedException
    laas.lock();
    try {
        penger = penger + verdi;
    } finally {
      laas.unlock()
    }

public void  ta ( int verdi ) throws InterrupedException
    laas.lock();
    try {
        penger = penger - verdi;
    } finally {
      laas.unlock()
    }
```



En lås (laas) slik at bare en tråd kommer inn i monitoren om gangen

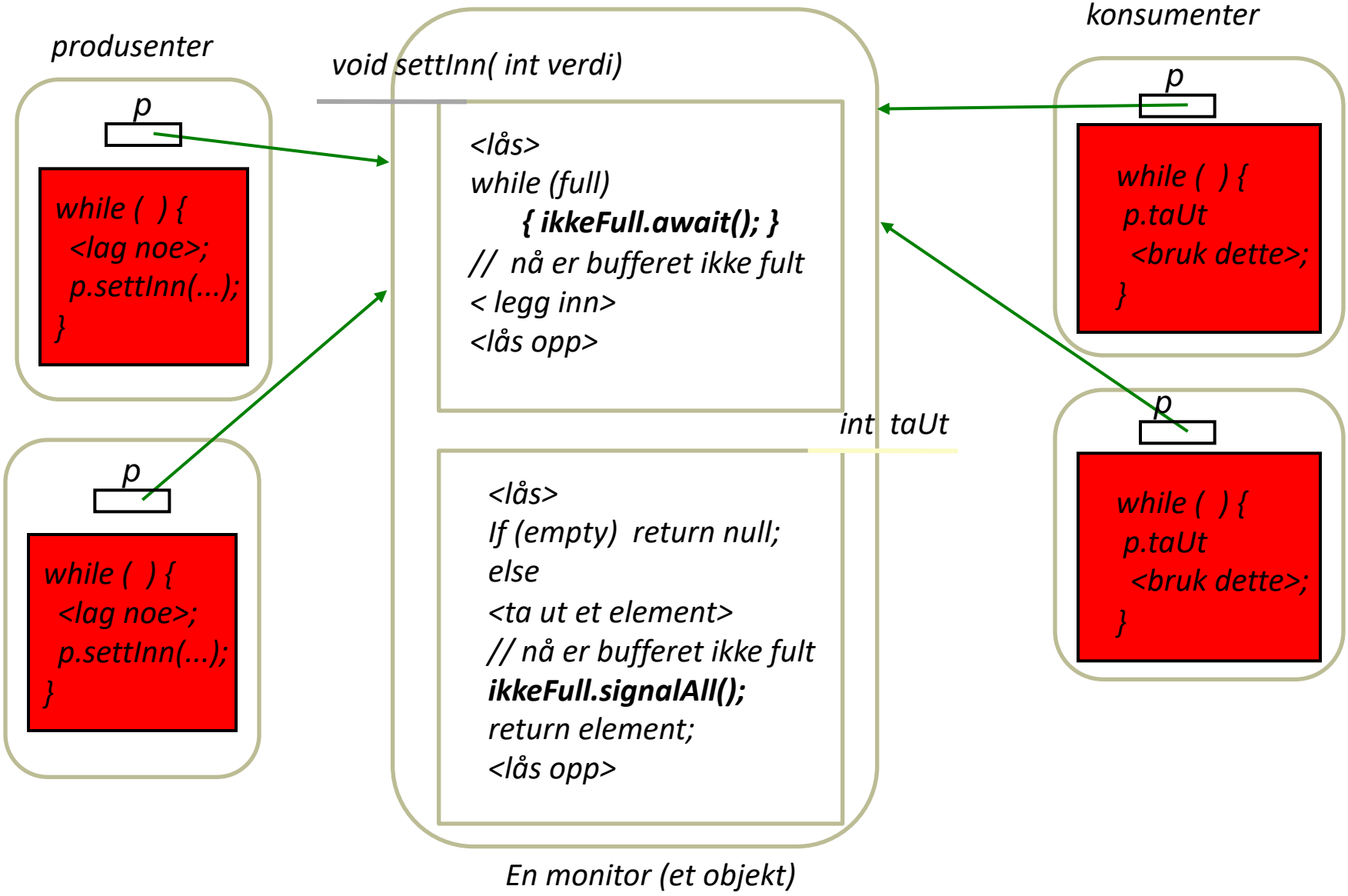
(import java.concurrent.locks.)*

Legg merke til bruken av finally

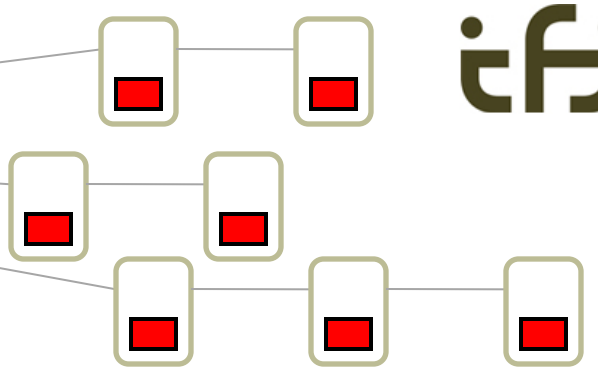
```
void  putInn (int verdi)  throws InterrupedException {
   laas.lock();
   try {
      :
      :
    } finally {
     laas.unlock();
    }
}
```

*Da blir laas.unlock() **alltid** utført !!*

Tråder: Produsenter og konsumenter



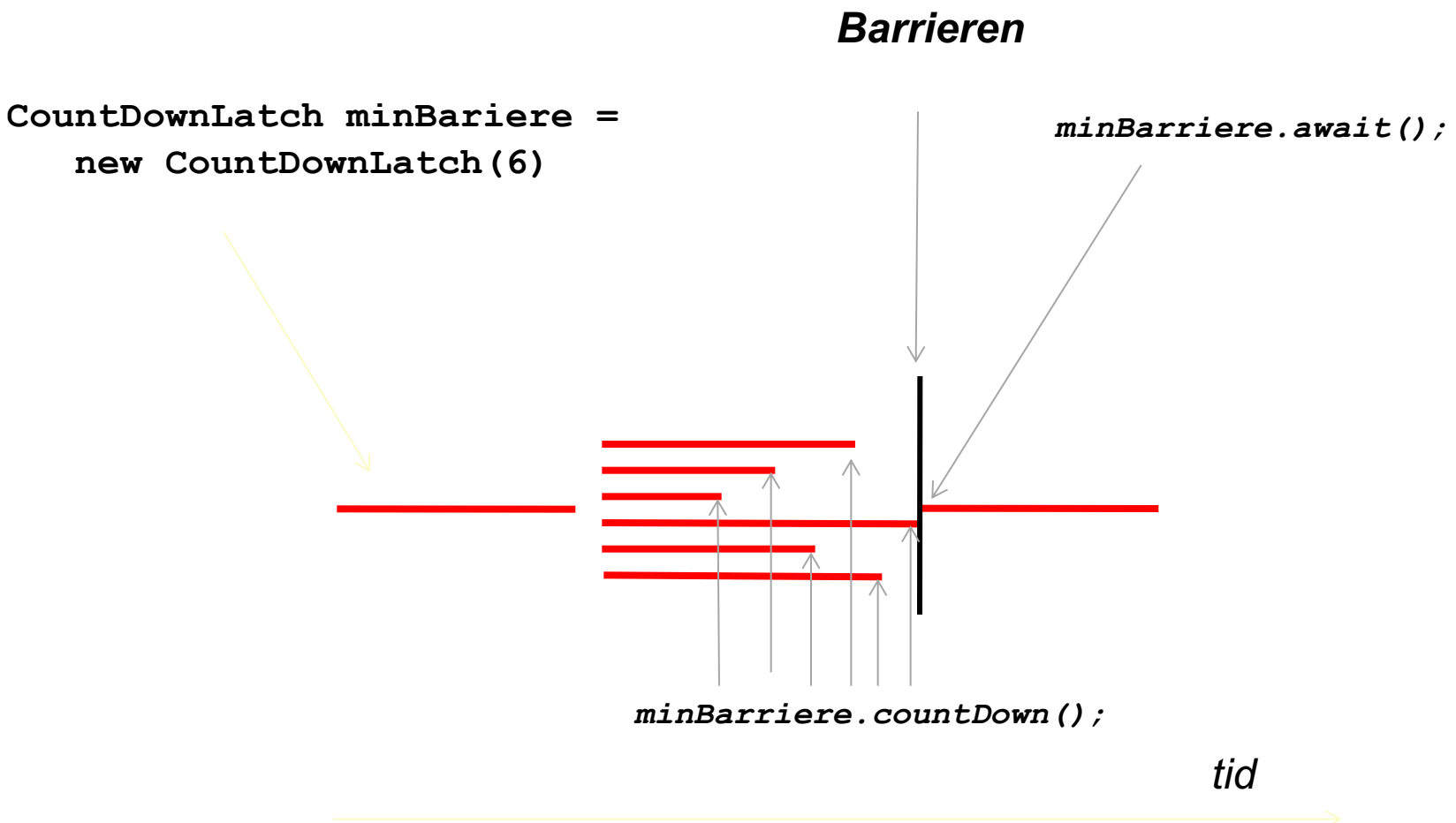
```
Lock laas = new ReentrantLock();
Condition ikkeFull =  laas.newCondition();
Condition ikkeTom = laas.newCondition();

void settInn ( int verdi) throws InterrupedException
    laas.lock();
    try {
       while  (full)  ikkeFull.await();
       // nå er det helst sikkert ikke fult
       :
        // det er lagt inn noe, så det er helt sikkert ikke tomt:
      ikkeTom.signalAll();
    } finally {
      laas.unlock();
    }

int taUt ( )  throws InterrupedException
     laas.lock();
     try {
        while  (tom)  ikkeTom.await();
        // nå er det helst sikkert ikke tomt;
        :
         // det er det tatt ut noe, så det er helt sikkert ikke fult:
       ikkeFull.signalAll();
     } finally {
       laas.unlock();
     }
```



Ofte har vi flere grunner til å vente i en monitor

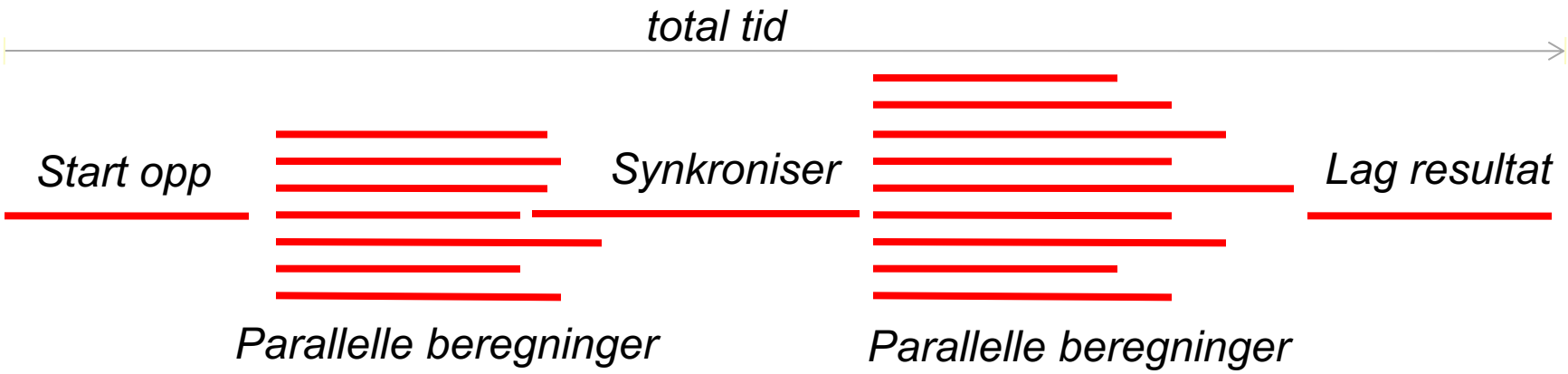
Barrierer i Java

```
import java.util.concurrent.*;
```



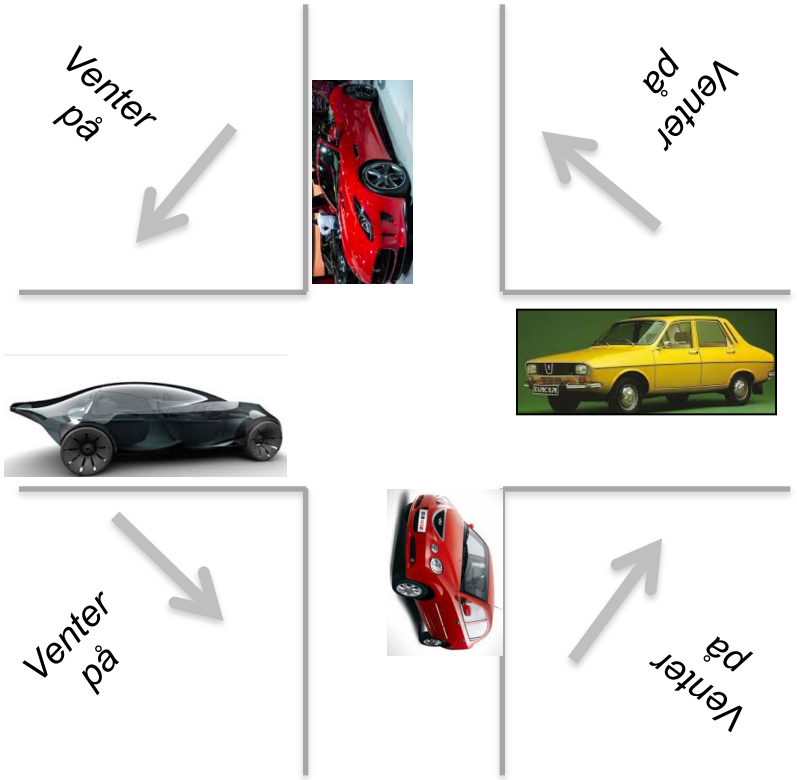
Amdahls lov

- En beregning delt opp i parallell går fortere jo mer uavhengig delene er
- **Amdahls lov**:
  - Totaltiden er
    - tiden i parallell +
    - tiden det tar å kommunisere / synkronisere/ gjøre felles oppgaver
  - Tiden det tar å synkronisere er ikke parallelliserbar (hjelper ikke med flere prosessorer)
  - Men du kan være smart og lage synkroniseringen så kort eller mellom så få tråder som mulig



Vranglås (deadlock)

- Vranglås skjer når flere tråder venter på hverandre i en sykel:
- Eksempel
  - Veikryss:
    alle bilene skal stoppe for
    biler fra høyre ->
    Alle stopper = VRANGLÅS



Enkleste eksempel på vranglås:

To biler kan risikere å vente på hverandre

Smalt veistykke, to biler kan ikke passere hverandre.
Bilene kan bare se den første delen.

49

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

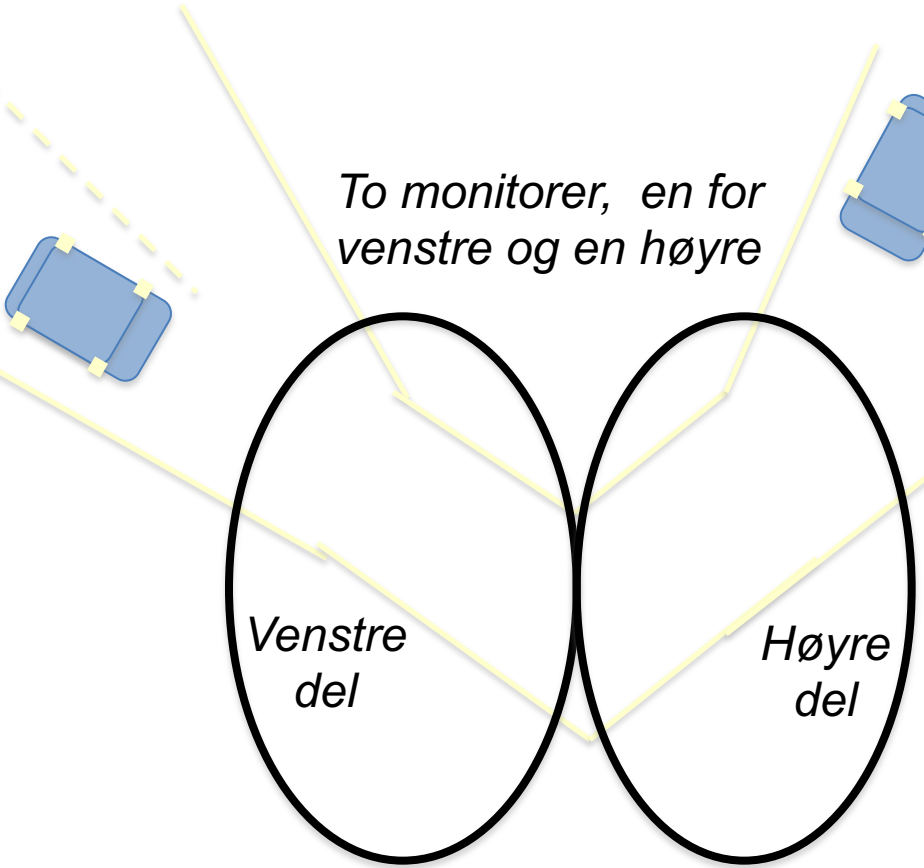
Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Program



Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

Kjør bilen

LYKKE TIL PÅ EKSAMEN !