UNIVERSITETET I OSLO

Institutt for informatikk

IN1010 våren 2021

tirsdag 11. mai

Repetisjon

Stein Gjessing

Innledning

En rask tur gjennom:

- Innkapsling
- Arv
- Datastrukturer
- Rekursjon
- Tråder
- GUI
- God programmeringsskikk
  - Testing, programmeringsmønstre, invarianter, interface

Innkapsling og arv

Innkapsling

```
public  Kaningard ()
        antall = 0;
```

```
public void settInn (Kanin en)
    if (antall < MAXANT) {
        alle [antall] = en;
        antall ++;
    }
```

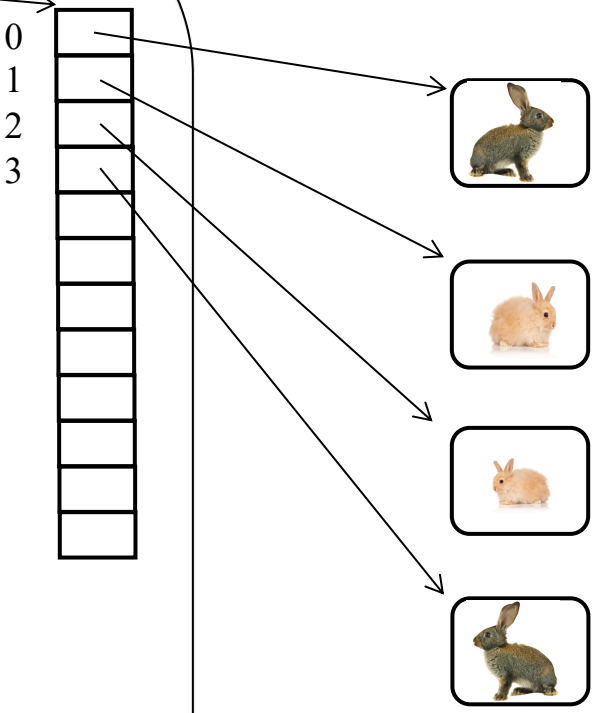
```
public Kanin   taUt( )
    if (antall > 0)  {
        antall --;
        return alle [antall];
    } else return null;
```

private Kanin [ ] alle

private Navn: antall

3

Type: int



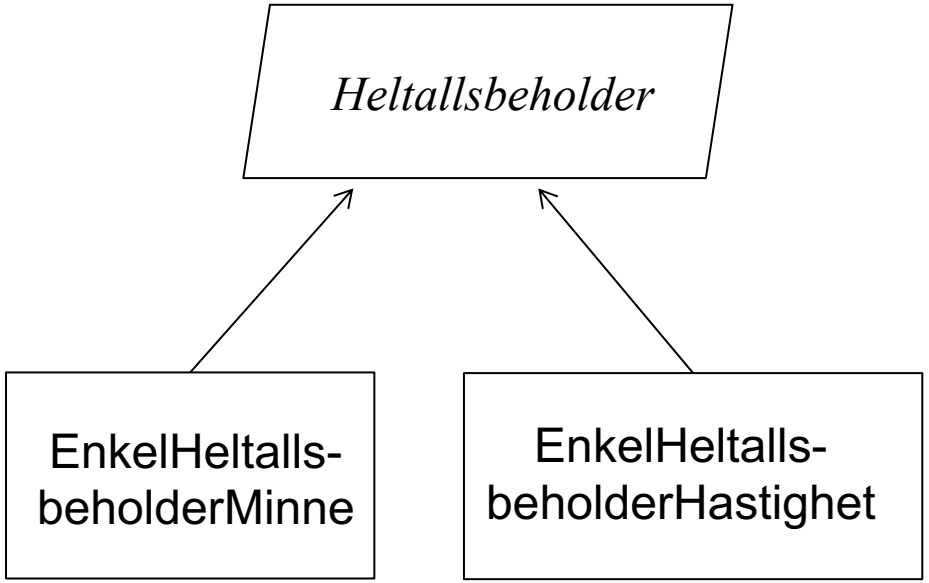
Privat og hemmelig datastruktur

som hjelper metodene til å gjøre det de skal.

Metodenes **semantikk / virkemåte**

Innkapsling

Ett interface - Flere forskjellige implementasjoner



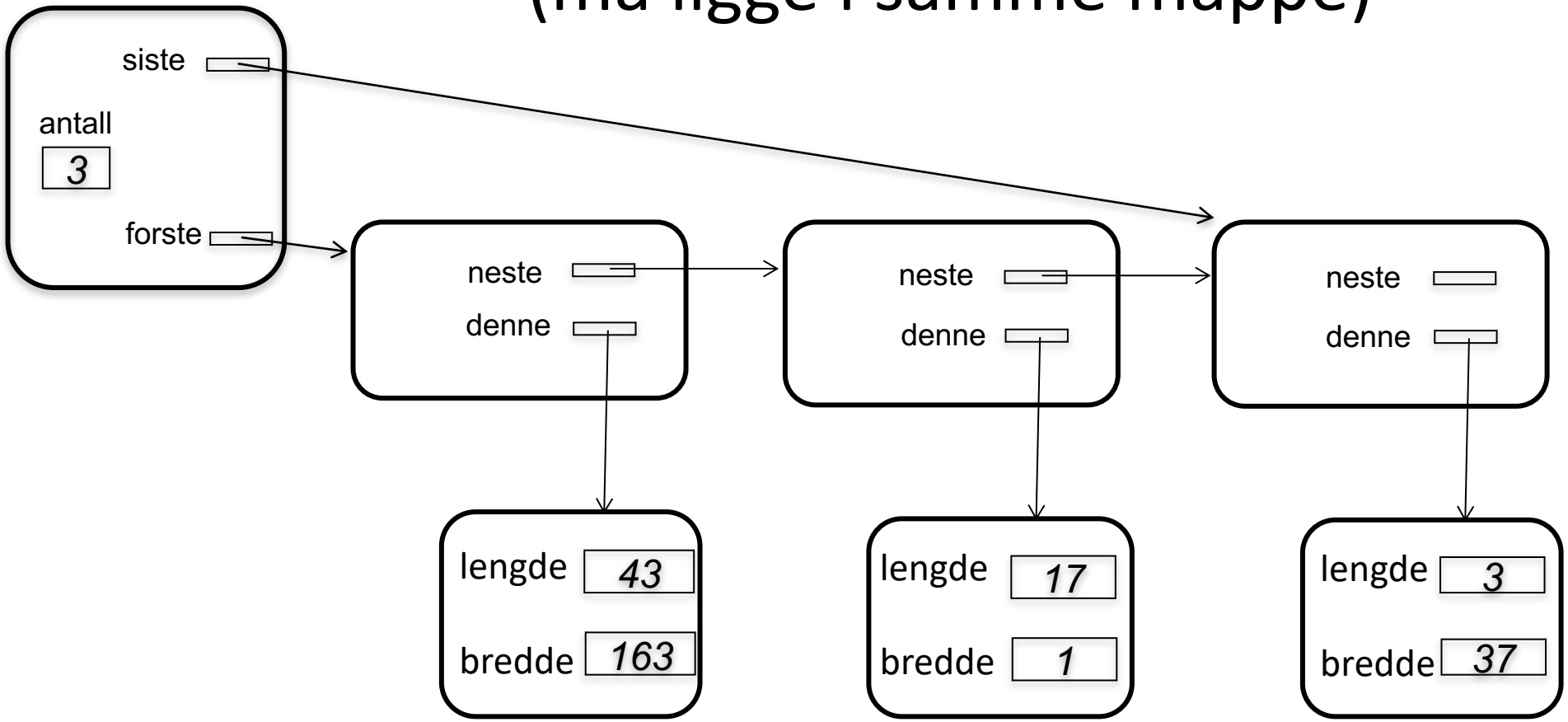

```
interface Heltallsbeholder {
    public void settInn(int tall);
    public int taUt( );
}
```

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

Innkapsling: Ikke vær for kategorisk

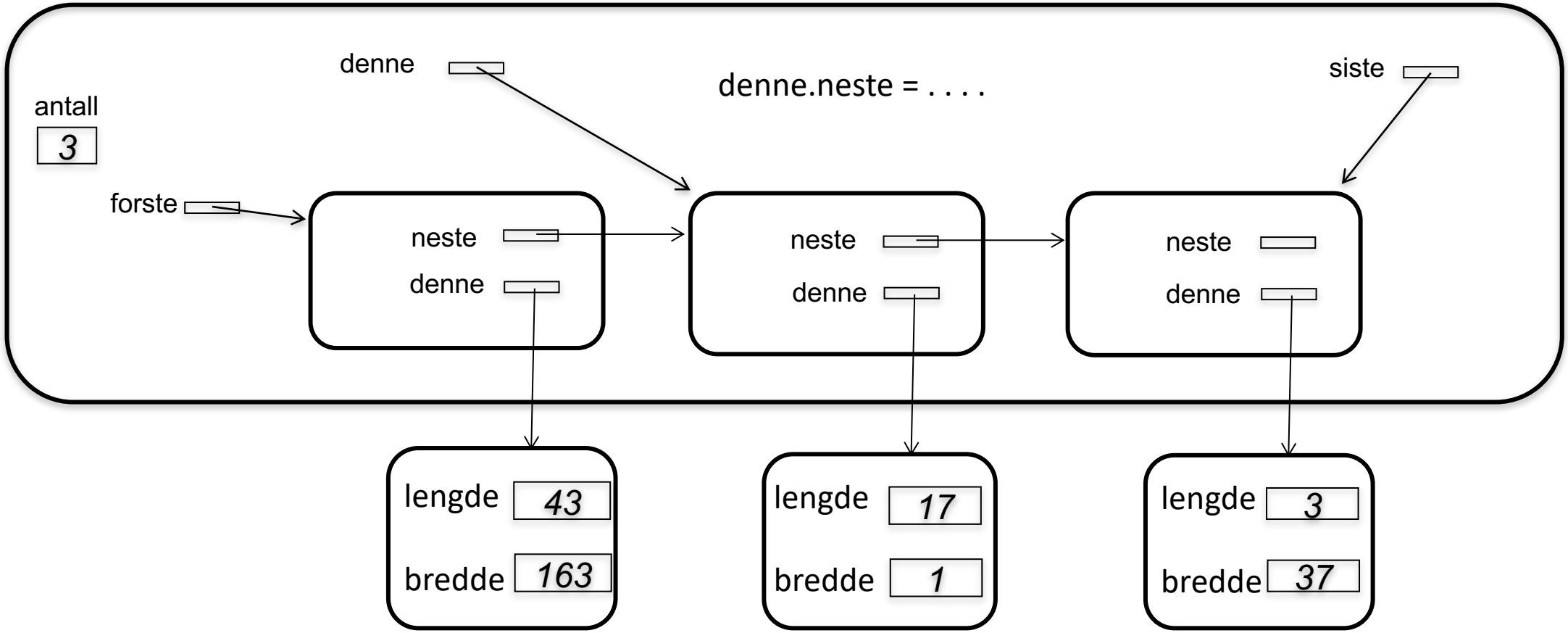
Variable i andre objekter kan noen ganger maipuleres utenifra (må ligge i samme mappe)



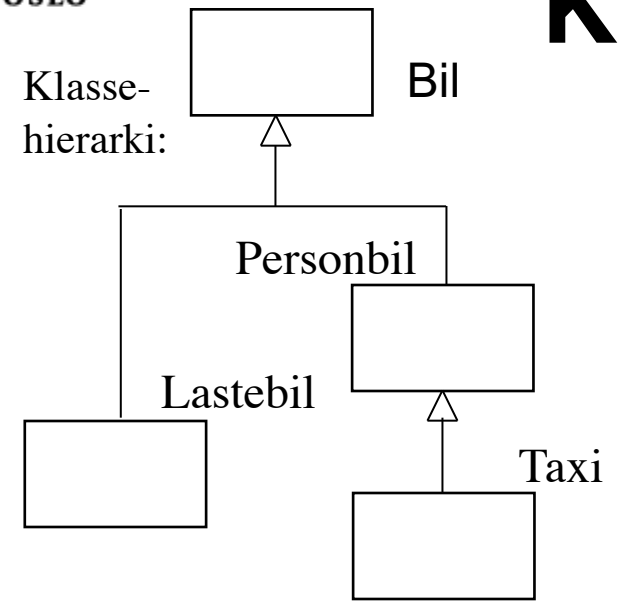
Innkapsling: Ikke vær for kategorisk

Mest opplagt med indre kasser (begge veier)

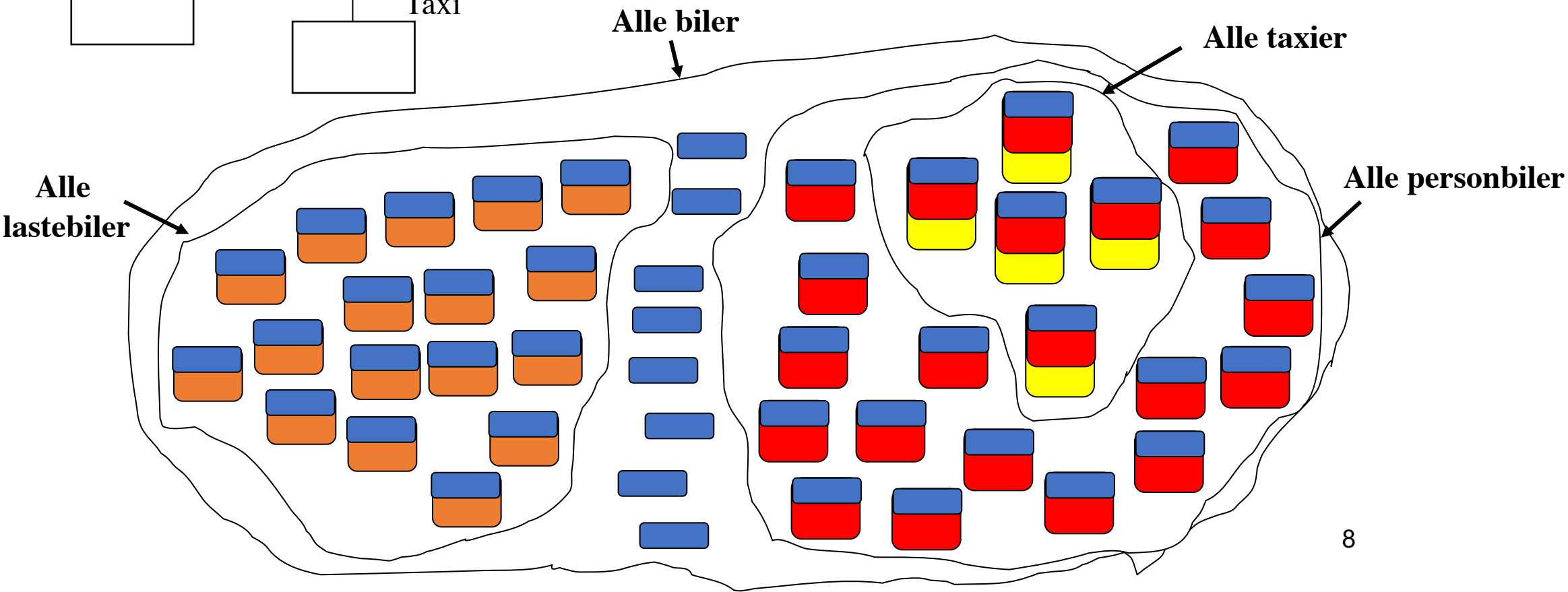
NB! I lageret ligger aldri objekter inni hverandre



Klasser - Subklasser

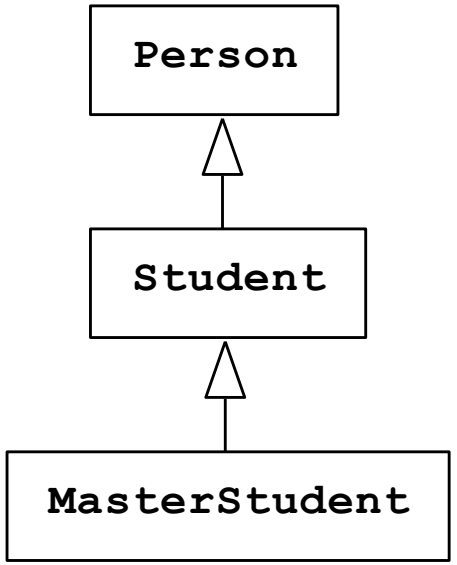


class Bil { <blå egenskaper>}
class Personbil extends Bil { <røde egenskaper> }
class Lastebil extends Bil { <brune egenskaper> }
class Taxi extends Personbil { <gule egenskaper> }



Roller - Referanser

```
pers = new MasterStudent( );
    . . .
```

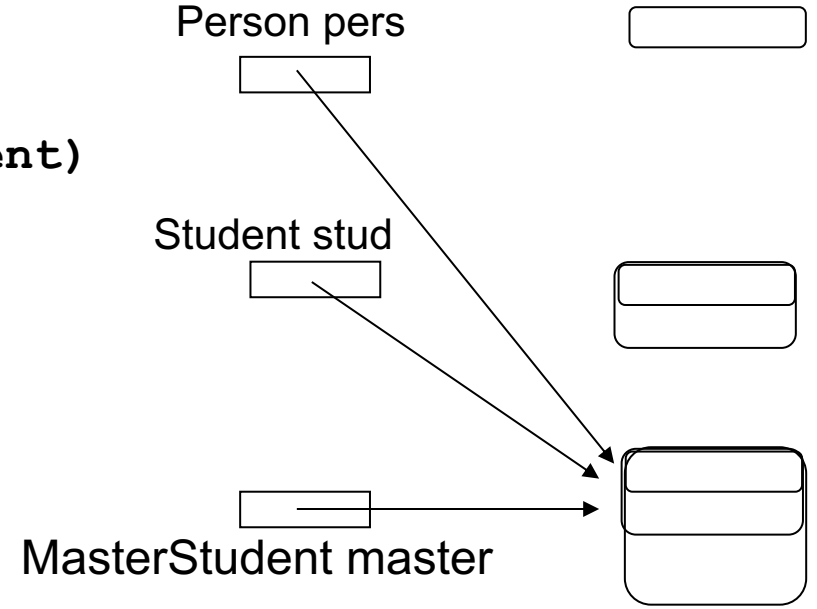


Spiller dette objektete denne rollen?

```
if (pers instanceof MasterStudent)
    ...
```

Kasting:

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



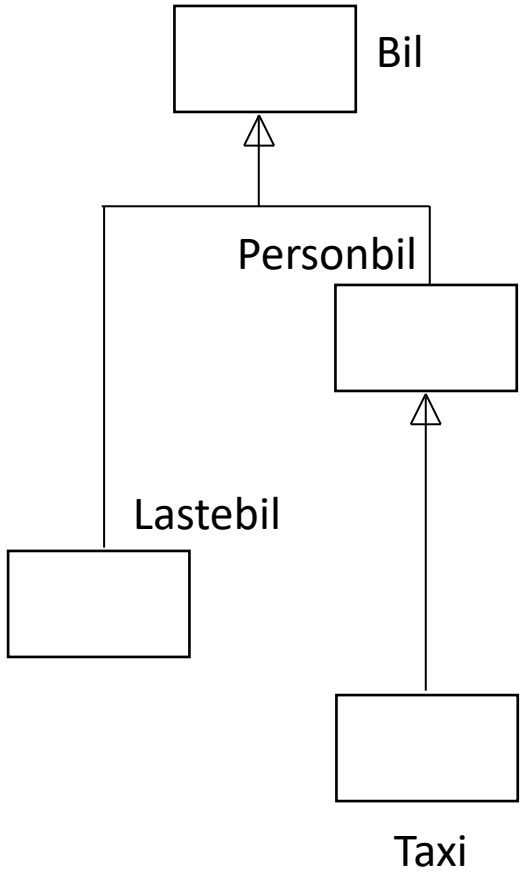
Alle objekter må spille den rollen som **typen** (klasse / interface) til referansen sier

Polymorfi
Tingen vet selv hva den er og skal gjøre


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

class Personbil extends Bil {
    protected int antallPassasjer;
    @Override
    public int skatt( ) {return pris * 2;}
}
class Lastebil extends Bil {
   protected double lastevekt;
   @Override
   public int skatt ( ) {return pris / 2;}
}
class Taxi extends Personbil {
   protected String loyveId;
   @Override
   public int skatt ( ) {return pris / 4;}
}
```

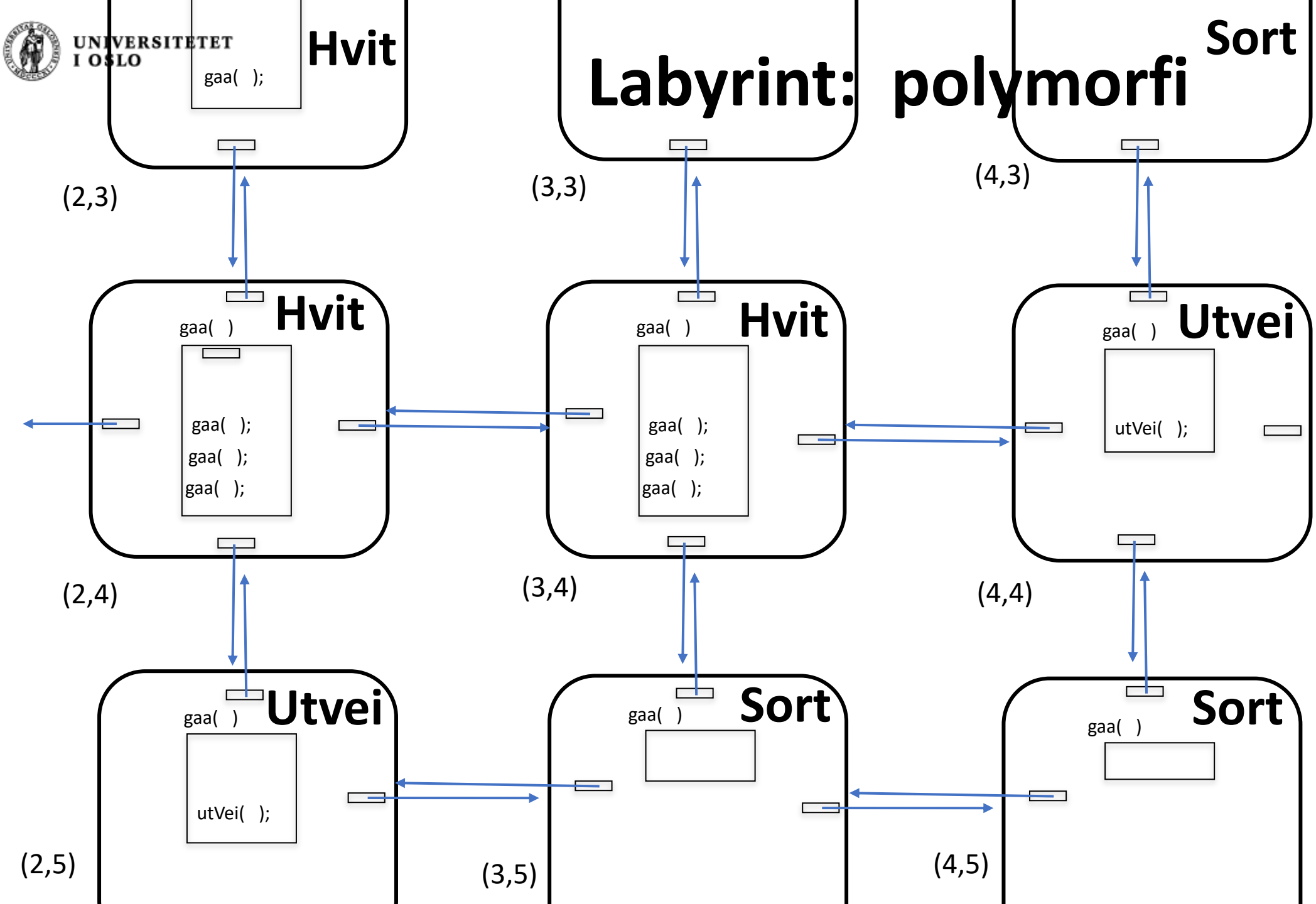
Klasse-hierarki:



UNIVERSITETET I OSLO

Labyrint: polymorfi

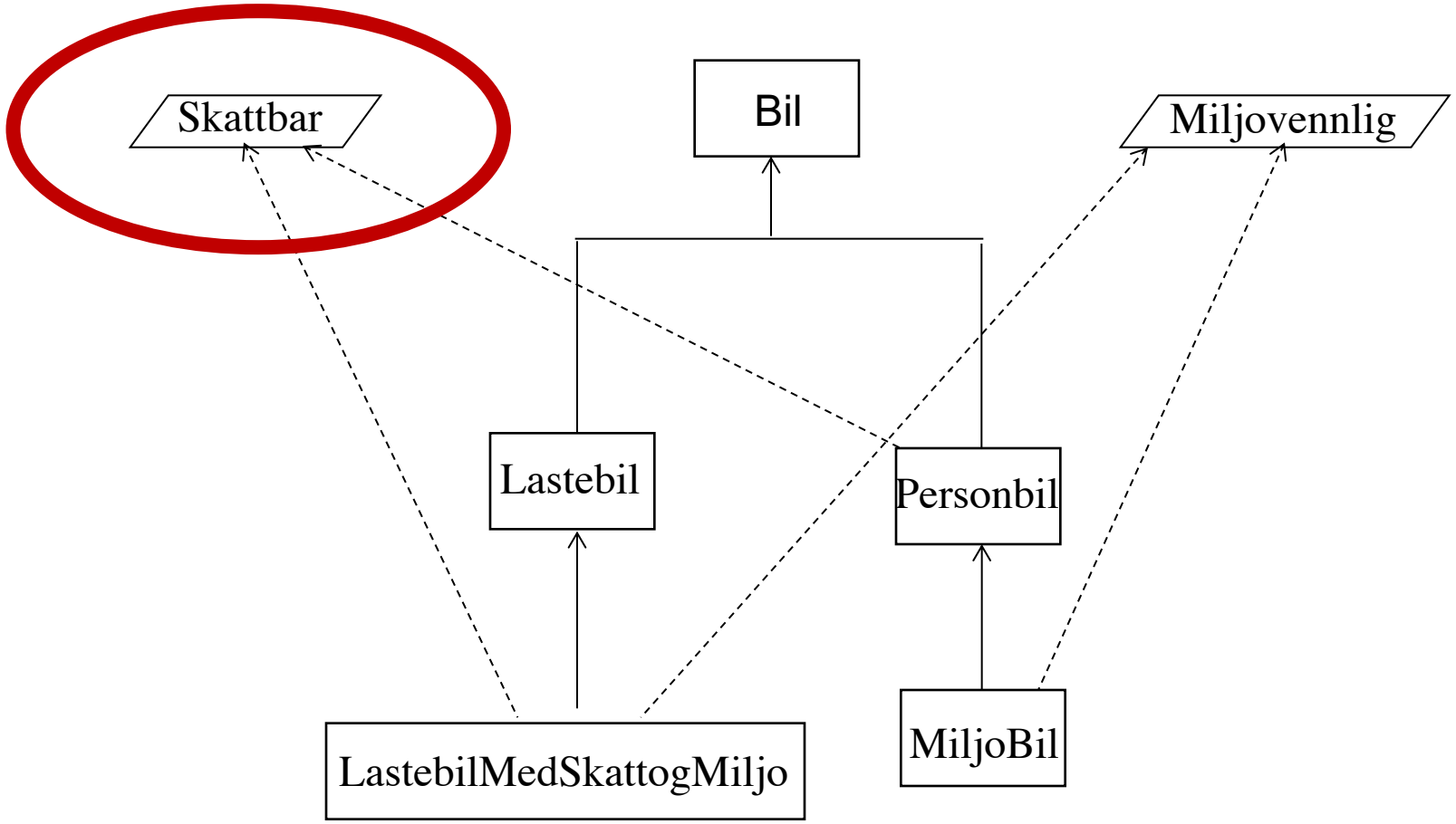
Institutt for informatikk



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();

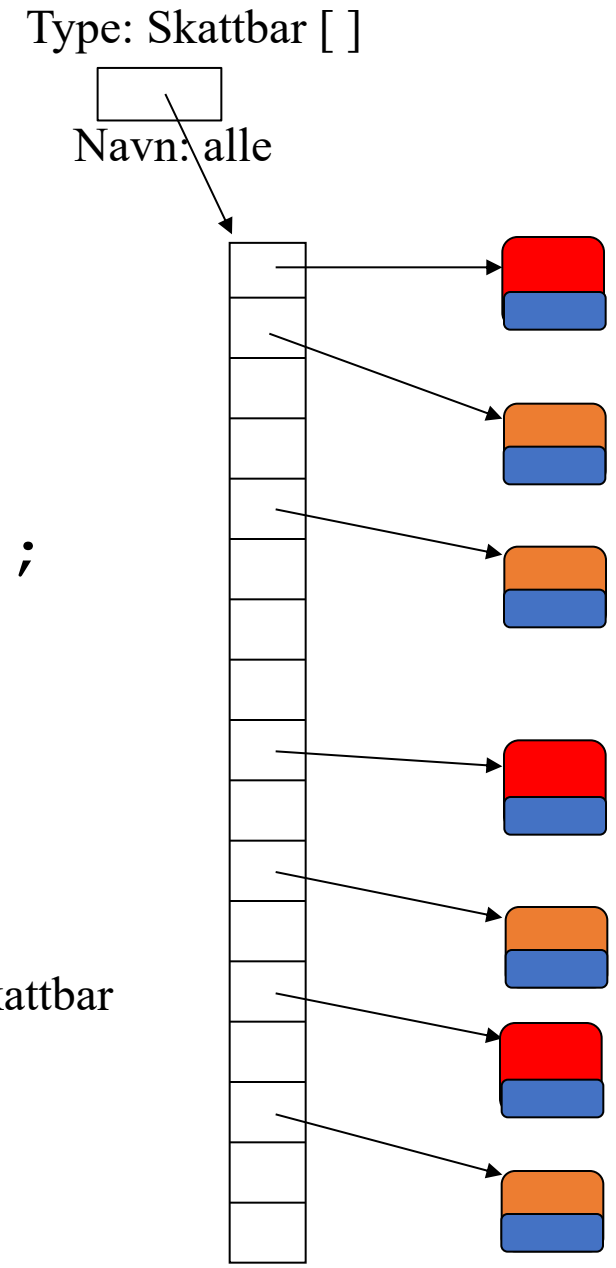
Interface - en rolle som kan spilles flere steder



Polymorfi


```
totalSkatt = 0;

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



Rollen Skattbar

Generiske klasser og generiske interface

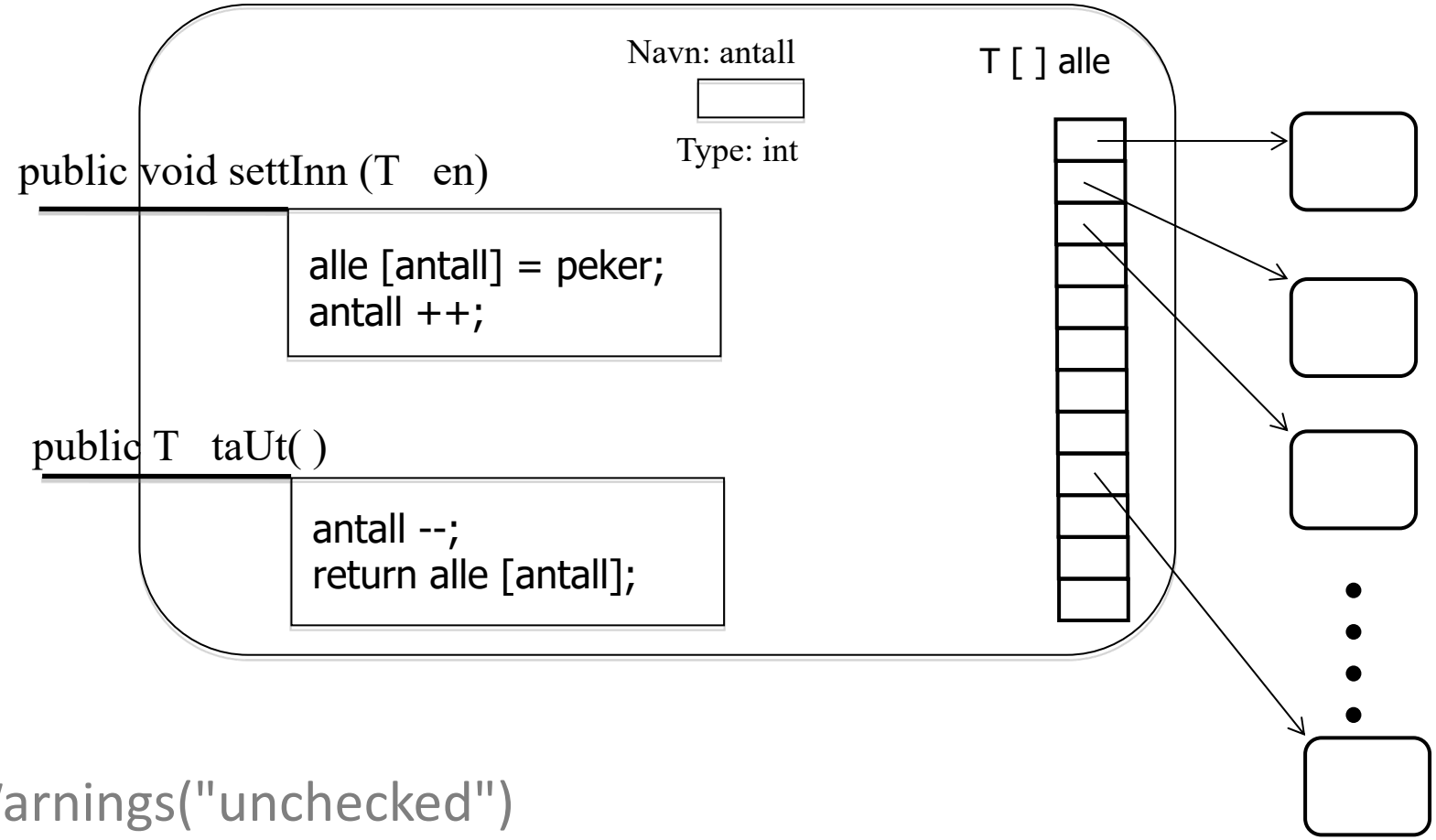
```
interface Beholder <T> {
        public void settInn (T  en);
        public T  taUt ( );
}


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

GeneriskBeholderTilEn <Bil> minGarasje = new GeneriskBeholderTilEn <Bil>();
```

Husk: interface Comparable<T>

Generiske klasser



Husk:

```
@SuppressWarnings("unchecked")
private T [ ] alle = (T [ ] ) new Object [100];
```

Datastrukturer

Datastrukturer: arrayer


```
public Kaningard ()
    antall = 0;

public void settInn (Kanin en)
    if (antall < MAXANT) {
        alle [antall] = en;
        antall ++;
    }

public Kanin taUt( )
    if (antall > 0) {
        antall --;
        return alle [antall];
    } else return null;
```

private Kanin [ ] alle

private Navn: antall

3

Type: int

0
1
2
3
MAXANT - 1




First Inn First Out - FIFO , Last Inn First Out - LIFO – Stakk, . . .

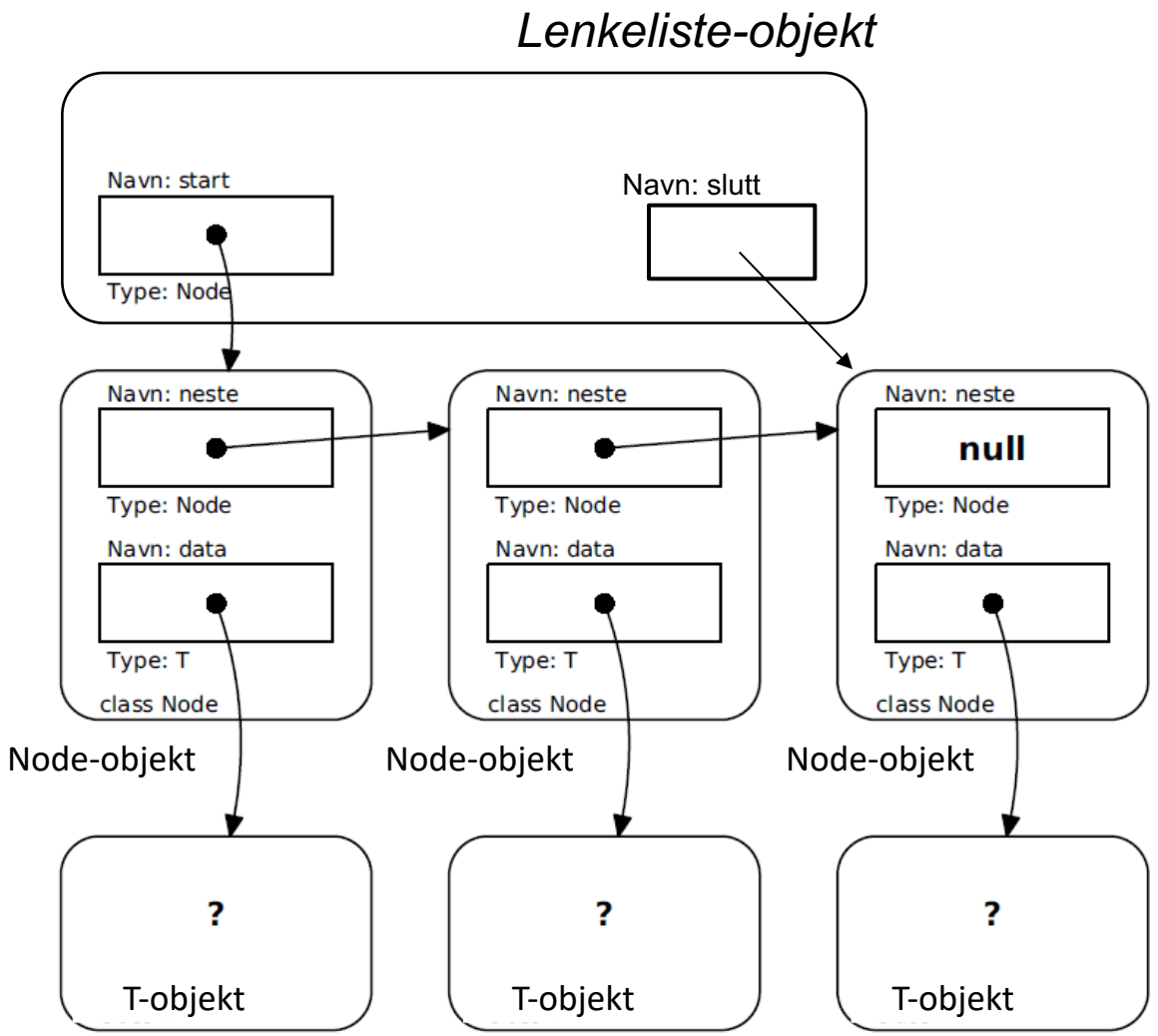
Datastrukturer: Enkel-kjedet liste

Start / første
slutt / siste

Må ta ut foran

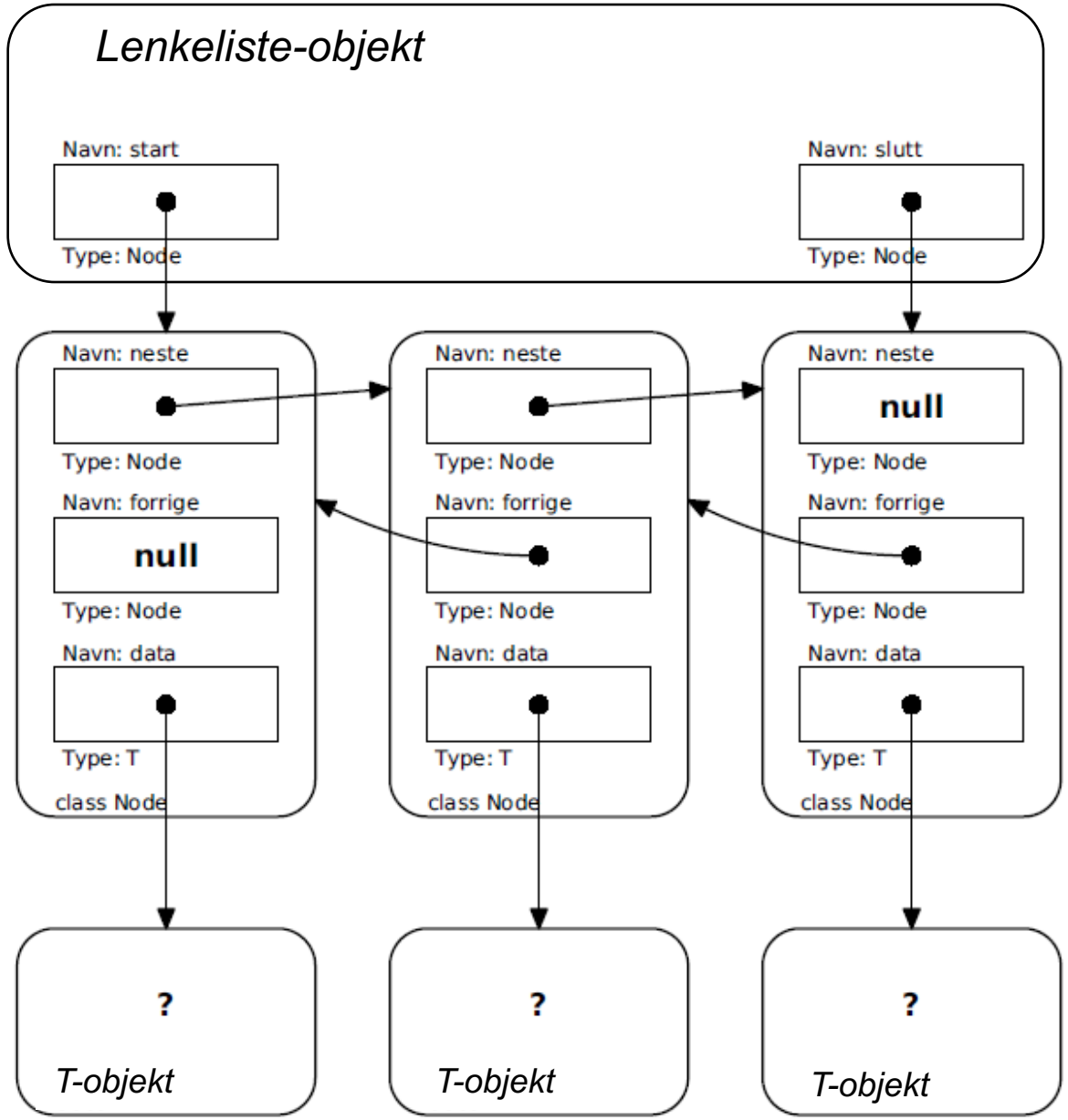
Sett inn foran = stakk
Sett inn bak = FIFO

Husk:
Datastruktur-
tegninger

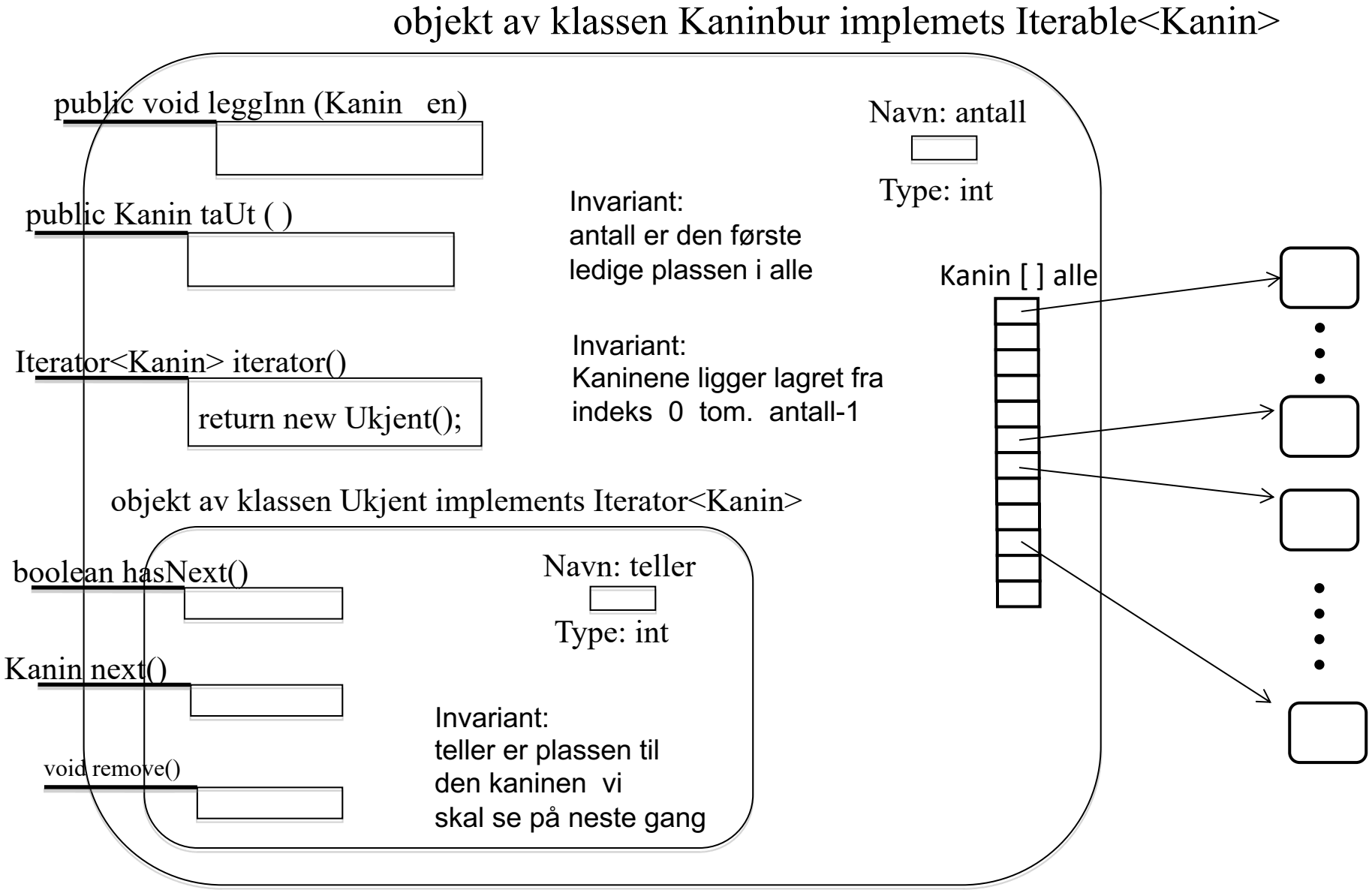


Toveisliste

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



Grensesnittene Iterable og Iterator



Rekursjon

Rekursjon: Ikke tenk for mye på det:

Just do it

Hanois tårn

Flytt 5 brikker fra 1 til 3:

Flytt 4 brikker fra 1 til 2

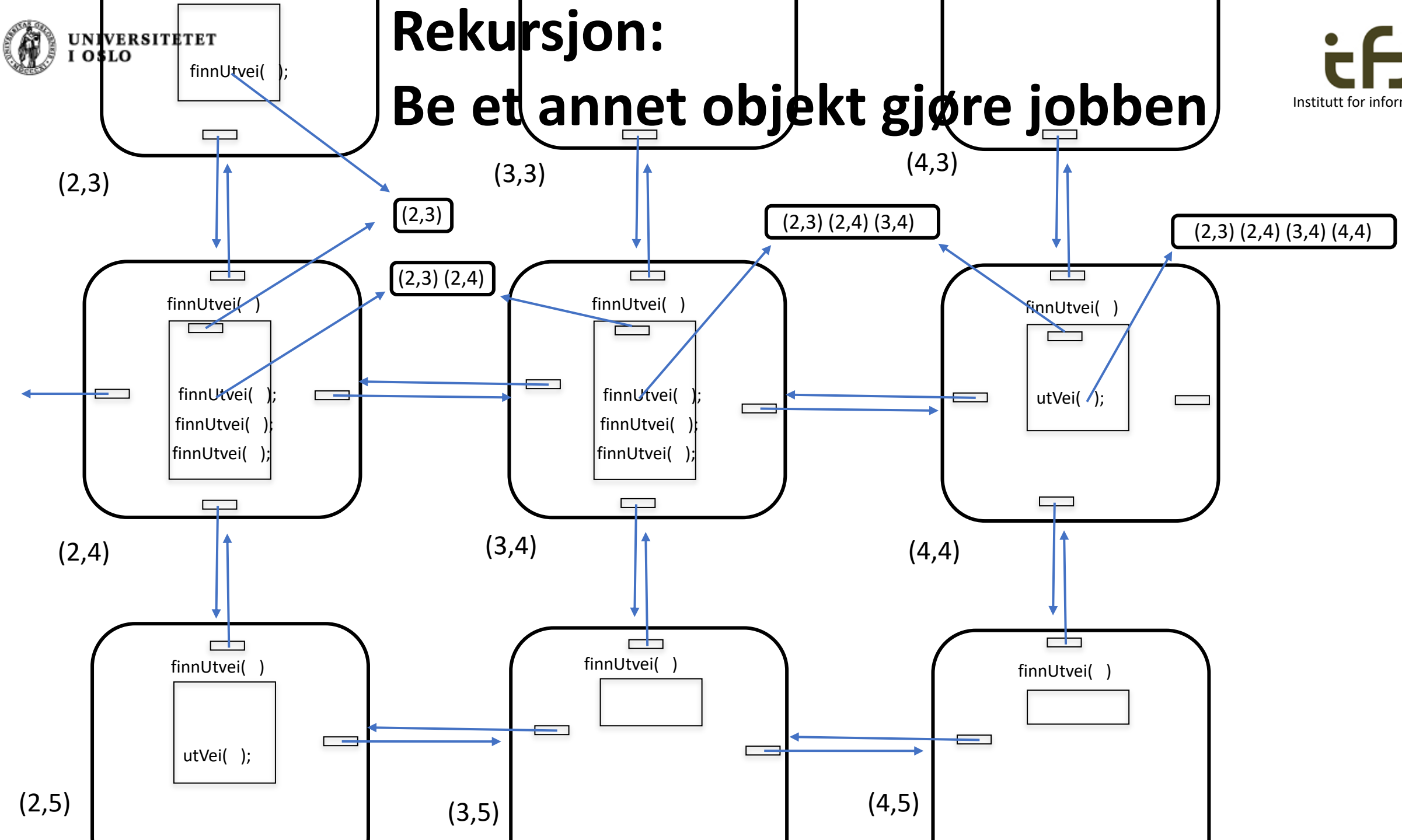
Flytt én brikke fra 1 til 3

Flytt 4 brikker fra 2 til 3

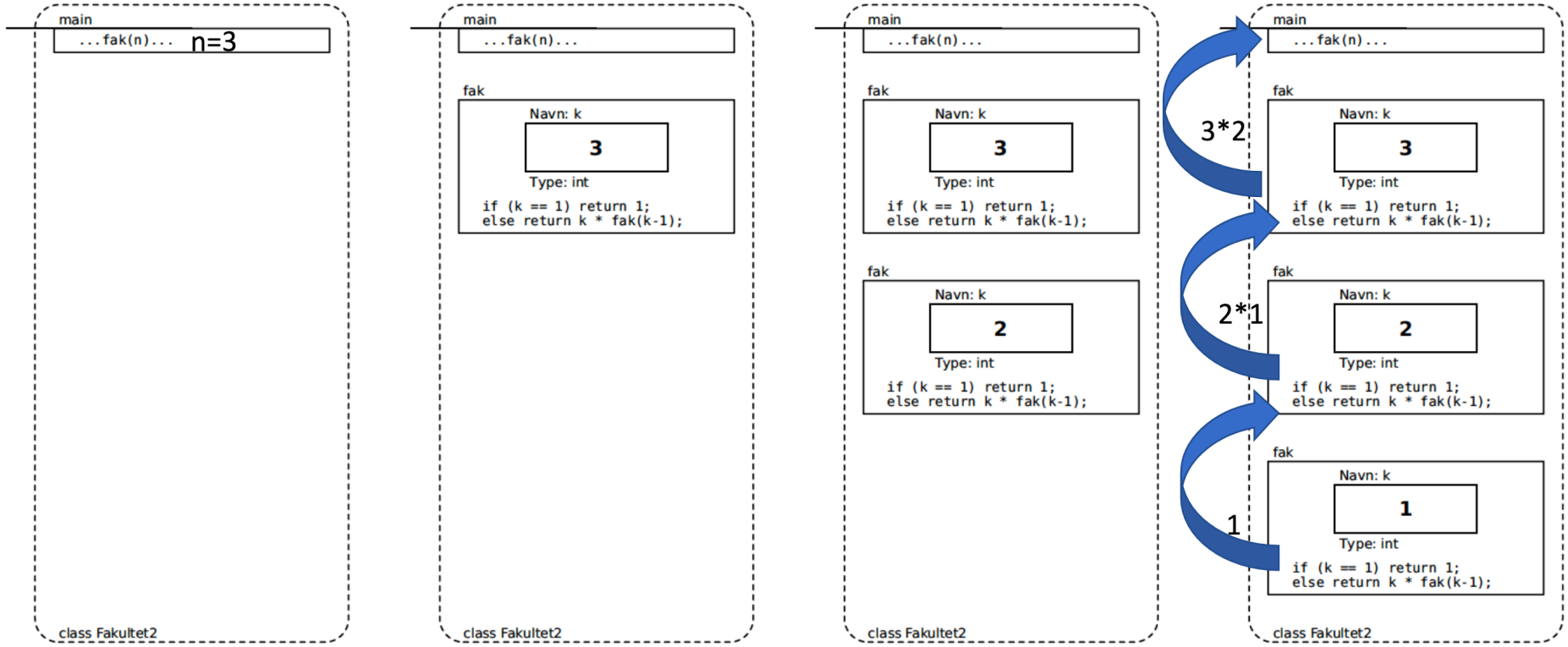
Invariant:
Små brikker ligger alltid over store

5 -> N, 4 -> N-1, . . . Flytte én brikke klarer vi

Rekursjon:
Be et annet objekt gjøre jobben



Rekursjon kan lage flere metode-instanser i samme objekt eller som her: klasse-datastruktur



Fra Dag Langmyhr

Hendelseshåndtering / GUI

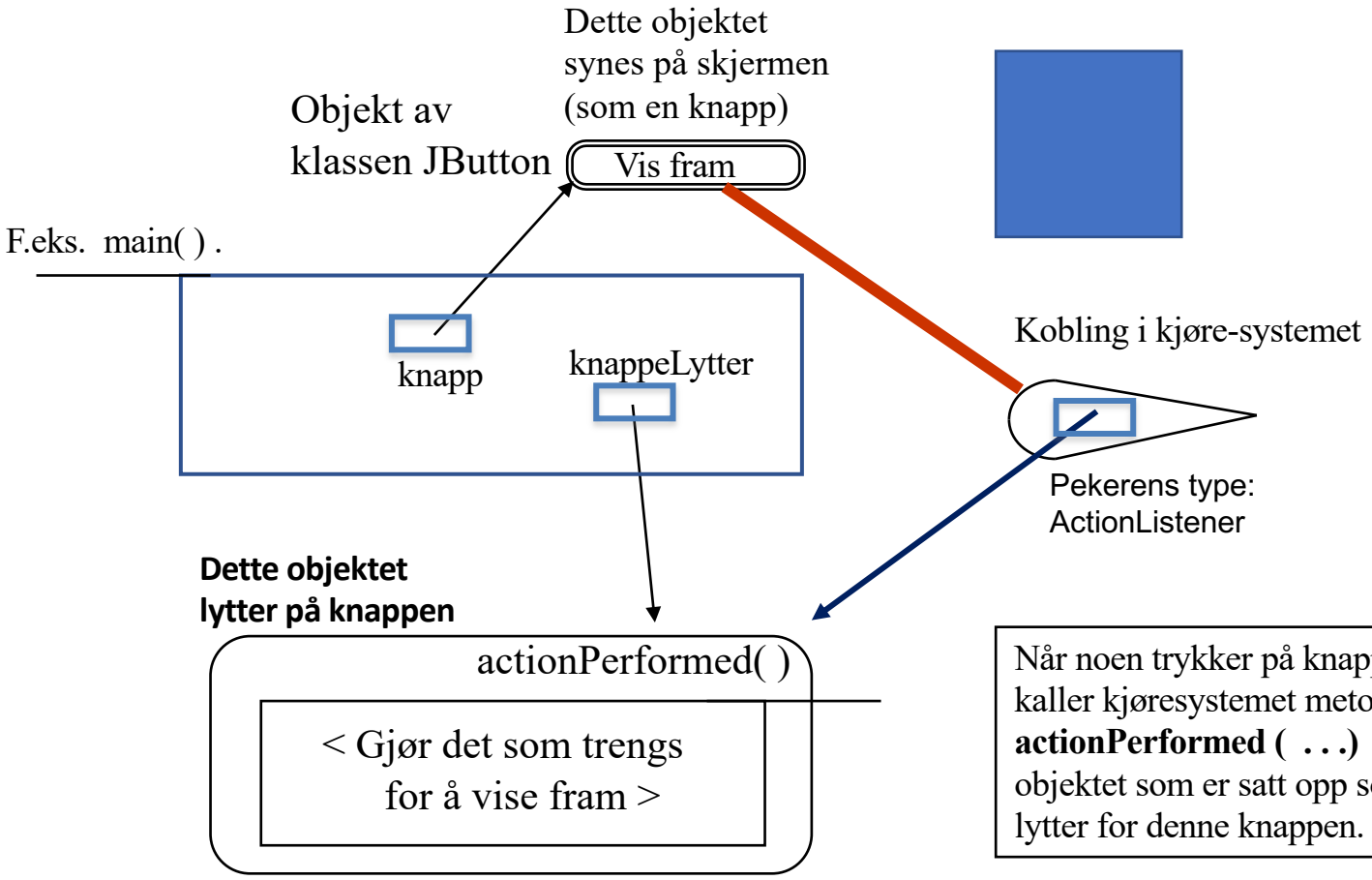
Hendelseshåndtering / GUI


```
JFrame vindu = new JFrame("Mitt vindu");
vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
vindu.add(panel);
```

```
class Lytter implements ActionListener  {
      @Override
       public void actionPerformed (ActionEvent e) {
              <Gjør det som trengs>
       }
}
```

```
 JButton knapp = new JButton(”Vis fram");
 Lytter knappeLytter = new Lytter ();
 knapp.addActionListener(knappeLytter);
panel.add(knapp);
```

Hendelseshåndtering / GUI




```
. . . .

knapp = new JButton (   );

knappeLytter = new Lytter (   );

knapp.addActionListener(knappeLytter);


panel.add(knapp);

. . . . .
```

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

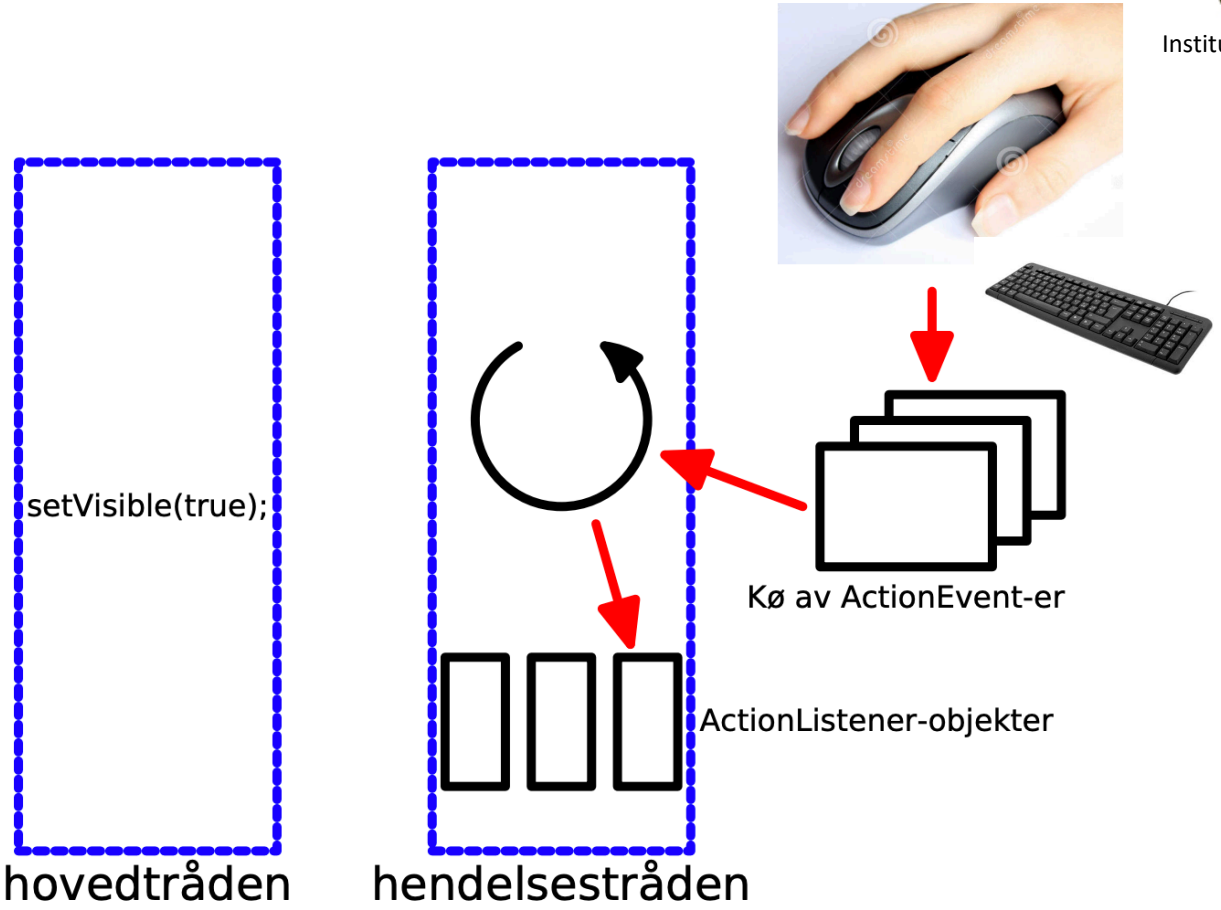
class Lytter implementerer grensesnittet ActionListener

For flere detaljer se Dag Langmyhrs programmer

Håndtere hendelser

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

Hendelsesløkken tar ActionEvent-ene etter tur, og den korrekte ActionListener vil bli kalt.

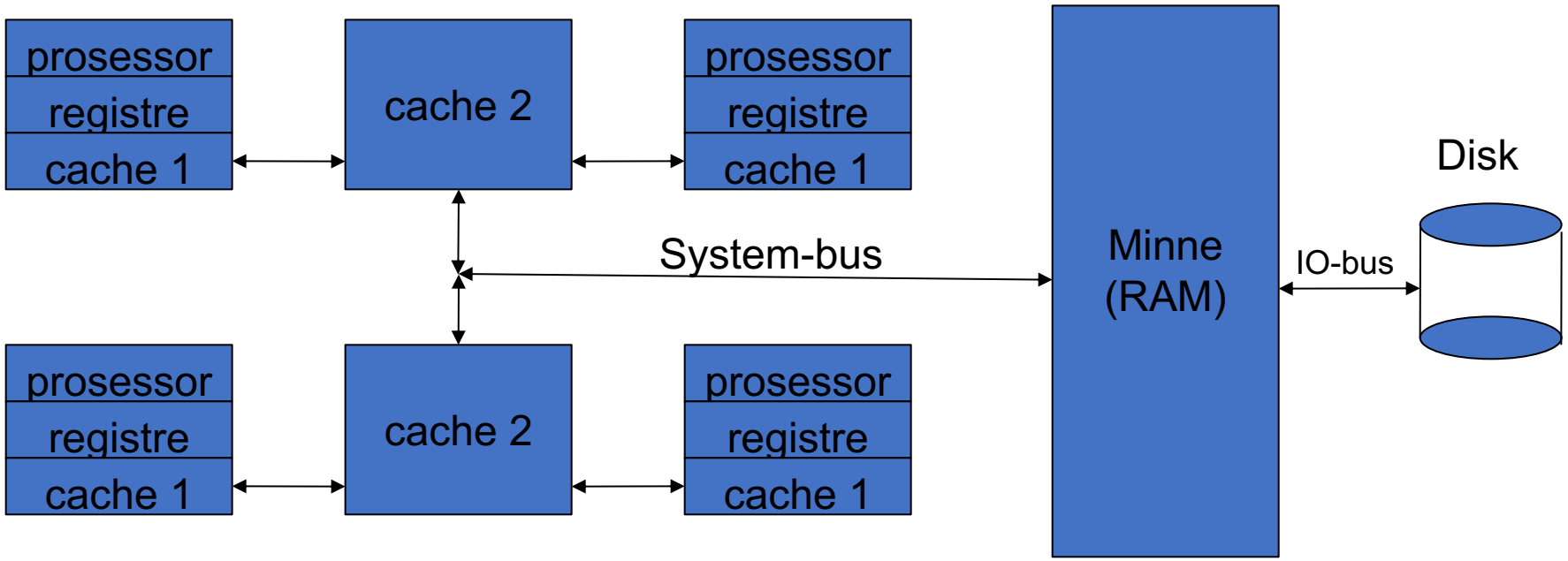


GUI-tråden

Tråder

Tråder

Datamaskinarkitektur

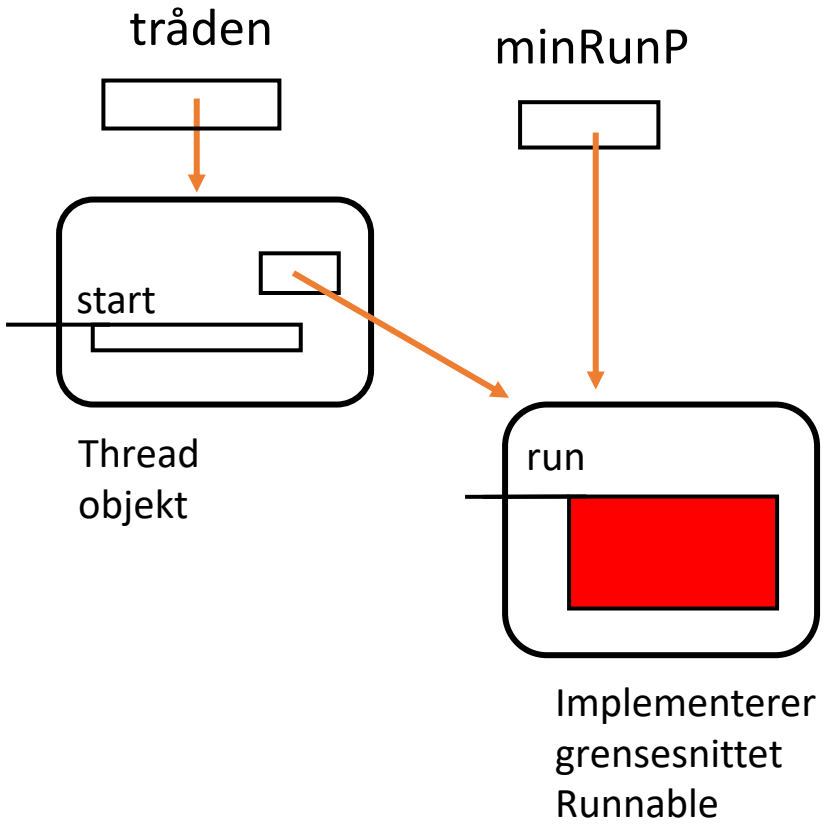


Tråder i Java


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

En tråd lages og startes opp slik:

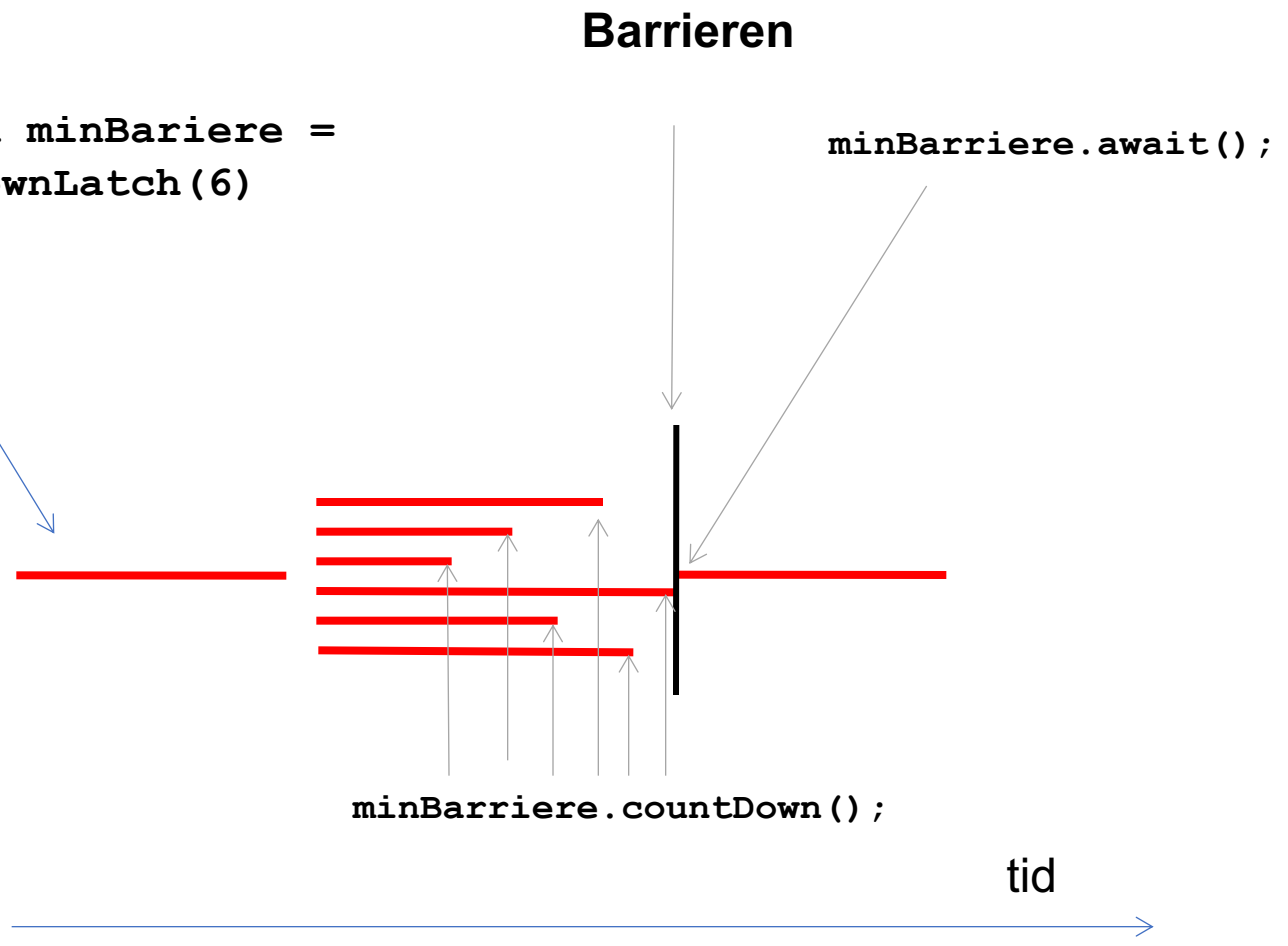
```
Runnable minRunP = new MinRun();
Thread tråden = new Thread(minRunP);
tråden.start( );
```



Barrierer i Java


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

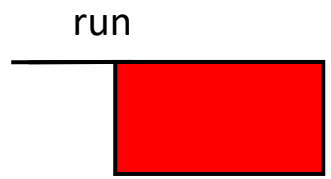
I trådene:



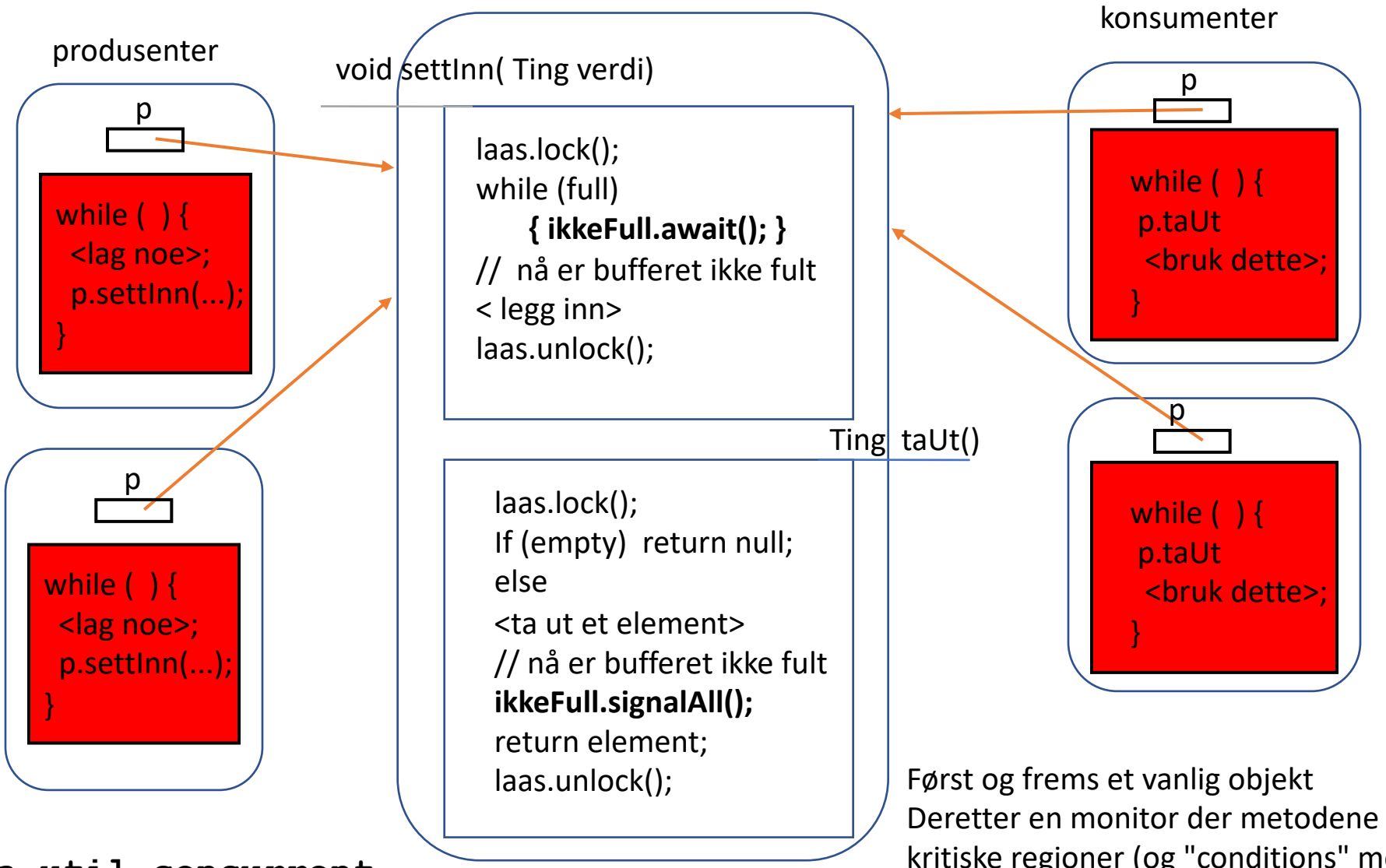
Monitorer og tråder – oppsummering

- I en monitor (som regel en felles buffer) kan programmet
  - låse og låse opp alle metoder (kritiske regioner)
  - utføre condition.wait(); som en tråd må vente på INNE I EN MONITOR (og si condition.signal();)

- I en tråd kan programmet
  - kalle på metoder i monitoren
  - vente med Thread.sleep()
  - vente på at én annen tråd terminerer: trad.join()
  - si fra at jeg terminerer med barriere.countDown()
  - vente på en barriere: barriere.await()
    - Ofte i main eller et sted der mange tråder er startet



Tråder: F.eks. produsenter og konsumenter



Først og frems et vanlig objekt
Deretter en monitor der metodene er kritiske regioner (og "conditions" med venting)

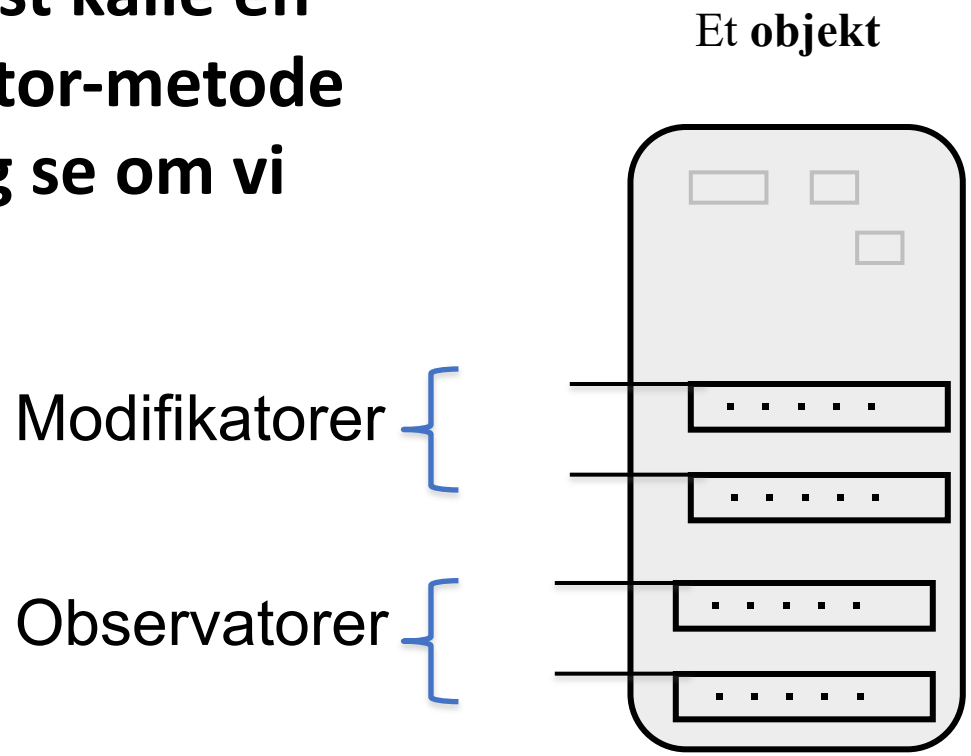
```
import java.util.concurrent
```

Programmeringsmetodologi

God programmeringsskikk

Enhetstesting

- **Når vi skal teste et objekt kan vi først kalle en observator-metode, så en modifikator-metode og så igjen observator-metoden og se om vi observerer det ønskede resultatet**



Eyvind: Testbarhet og SOLID

- Single responsibility principle
  - **En klasse bør bare ha ett ansvarsområde**
- Open/closed principle
  - Komponenter bør være åpne for utvidelser, men lukket for modifikasjoner.
  - **Lag superklasser som verktøy (med private instansvariabler)**
- Liskov substitution principle
  - **Bruk interface og polymorfi**
- Interface segregation principle
  - **Bruk små interface**
- Dependency inversion principle
  - **Bruk interface**

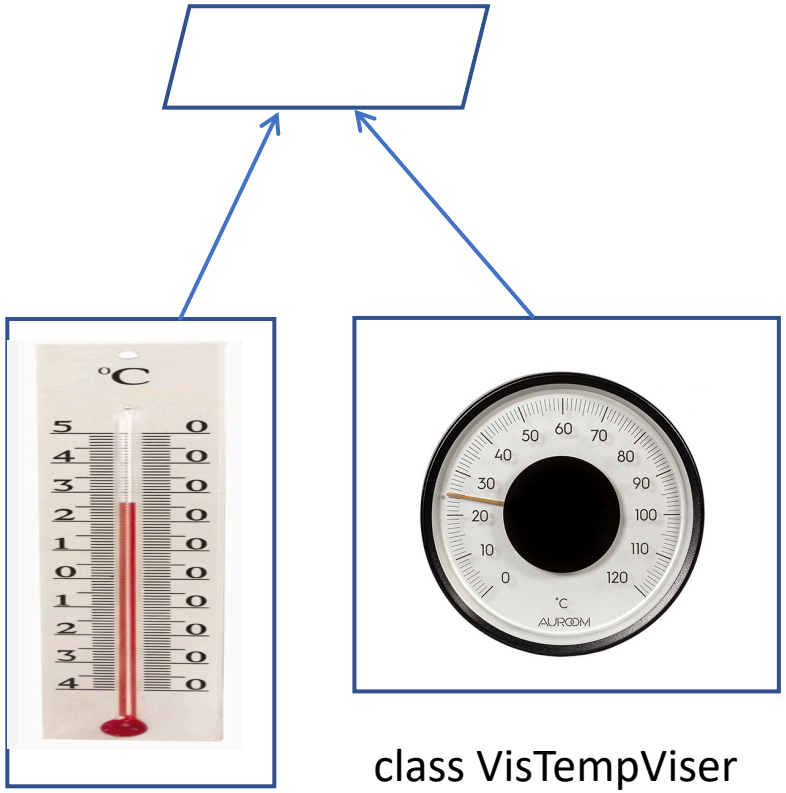
Eric: Patternes

- Kommandoløkke
- GUI (Hendelseshåndtering - ActionListener . . .)

- Model – View (– Control)
- Observer

Observer pattern: Klassehierarki

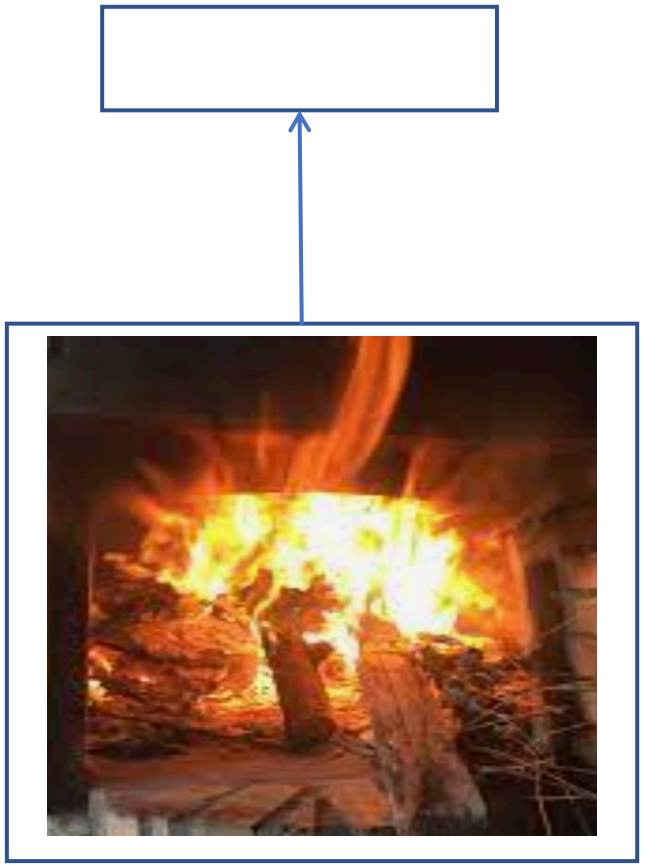
interface Observer



class VisTempSoyle

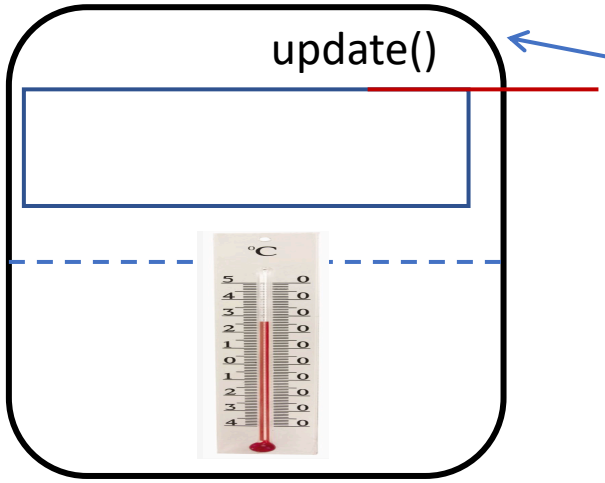
class VisTempViser

class Observable

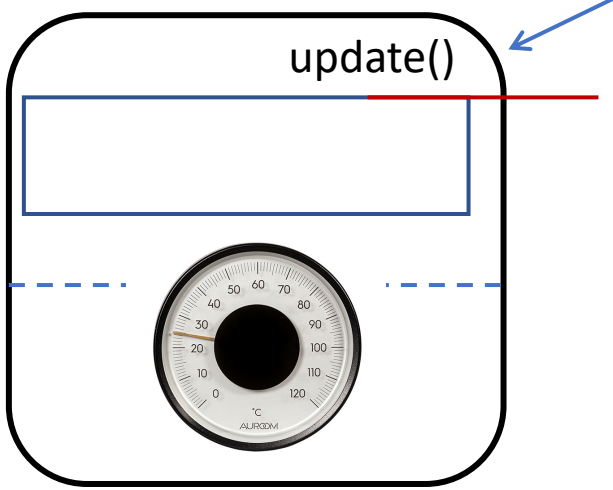


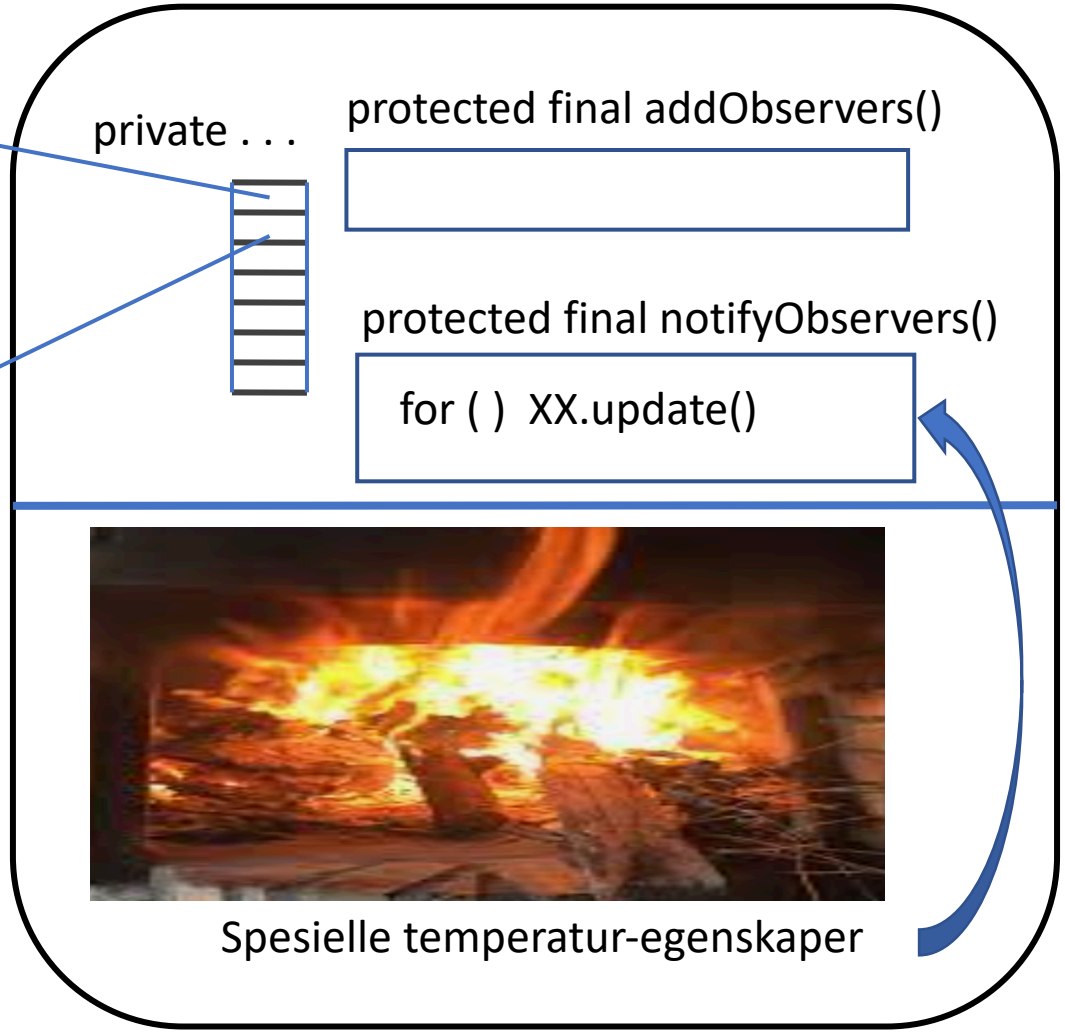
class Temperatur

Observer pattern: Objekter



Forskjellige **temperaturmålere** som viser fram temperaturen:
Objekter som implementerer Observer





Temperatur: Objekt som er subklasse av Observable

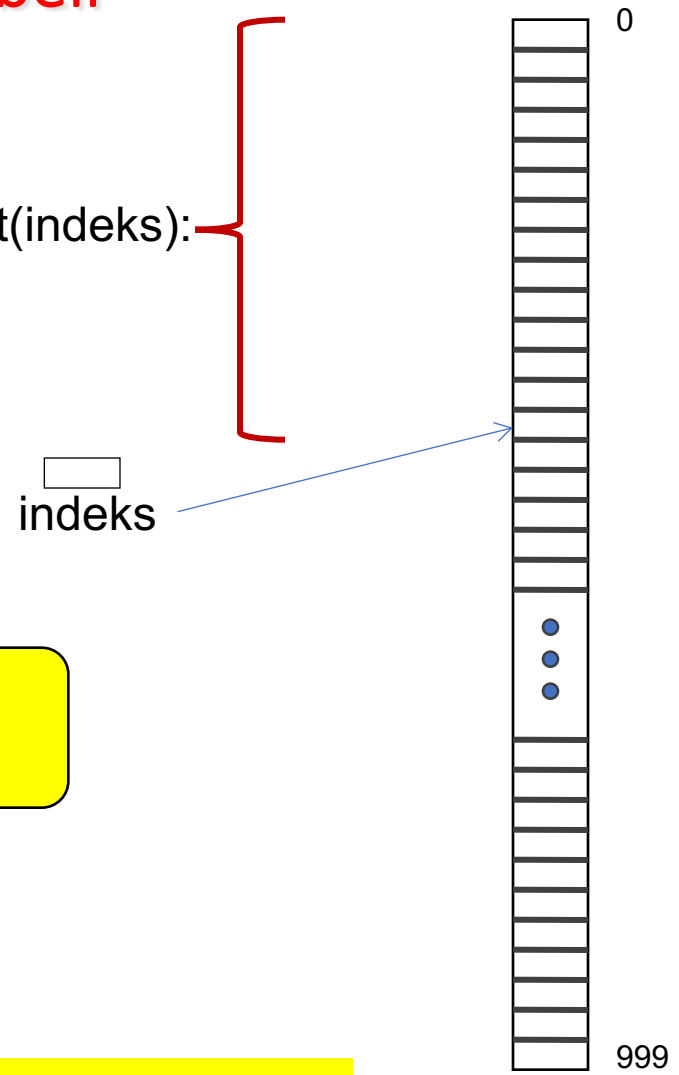
Løkkeinvarianter

For-betingelse her

tom - true

Finne minste verdi i tabell

minstTilNaa



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

= Invariant(indeks):

Induksjons-basis: indeks = 0

Induksjons-skritt: indeks = indeks + 1

Resultat: indeks = 999:

Bak-betingelse:

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

Invarianter i objekter

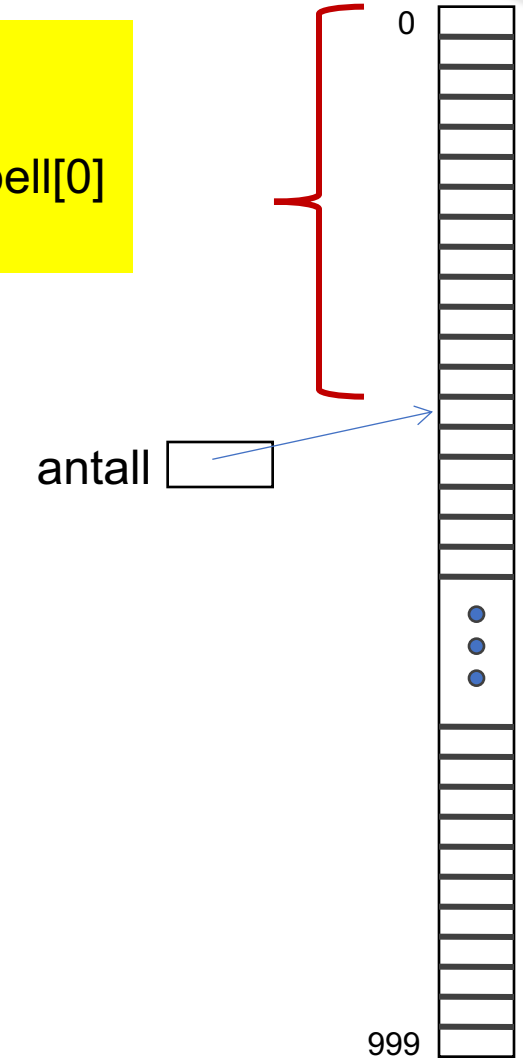
(class invariantes)

Uten tråder

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

```
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)

Eksamenstips

1. Les og forstå oppgaven
2. Les og forstå oppgaven
3. Les og forstå oppgaven

4. Ofte er oppgaven lettere enn du tror
5. Løs litt av hver oppgave
   - Karakteren din er et veiet snitt av karakteren på alle deloppgavene

Konklusjon

Viktig i IN1010:

- Arv
- Datastrukturer
- Rekursjon
- Tråder
- GUI
- God programmeringsskikk
  - Testing, programmeringsmønstre, invarianter

Lykke til på EKSAMEN !