UNIVERSITETET I OSLO

Institutt for informatikk

Invarianter, tilstander og litt mer semantikk

INF1010 – 11. mai 2017

Stein Gjessing

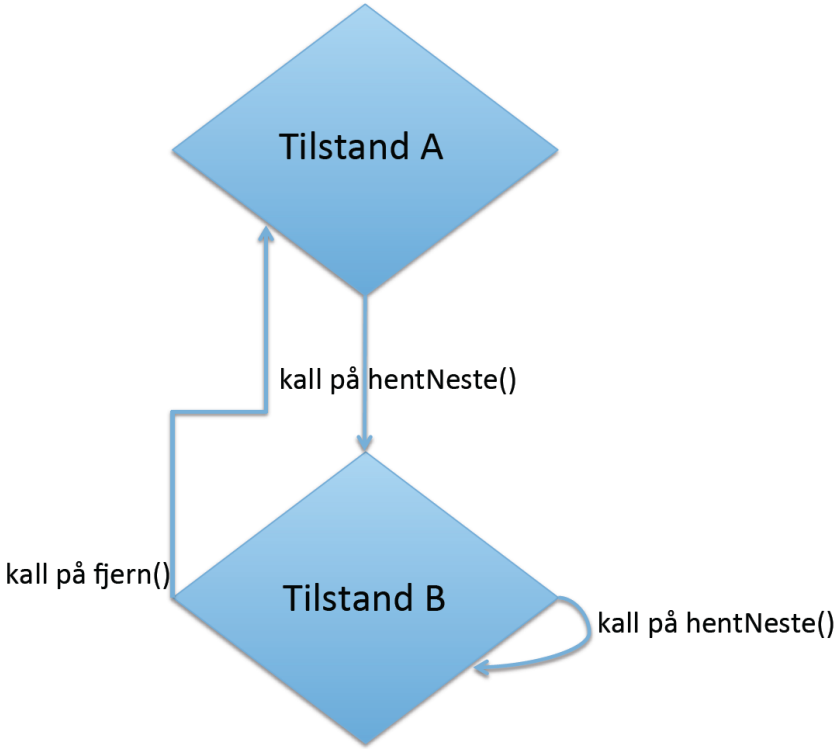
Invariant (= Invariant tilstandspåstand)

- Vi har sett flere ganger at det er svært nyttig å formulere **tilstandspåstander/tilstander** som styrer programmet (neste side).

- Når du lager en **løkke** er det alltid en invariant eller tilstandspåstand som sier hvor langt arbeidet i løkka er kommet
- Data i et **objekt** er alltid styrt av en (eller flere) invarianter (konsistensregler, tilstandspåstander)

- Tilstandspåstand = engelsk: assertion

Eksempel fra Stein Michaels forelesning om iteratorer



Tilstandsdiagram. Engelsk: state diagram

WIKIPEDIA:

Invariant (computer science)

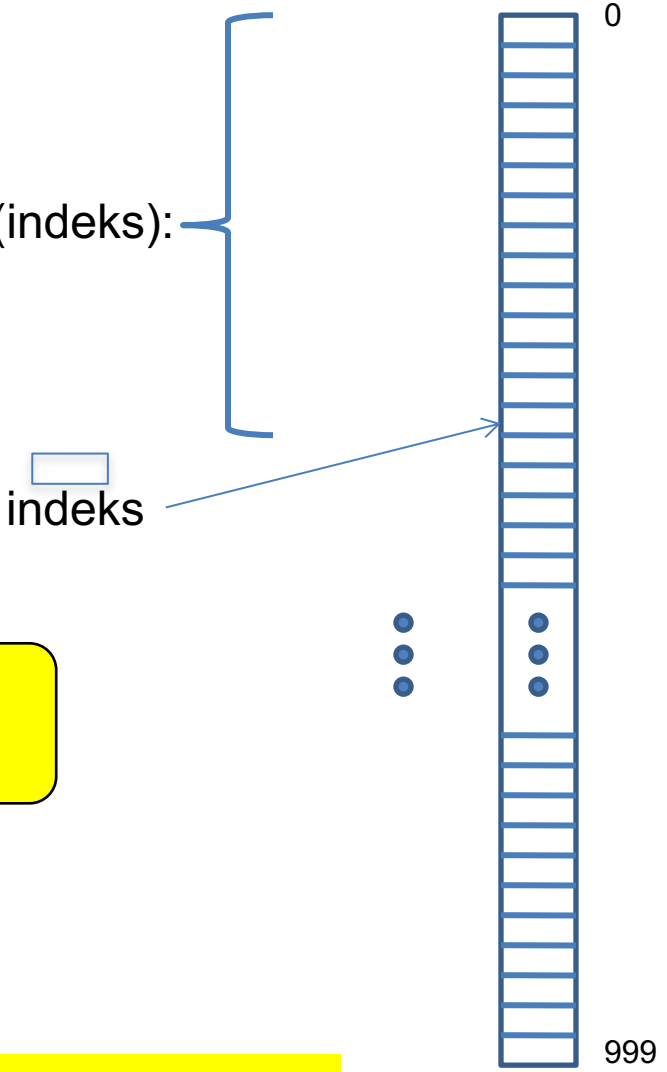
- In computer science, an **invariant** is a condition that can be relied upon to be true during execution of a program, or during some portion of it. It is a logical assertion that is held to always be true during a certain phase of execution.
  For example, a **loop invariant** is a condition that is true at the beginning and end of every execution of a loop.
  . . . . .
- Programmers often use assertions in their code to make invariants explicit. Some object oriented programming languages have a special syntax for specifying **class invariants.**

Invarianter på data 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]

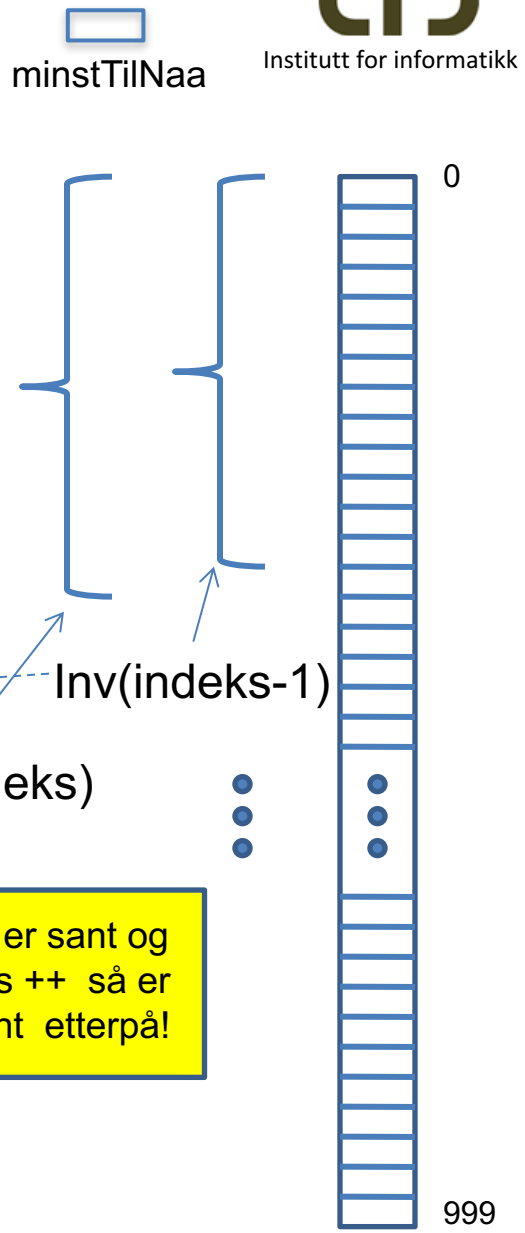
Invarianter på data i løkker

Eksempel: **Finne minste verdi i tabell**


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

int minstTilNaa = tabell[0];

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

for (indeks = 1; indeks < 1000; indeks ++) {

    // minstTilNaa inneholder minste verdi i området
    //  fra og med tabell [0] til og med tabell[indeks-1]

    if (minstTilNaa > tabell [indeks] ) minstTilNaa = tabell[indeks]

    // minst TilNaa inneholder minste verdi i området
    //  fra og med tabell [0] til og med tabell[indeks]
}

// Nå er indeks == 1000
 // minstTilNaa inneholder minste verdi i området
 //  fra og med tabell [0] til og med tabell[indeks-1]
// Da følger:
 // minstTilNaa inneholder minste verdi i området
 //  fra og med tabell[0] til og med tabell[999] !!!!!!
```

Inv(0)

Induksjons-basis



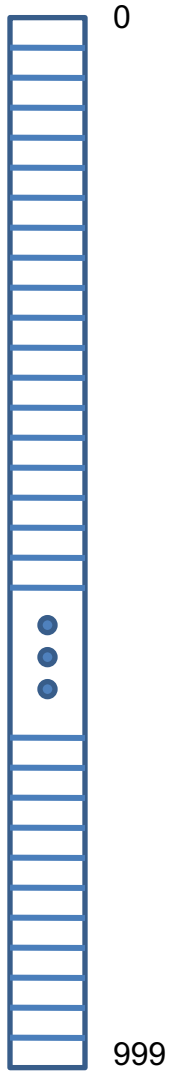
Hvis Inv(indeks) er sant og vi utfører: indeks ++ så er Inv(indeks-1) sant etterpå!

Induksjons-skritt

resultat

Invarianter på data i løkker

Eksempel: **Finne minste verdi i tabell**




```
// 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)

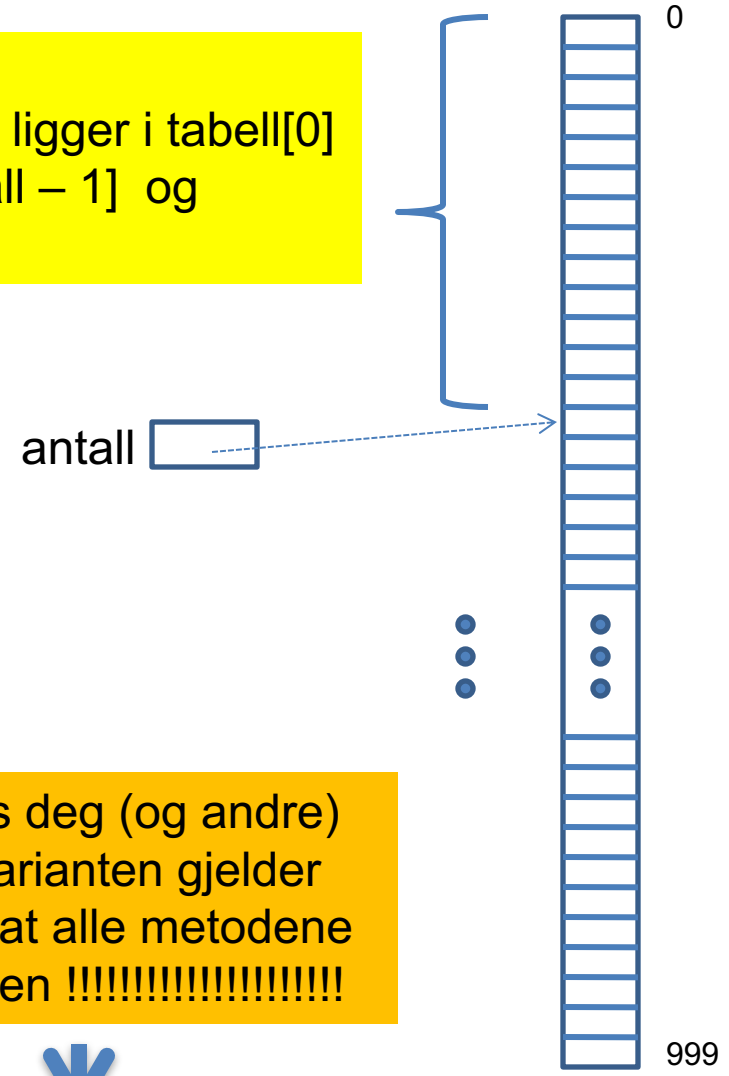
```
// minstTilNaa inneholder minste verdi i området
//  fra og med tabell[0] til og med tabell[999] !!!!!!
```

Invarianter på data i objekter

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 invarianten gjelder initielt og at alle metodene bevarer den !!!!!!!!!!!!!!!!!!!!!

Da gjelder den alltid

Spesifikasjon av objekters oppførsel (semantikk) ved hjelp av sekvens av metodekall:

Hvis objektet (på forrige side) har hatt følgende sekvens av kall:

settInn(7), settInn(4), taUt(), settInn(8), settInn(2), settInn(9), taUt()

Hva returnerer nå neste kall på taUt() ?

Hva returnerer neste kall på taUt() om dette var en FIFO-kø ?