Repetisjon

IN1010

Annika Willoch Olstad

12.03.24

Plan

1. Arv og subklasser
2. Beholdere
3. Rekursjon
4. Oppsummering
5. Midtveiseksamen

Fredagsjava 🎉

Se beskjed på semestersiden:

Vi fortsetter med **Fredagsjava** 🎉

Tid: Fredager i oddetallsuker, kl. 14.15-16.00
Sted: Seminarrom Logo

Fredagene det blir arrangert er:

- 15.mars
- 12.april
- 26.april
- 10.mai
- 24.mai

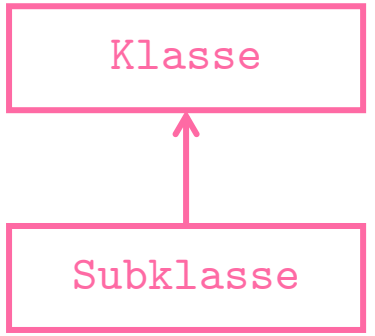
Fredagsjava er et ekstratilbud i IN1010 og en hyggelig måte å avslutte uken på, med programmering. Det er en åpen lab på tvers av grupper, der det vil være gruppelærere tilstede frem til kl 16 som kan svare på korte/ konkrete spørsmål - men det forutsettes at du har deltatt på forelesning og gruppetimer. Vi oppfordrer vi til å arbeide med Trix-oppgaver, ukesoppgaver, eller andre Java-oppgaver.

Spørsmål kan sendes til Jonas , Victor og Annika :)

Arv og Subklasser

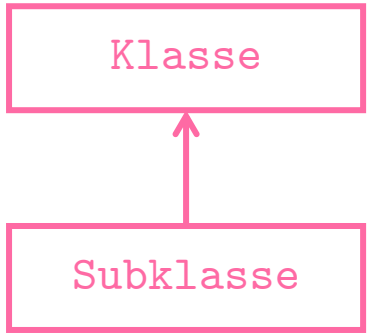
Arv og Subklasser 1

- **Klasse**: beskriver objekter med visse egenskaper
- **Subklasse**: beskriver objekter med samme egenskaper som beskrevet av **Klasse** (*superklassen)*
  - Men har i tillegg noe mer/spesielle egenskaper



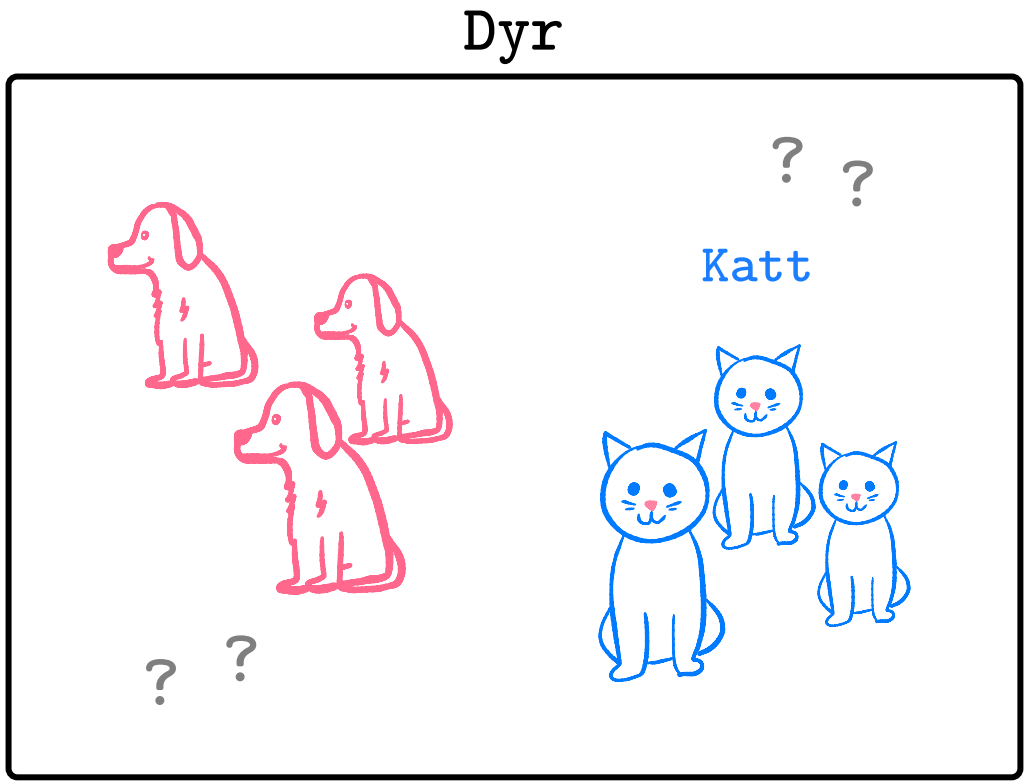
Arv og Subklasser 1

- **Klasse**: beskriver objekter med visse egenskaper
- **Subklasse**: beskriver objekter med samme egenskaper som beskrevet av **Klasse** (*superklassen)*
  - Men har i tillegg noe mer/spesielle egenskaper
- Nøkkelord: `extends`
  - Arve fra en klasse
- Nøkkelord: `super`
  - Aksessere variabler/metoder i superklassen

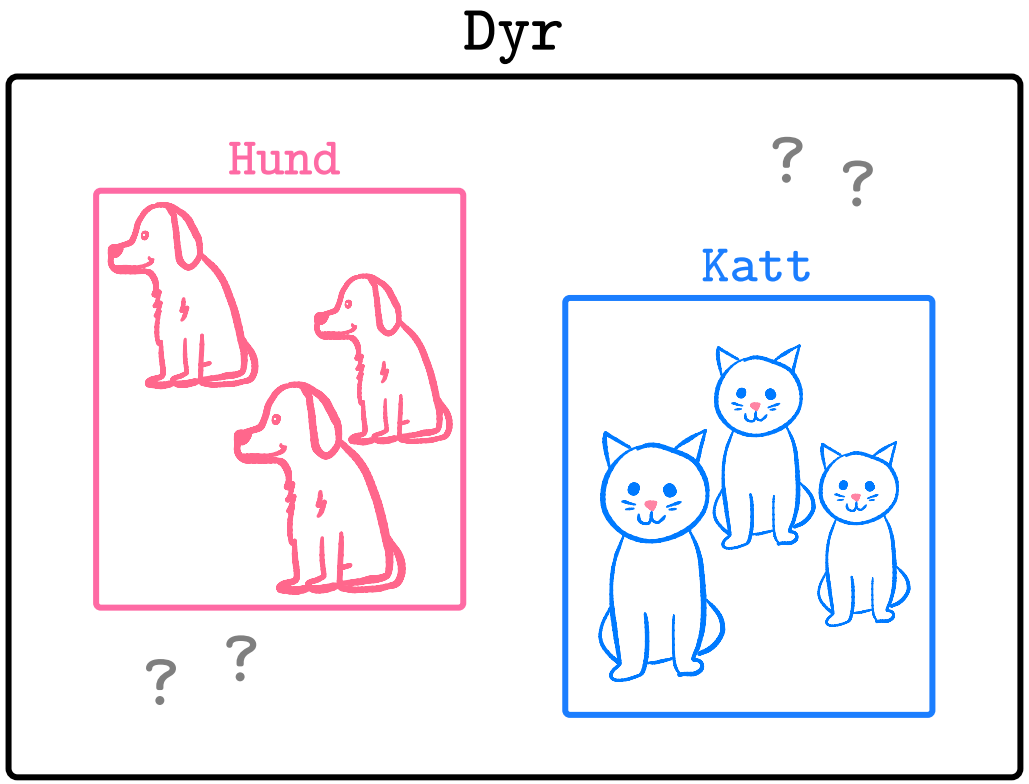



```
class Klasse {...}
class Subklasse extends Klasse {...}
```

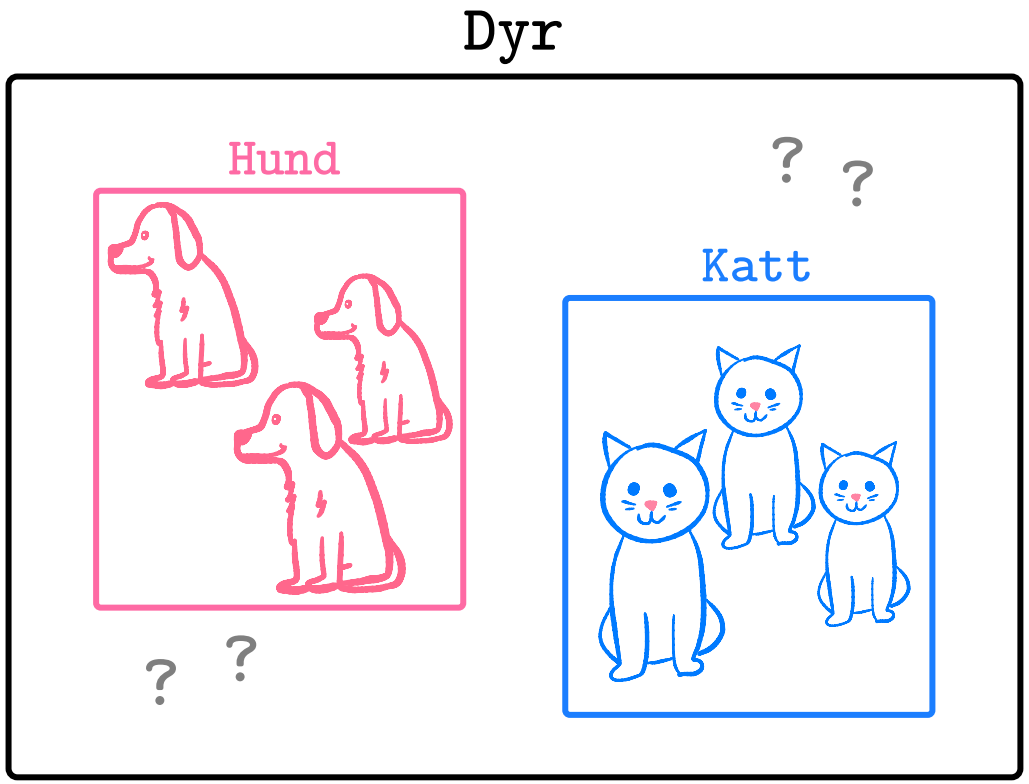
Arv og Subklasser 1 - Eksempel

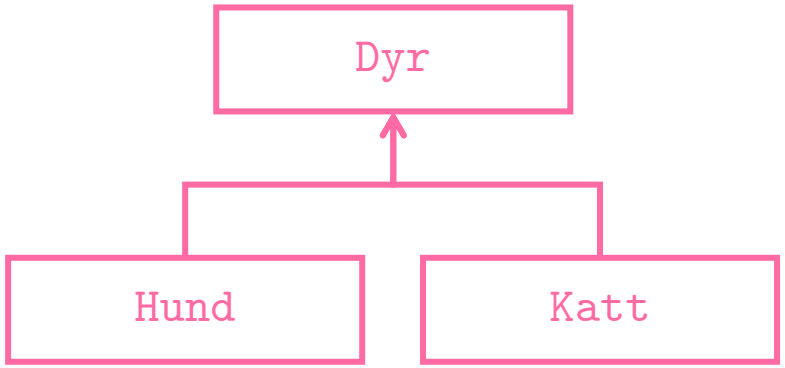


Arv og Subklasser 1 - Eksempel

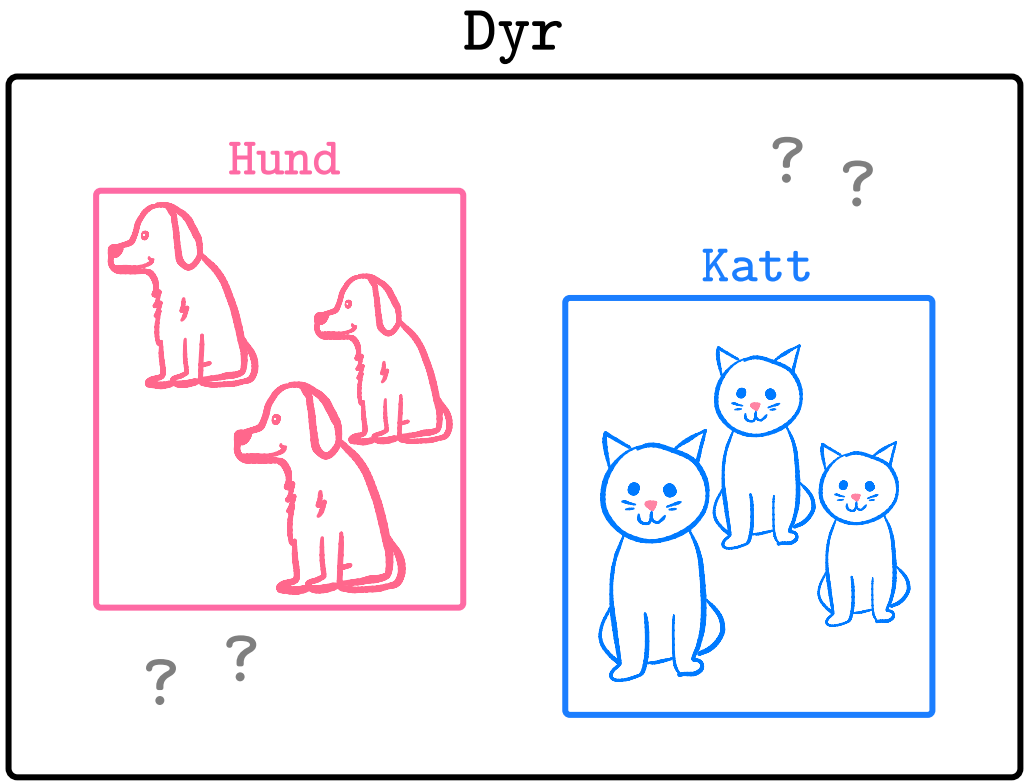


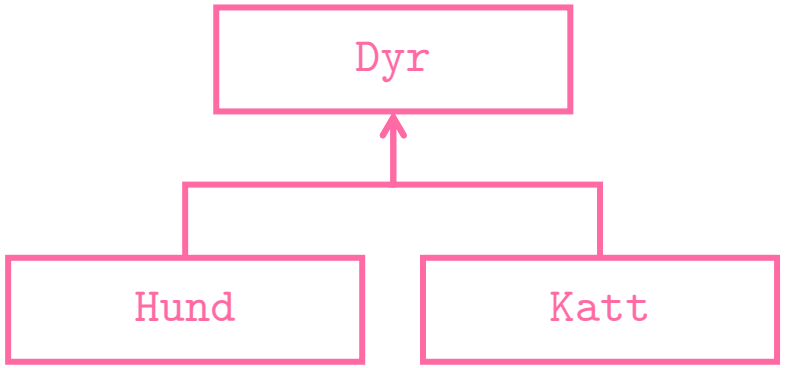
Arv og Subklasser 1 - Eksempel





Arv og Subklasser 1 - Eksempel






```
class Dyr {...}
class Hund extends Dyr {...}
class Katt extends Dyr {...}
```

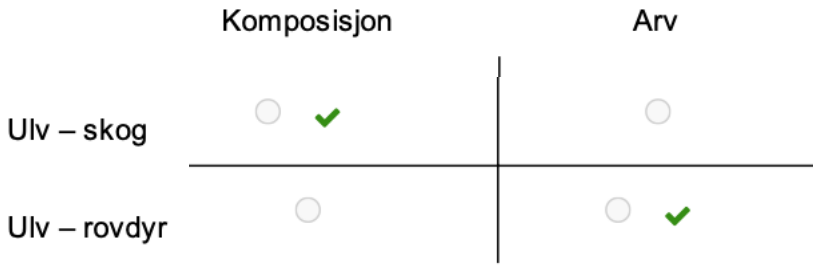
Arv og Subklasser 2

- **Arv vs. Komposisjon**
  - **Arv**: en klasse utvider en annen
  - **Komposisjon:** refererer til objekter av andre klasser

Arv og Subklasser 2

- **Arv vs. Komposisjon**
  - **Arv**: en klasse utvider en annen
  - **Komposisjon:** refererer til objekter av andre klasser

- Eksempel fra prøveeksamen
  - Ulv *er en type* rovdyr
    - Men **ikke** *en type* skog
  - Ulv har felles egenskaper med andre rovdyr
    - Men ikke samme egenskaper som en skog



Eksempel fra prøveeksamen

Arv og Subklasser 2

- **Polymorfi** = «flere former»
  - Overskriver (`@Override`)
    - Samme metode har flere former

Arv og Subklasser 2

- **Polymorfi** = «flere former»
  - Overskriver (`@Override`)
    - Samme metode har flere former

```
class Dyr {
    public void lagLyd(){System.out.println("Pip!");}
}

class Hund extends Dyr {
    public void lagLyd(){System.out.println("Voff!");}
}

class Katt extends Dyr {
    public void lagLyd(){System.out.println("Mjau!");}
}
```

Arv og Subklasser 3

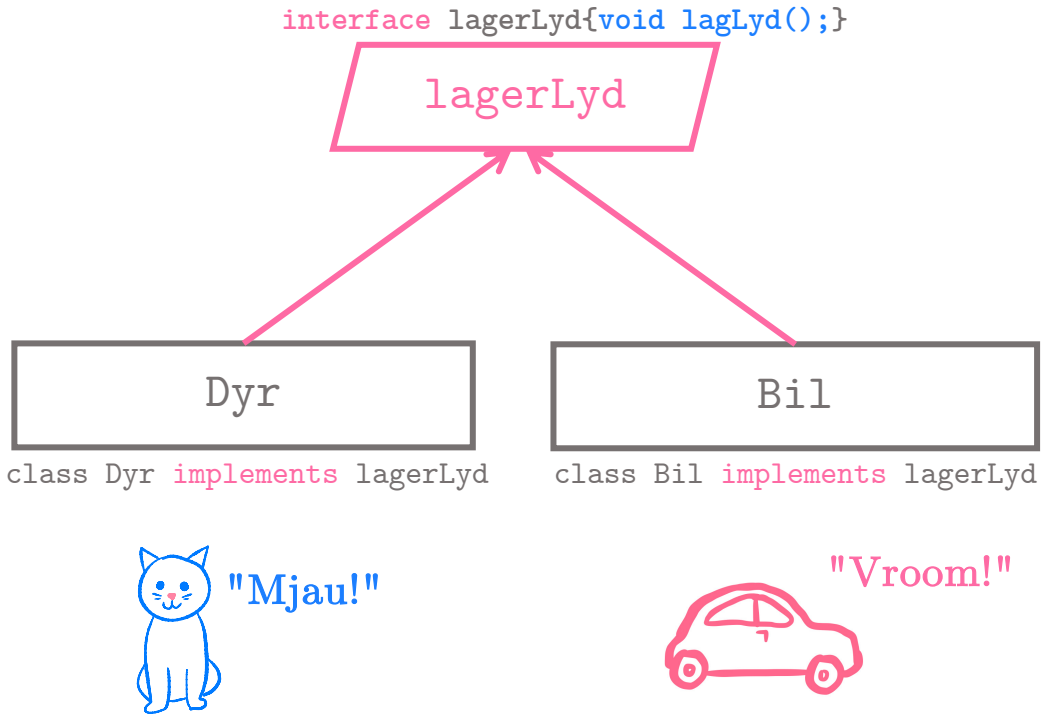
- **Interface/grensesnitt**
  - Samling felles egenskaper
    - Ulike ting av ulike typer kan dele egenskaper
      - F.eks. `lagerLyd`

- Nøkkelord:
  - `interface`
  - `implements`

Arv og Subklasser 3

- **Interface/grensesnitt**
  - Samling felles egenskaper
    - Ulike ting av ulike typer kan dele egenskaper
      - F.eks. `lagerLyd`

- Nøkkelord:
  - `interface`
  - `implements`



Utfordring: Casting

- *Referansens type* avgjør hvordan objektet fremtrer
  - Ser på objektet med ulike «briller»
  - Hvilke egenskaper har vi tilgang til?
- **Casting/typekonvertering**
  - Lar oss «bytte briller»

Utfordring: Casting - eksempel

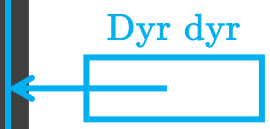
```java
public static void main(String[] args) {
    Dyr dyr = new Hund();
    dyr.gjorTriks(); // går ikke



}
```

```java
class Dyr {
    public void lagLyd(){...}
}

class Hund extends Dyr {
    public void lagLyd(){...}
    public void gjorTriks(){...}
}
```



Utfordring: Casting - eksempel

```
public static void main(String[] args) {
    Dyr dyr = new Hund();
    dyr.gjorTriks(); // går ikke

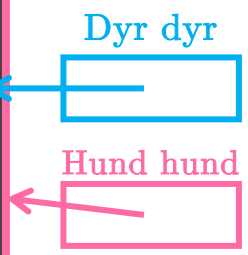
    Hund hund = (Hund) dyr;
    hund.gjorTriks();



}
```

```
class Dyr {
    public void lagLyd(){...}
}

class Hund extends Dyr {
    public void lagLyd(){...}
    public void gjorTriks(){...}
}
```



Utfordring: Casting - eksempel

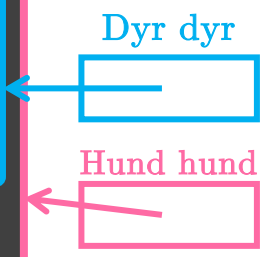
```
public static void main(String[] args) {
    Dyr dyr = new Hund();
    dyr.gjorTriks(); // går ikke

    Hund hund = (Hund) dyr;
    hund.gjorTriks();

    System.out.println(dyr.equals(hund));
    // --> true
}
```

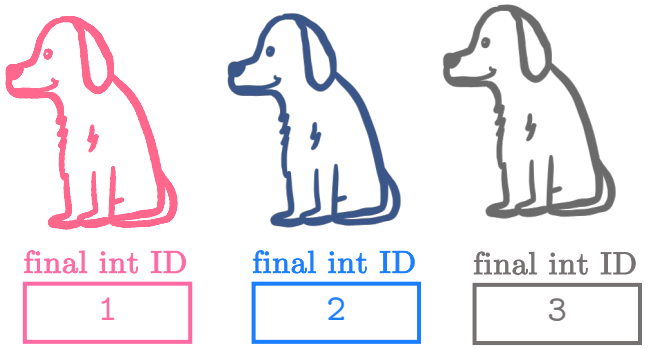
```
class Dyr {
    public void lagLyd(){...}
}

class Hund extends Dyr {
    public void lagLyd(){...}
    public void gjorTriks(){...}
}
```



Utfordring: `final` og `static`

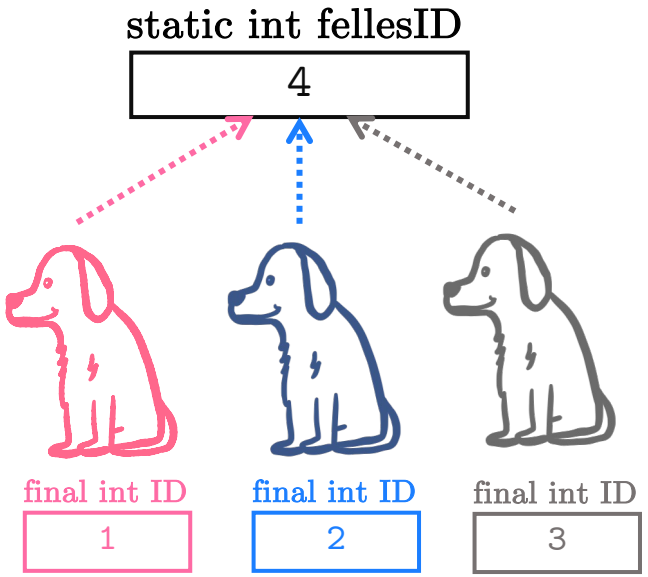
- `final`: nøkkelord for **konstanter**
  - Kan ikke endres etter initialisering



Utfordring: `final` og `static`

- `final`: nøkkelord for **konstanter**
  - Kan ikke endres etter initialisering

- `static`: nøkkelord for **felles** for alle objekter av en klasse
  - Statiske metoder og felter
  - Kan ikke bruke `this`
  - Kan kalles uten objekt

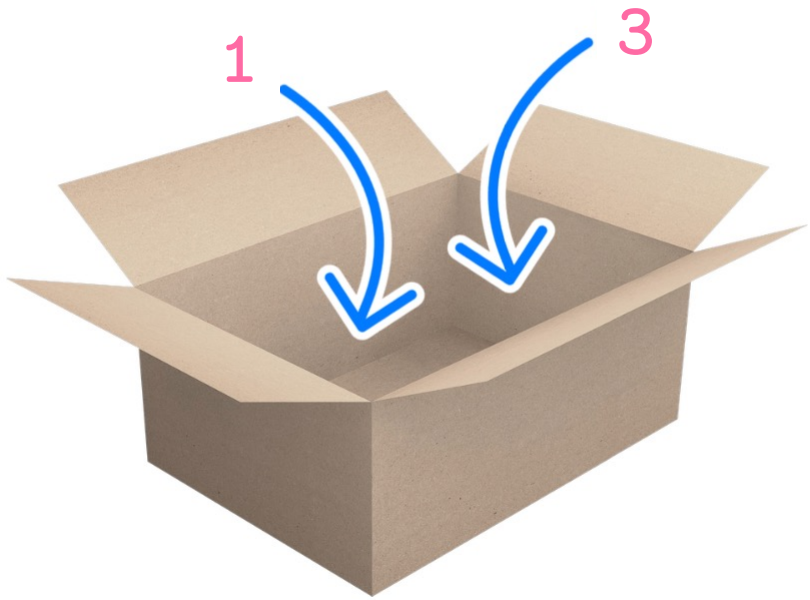


Beholdere

og generiske klasser

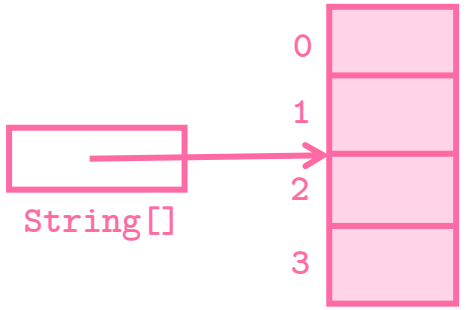
Beholdere

- Samling elementer
- I Java:
  - array
  - ArrayList
  - HashMap
  - ...



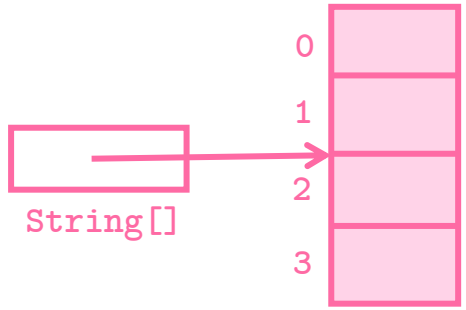
Array og ArrayList

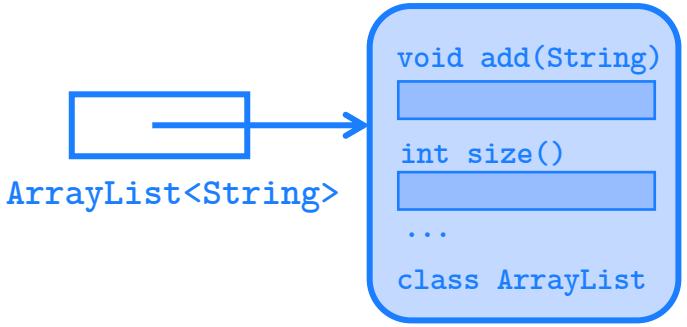
- **array**: sammenhengende gruppe i minnet som holder på et gitt antall verdier av samme type
  - Konstant størrelse
  - Tilordner og henter verdier med indekser (plassering i arrayet)



Array og ArrayList

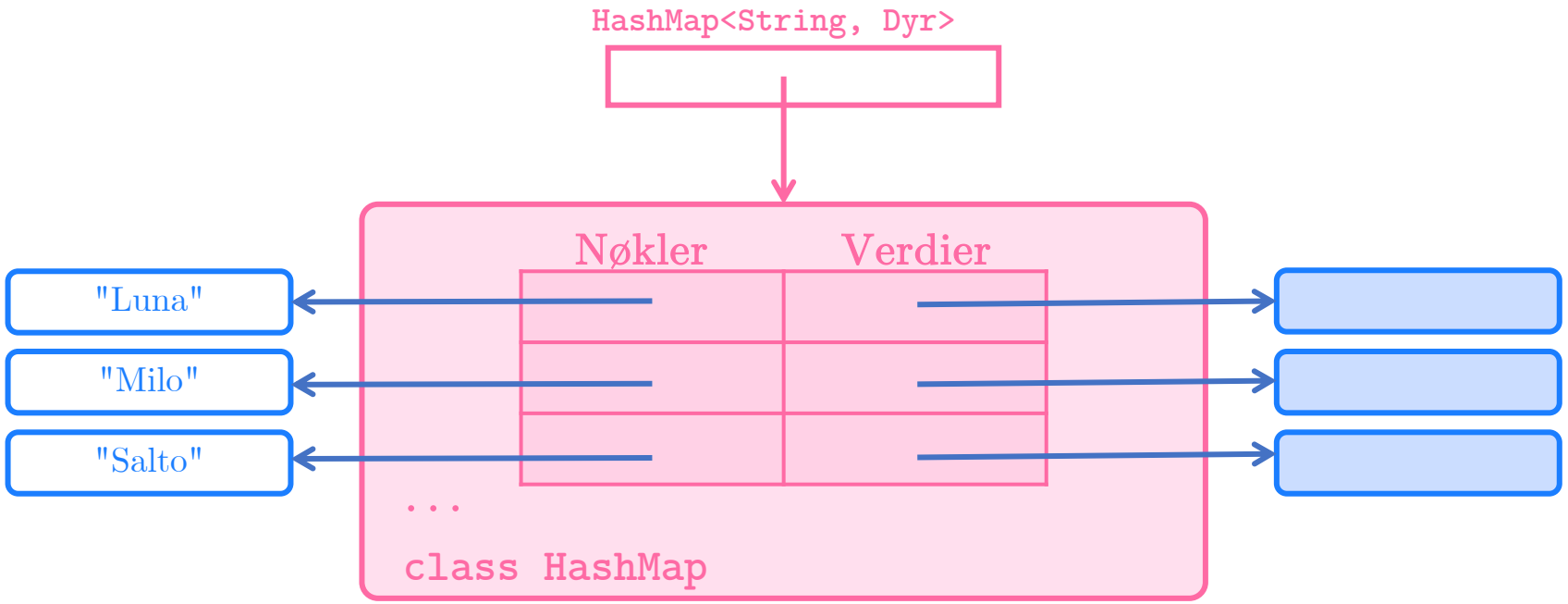
- **array**: sammenhengende gruppe i minnet som holder på et gitt antall verdier av samme type
  - Konstant størrelse
  - Tilordner og henter verdier med indekser (plassering i arrayet)
- **ArrayList**: fleksibel listeklasse i Java-biblioteket
  - Kan utvides/trekkes sammen etter behov
  - Men tar mer plass





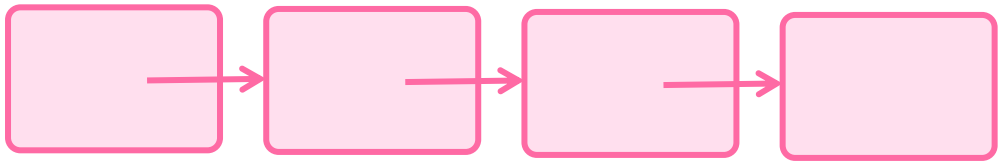
HashMap

- Samling hvor elementer identifiseres med en nøkkel



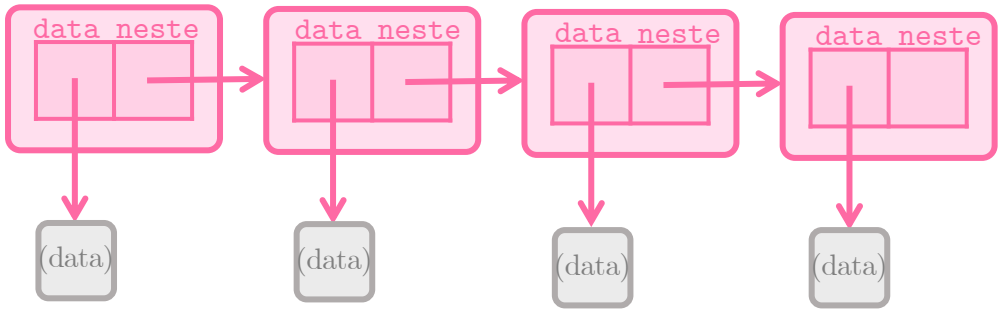
Lenkeliste

- Listestruktur med noder som er lenket sammen



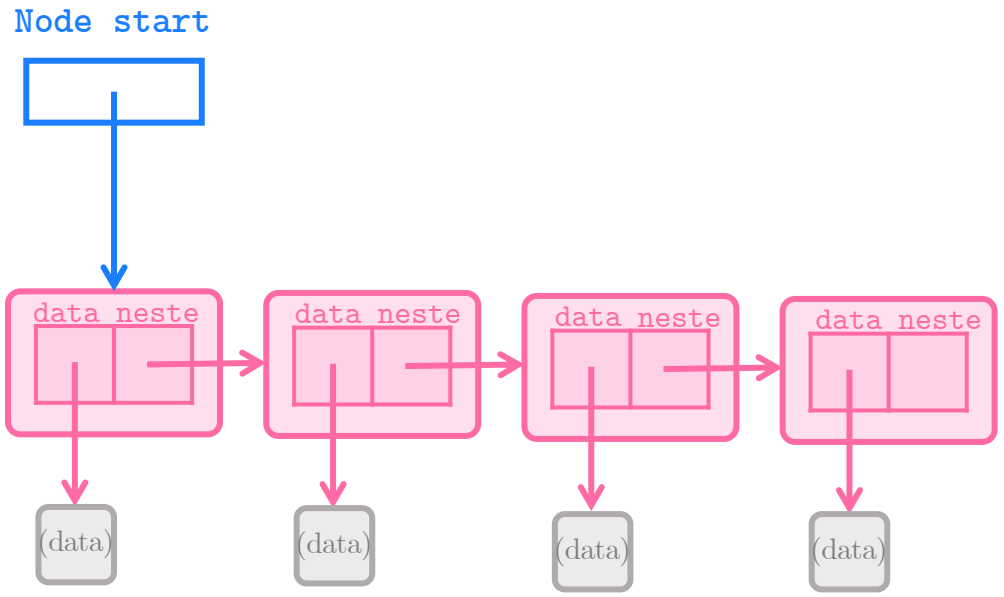
Lenkeliste

- Listestruktur med noder som er lenket sammen
  - **Node**: indre klasse
    - Holder på **data** og **referanse til neste node** i listen (*enkellenket*)



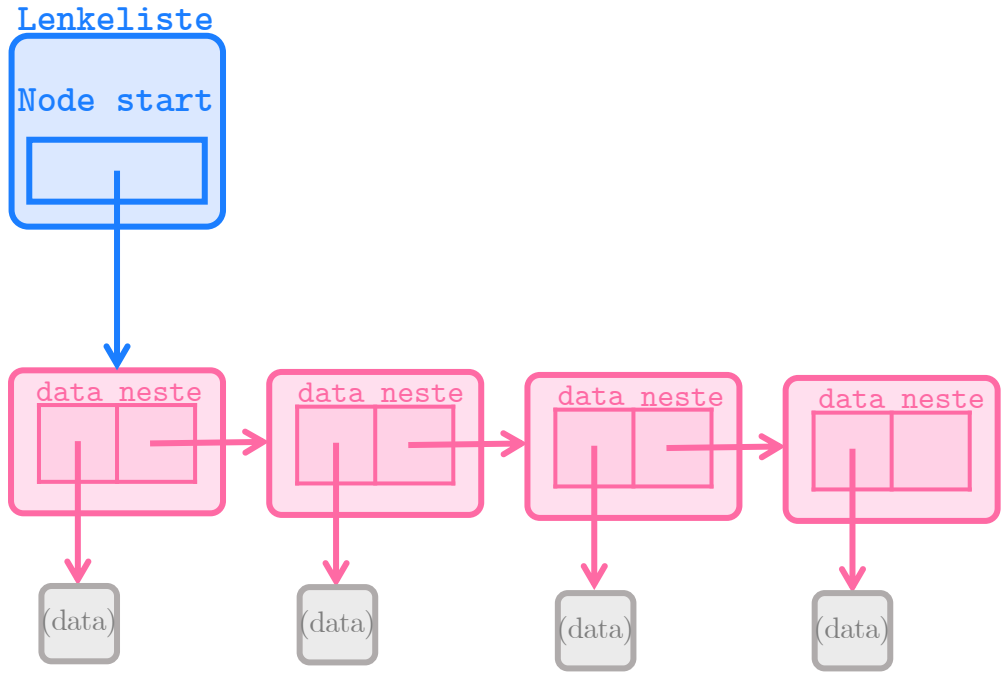
Lenkeliste

- Listestruktur med noder som er lenket sammen
  - **Node**: indre klasse
    - Holder på **data** og **referanse til neste node** i listen (*enkellenket*)
- Lenkelisten har en **start**-variabel
  - Refererer til **første** element i lenken



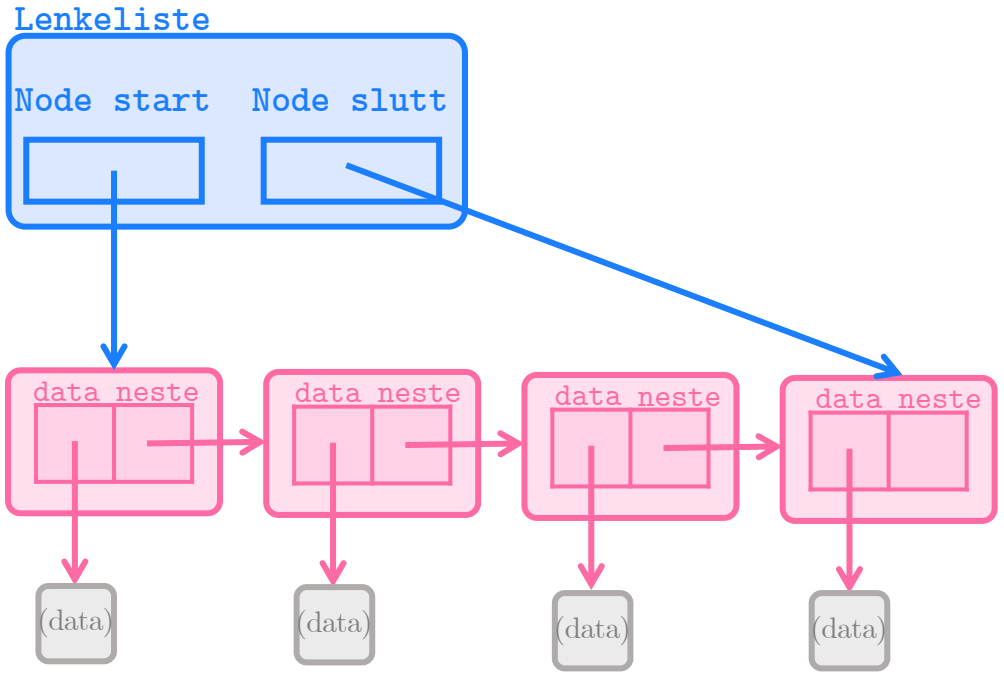
Lenkeliste

- Listestruktur med noder som er lenket sammen
  - **Node**: indre klasse
    - Holder på **data** og **referanse til neste node** i listen (*enkellenket*)
- Lenkelisten har en **start**-variabel
  - Refererer til **første** element i lenken



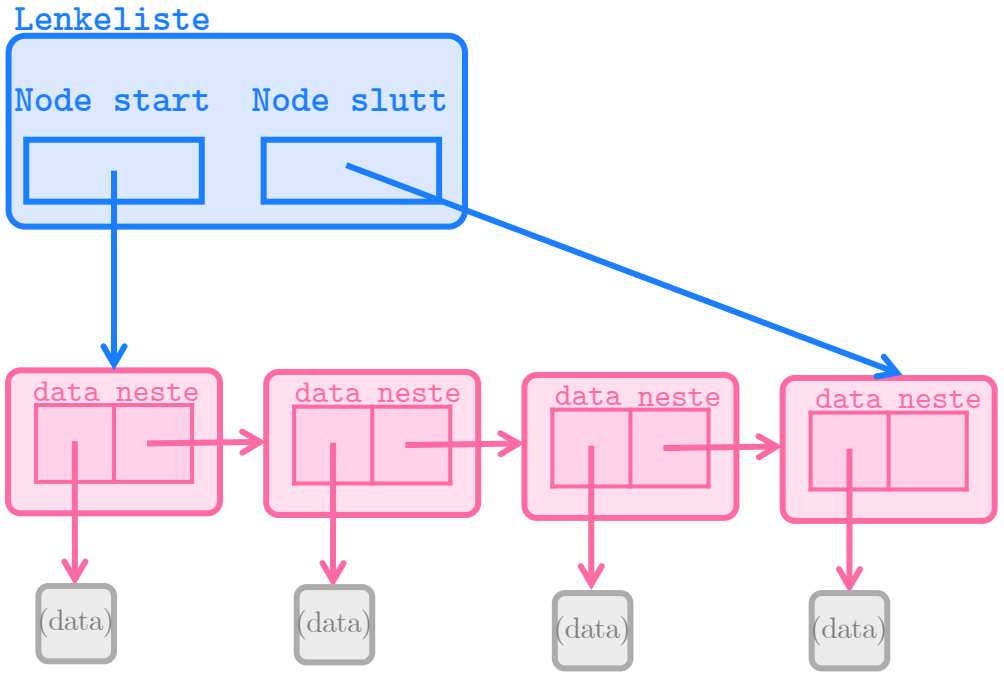
Lenkeliste

- Listestruktur med noder som er lenket sammen
  - **Node**: indre klasse
    - Holder på **data** og **referanse til neste node** i listen (*enkellenket*)
- Lenkelisten har en **start**-variabel
  - Refererer til **første** element i lenken
  - Kan også ha til **siste** element



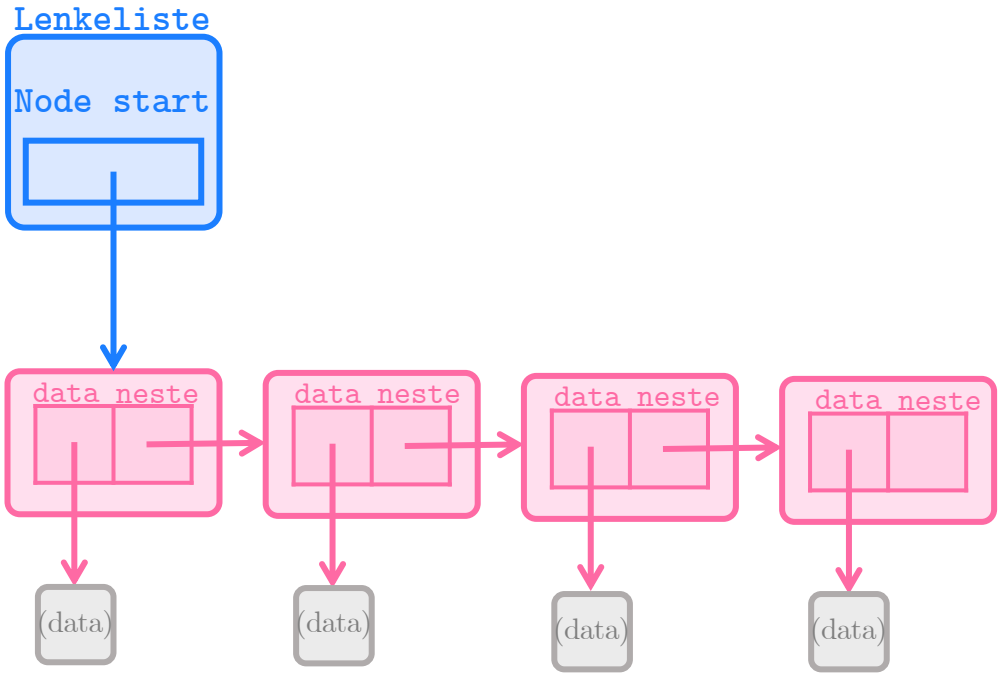
Lenkeliste

- Listestruktur med noder som er lenket sammen
  - **Node**: indre klasse
    - Holder på **data** og **referanse til neste node** i listen (*enkellenket*)
- Lenkelisten har en **start-**variabel
  - Refererer til **første** element i lenken
  - Kan også ha til **siste** element
- Kan være **enkel-** eller **dobbellenket**



Lenkeliste: Enkellenket

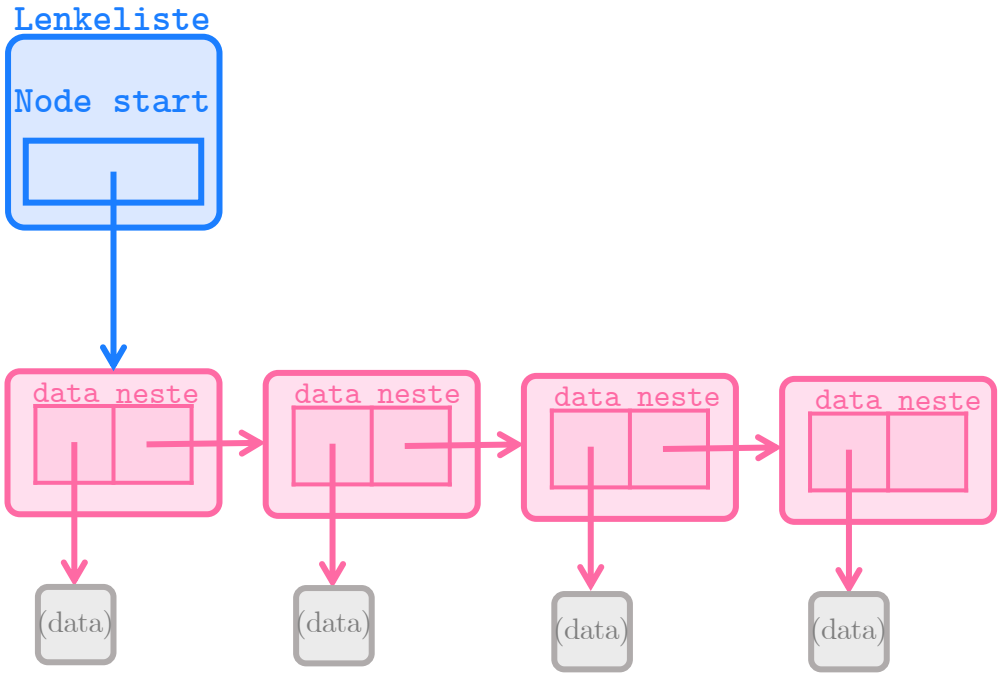
- Enkellenket liste:
  - **neste**-peker



Lenkeliste: Enkellenket

- Enkellenket liste:
  - **neste**-peker

```
class Lenkeliste {
    protected class Node {
        Node neste = null;
        //...data
    }
    protected Node start;
}
```



Lenkeliste: Dobbellenket

- Dobbellenket liste:
  - **neste**-peker



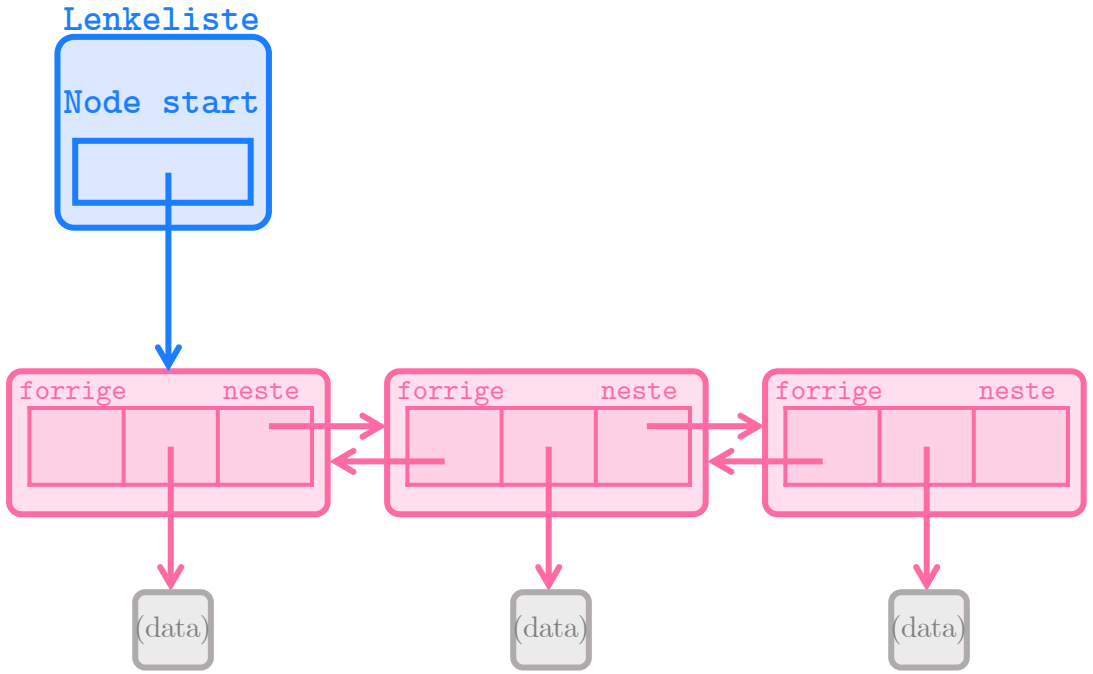
Lenkeliste: Dobbellenket

- Dobbellenket liste:
  - **neste**-peker
  - **forrige**-peker



Lenkeliste: Dobbellenket

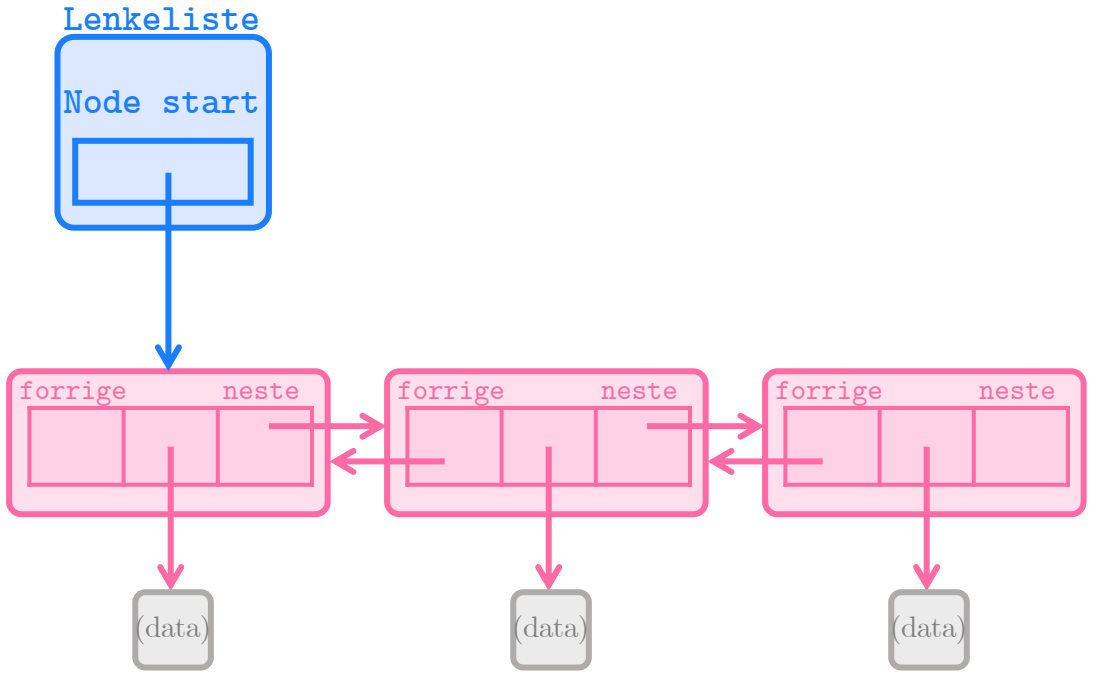
- Dobbellenket liste:
  - **neste**-peker
  - **forrige**-peker



Lenkeliste: Dobbellenket

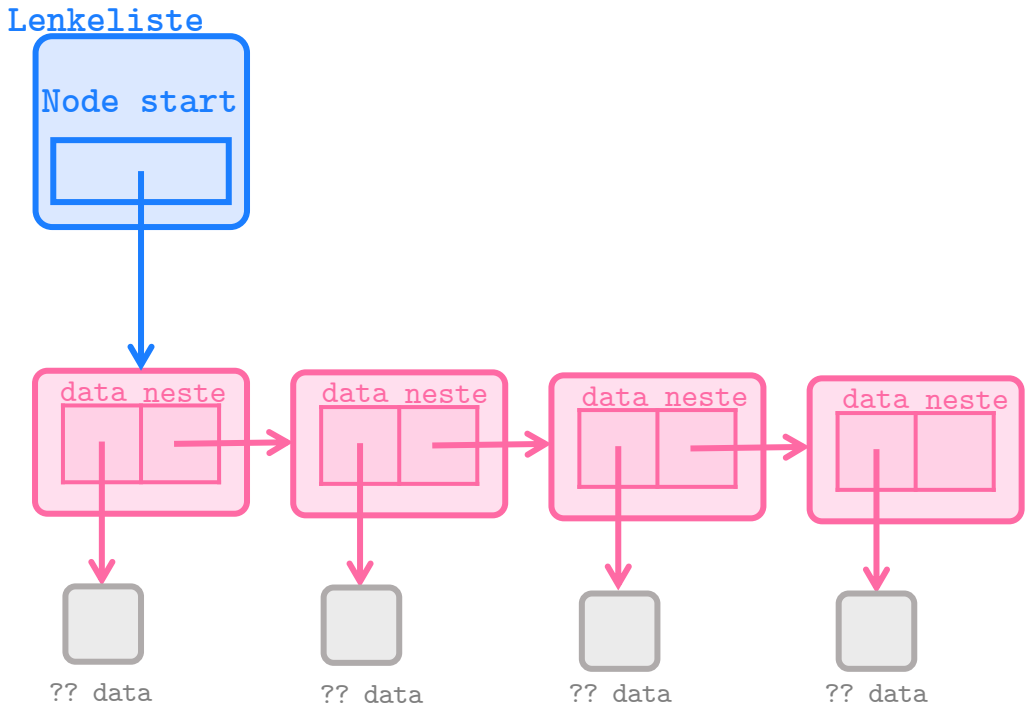
- Dobbellenket liste:
  - **neste**-peker
  - **forrige**-peker

```
class Lenkeliste {
    protected class Node {
        Node neste = null;
        Node forrige = null;
        //...data
    }
    protected Node start;
}
```



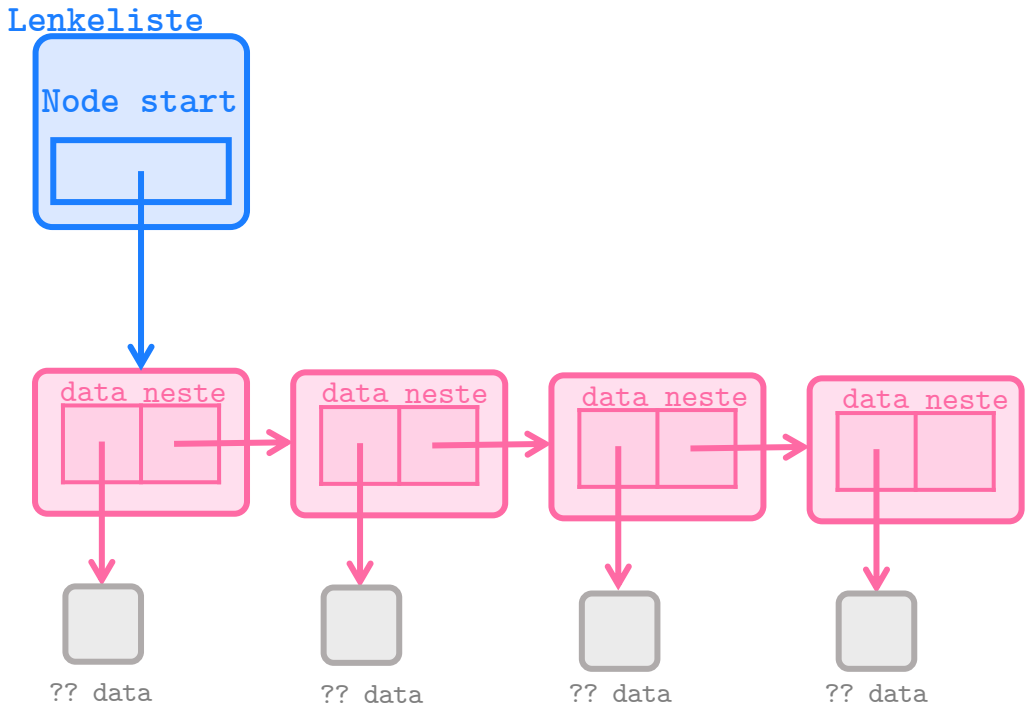
Generiske klasser

- Hvilken type har dataen vi skal lagre?
  → **Generiske klasser**



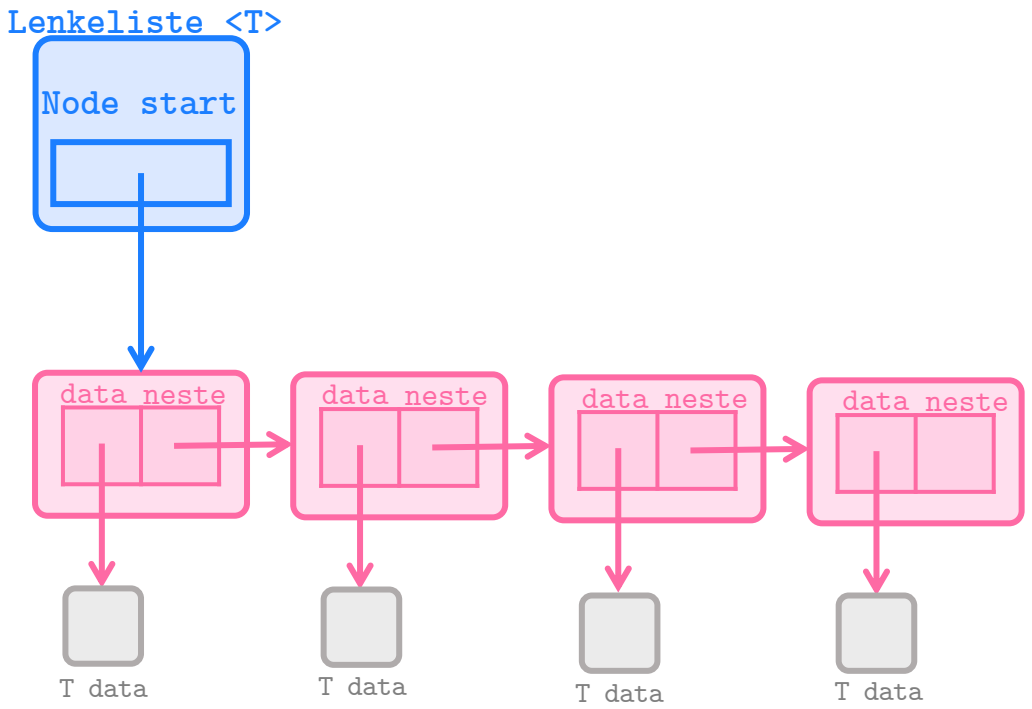
Generiske klasser

- Hvilken type har dataen vi skal lagre?
  - → **Generiske klasser**
- Klasser med parametere som angir typen til elementene inni klassen
  - **Typeparametre/klasse-parametre**
    - Markeres med f.eks. <T>
  - Nyttig for bl.a. beholdere



Generiske klasser

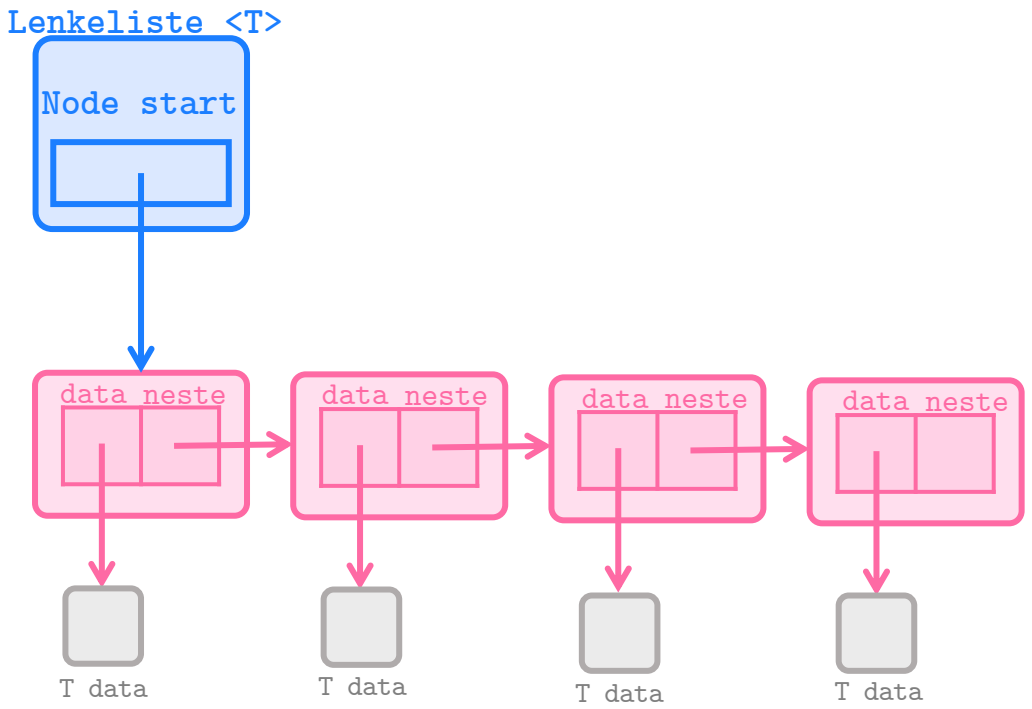
- Hvilken type har dataen vi skal lagre?
  - → **Generiske klasser**
- Klasser med parametere som angir typen til elementene inni klassen
  - **Typeparametre/klasse-parametre**
    - Markeres med f.eks. <T>
  - Nyttig for bl.a. beholdere



Generiske klasser

```
class Lenkeliste<T>{
    protected class Node {
        Node neste = null;
        T data;

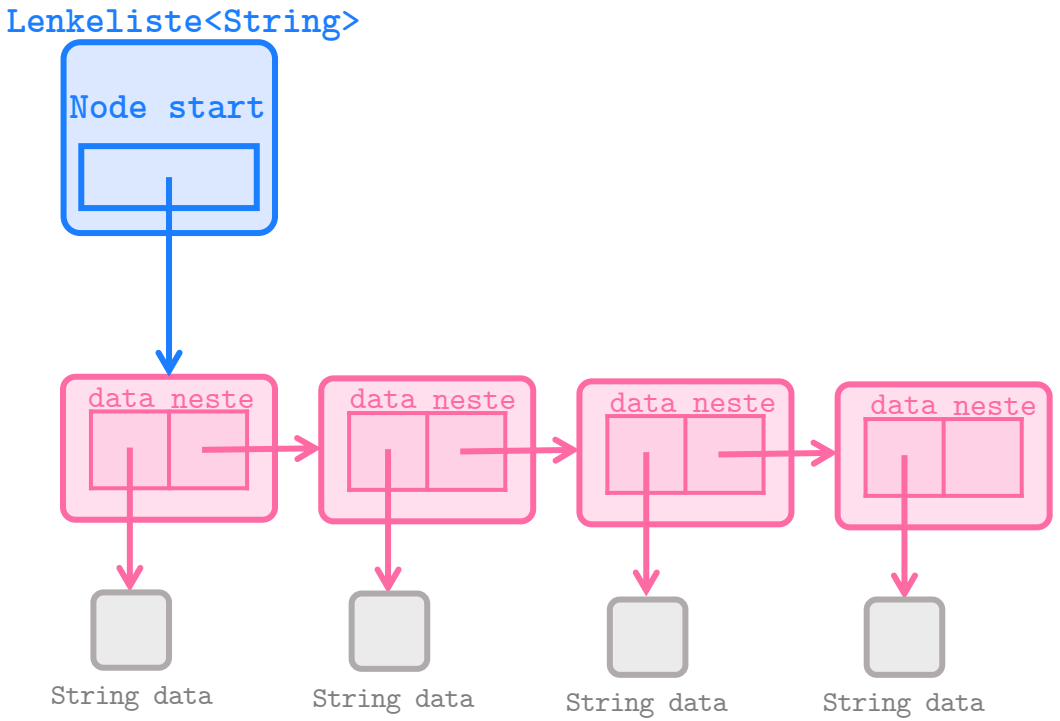
        Node (T x) {
            data = x;
        }
    }
    protected Node start = null;
}
```



Generiske klasser - eksempel

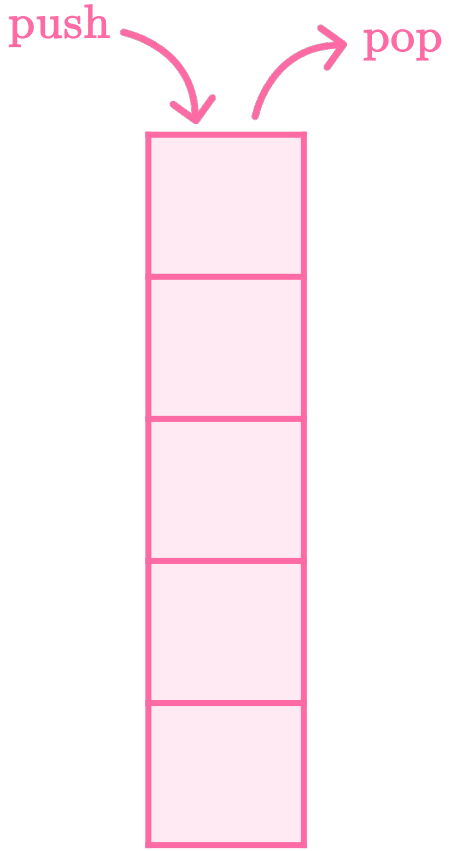
```
class Lenkeliste<T>{
    protected class Node {
        Node neste = null;
        T data;

        Node (T x) {
            data = x;
        }
    }
    protected Node start = null;
}
```



Stabel

- LIFO: Last In First Out
- Legger til og fjerner elementer fra toppen av listen
  - **Push:** legg element på toppen
  - **Pop:** ta av element fra toppen



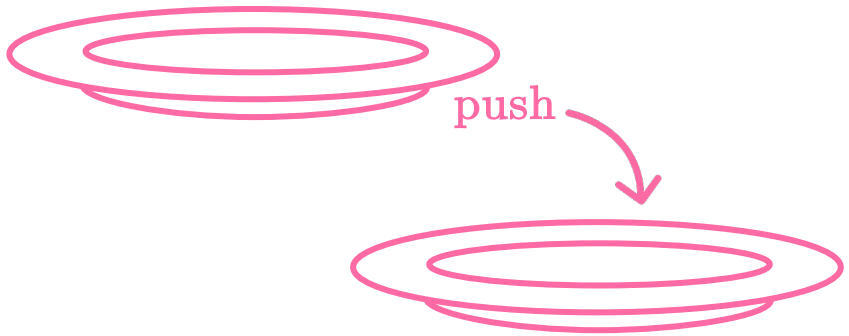
Stabel - Sammenlikning

- LIFO: Last In First Out
- Legger til og fjerner elementer fra toppen av listen
  - **Push:** legg element på toppen
  - **Pop:** ta av element fra toppen



Stabel - Sammenlikning

- LIFO: Last In First Out
- Legger til og fjerner elementer fra toppen av listen
  - **Push:** legg element på toppen
  - **Pop:** ta av element fra toppen



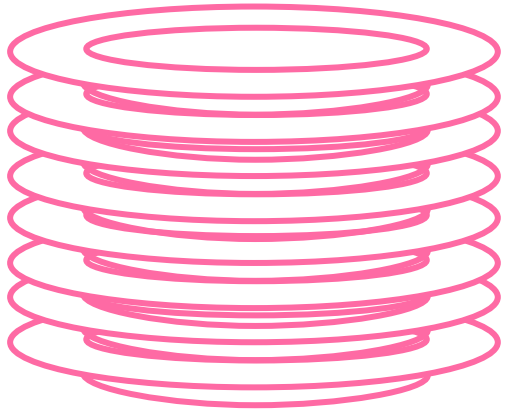
Stabel - Sammenlikning

- LIFO: Last In First Out
- Legger til og fjerner elementer fra toppen av listen
  - **Push:** legg element på toppen
  - **Pop:** ta av element fra toppen



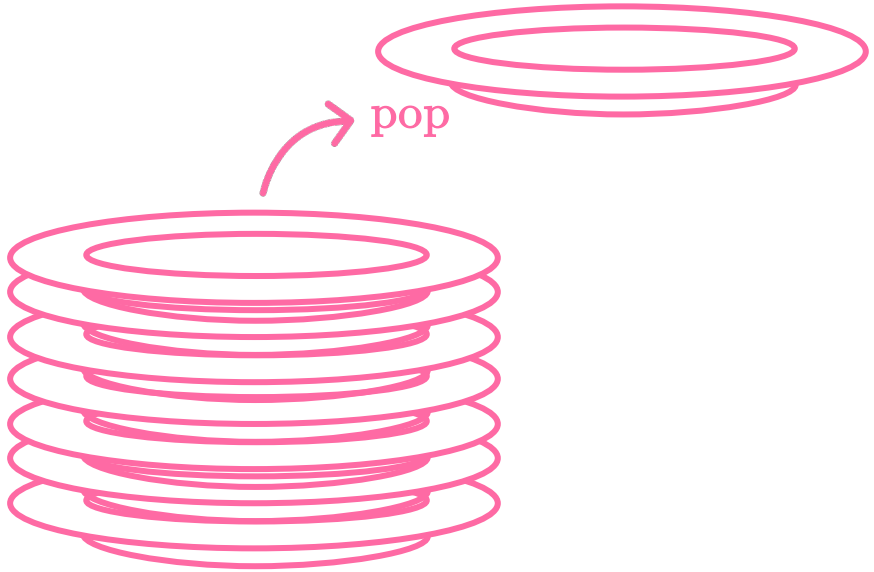
Stabel - Sammenlikning

- LIFO: Last In First Out
- Legger til og fjerner elementer fra toppen av listen
  - **Push:** legg element på toppen
  - **Pop:** ta av element fra toppen



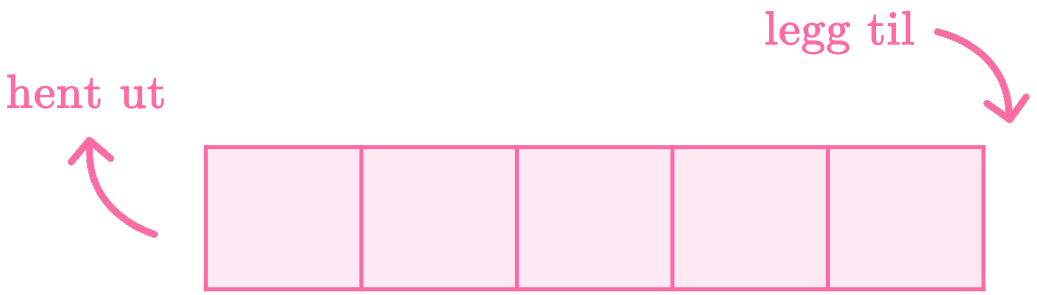
Stabel - Sammenlikning

- LIFO: Last In First Out
- Legger til og fjerner elementer fra toppen av listen
  - **Push:** legg element på toppen
  - **Pop:** ta av element fra toppen



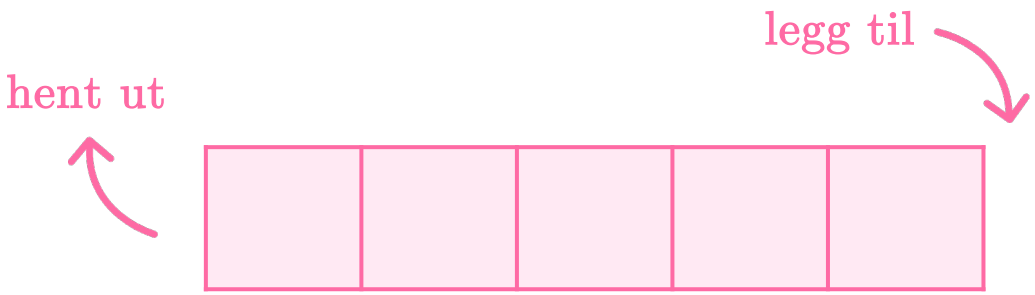
Kø

- FIFO: First In First Out
- Legger til elementer bakerst og henter ut fremst



Kø

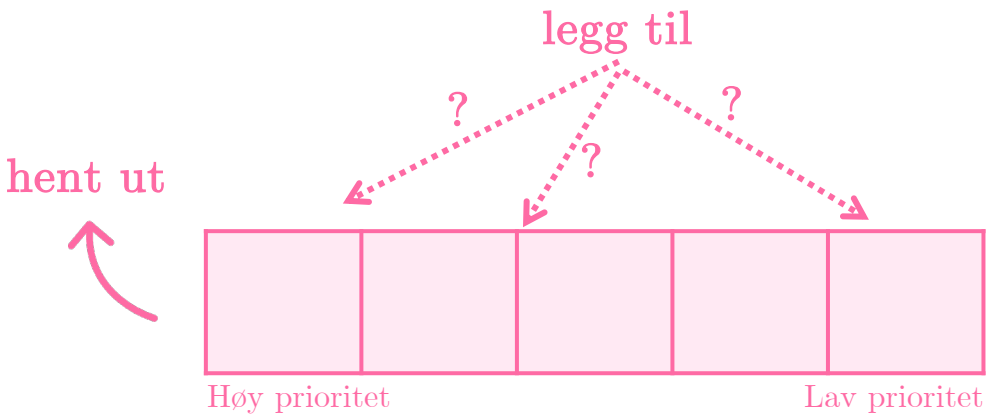
- FIFO: First In First Out
- Legger til elementer bakerst og henter ut fremst





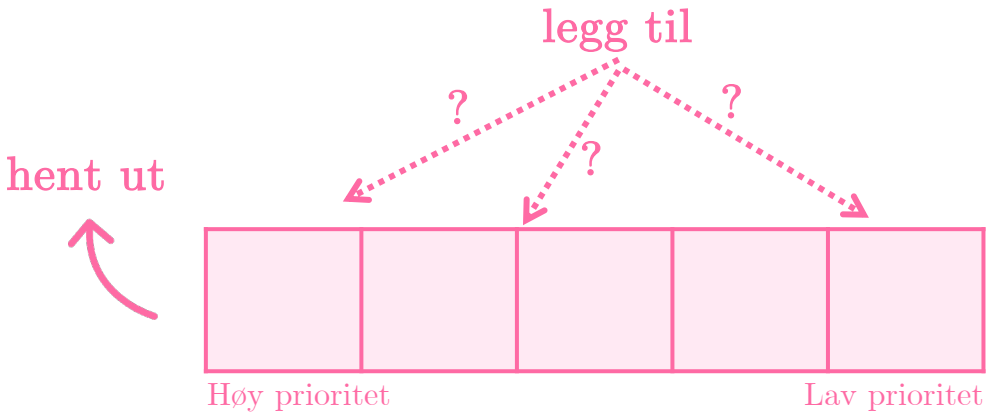
Prioritetskø

- Elementer med høyest prioritet hentes ut først



Prioritetskø

- Elementer med høyest prioritet hentes ut først





Utfordring: Comparable

- `Comparable<T>`: interface som angir noe som kan sammenliknes
  - `compareTo(T annen)`
    - **Dette** objektet sammenliknes med **annen**
    - Returnerer et heltall:
      - `0` **dette** objektet er likt **annen**
      - `< 0` **dette** objektet er mindre enn **annen**
      - `> 0` **dette** objektet er større enn **annen**

Utfordring: Comparable

- `Comparable<T>`: interface som angir noe som kan sammenliknes
  - `compareTo(T annen)`
    - **Dette** objektet sammenliknes med **annen**
    - Returnerer et heltall:
      - `0` **dette** objektet er likt **annen**
      - `< 0` **dette** objektet er mindre enn **annen**
      - `> 0` **dette** objektet er større enn **annen**

```java
public int compareTo (Hund annen) {
    if (this.alder < annen.alder) {
        return -1;
    } else if (this.alder > annen.alder) {
        return 1;
    } else {
        return 0;
    }
}
// Alternativ
public int compareTo (Hund annen) {
    return this.alder - annen.alder;
}
```

Trix 07.01

Rekursjon

Rekursjon

- En **rekursiv** metode kaller seg selv

Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

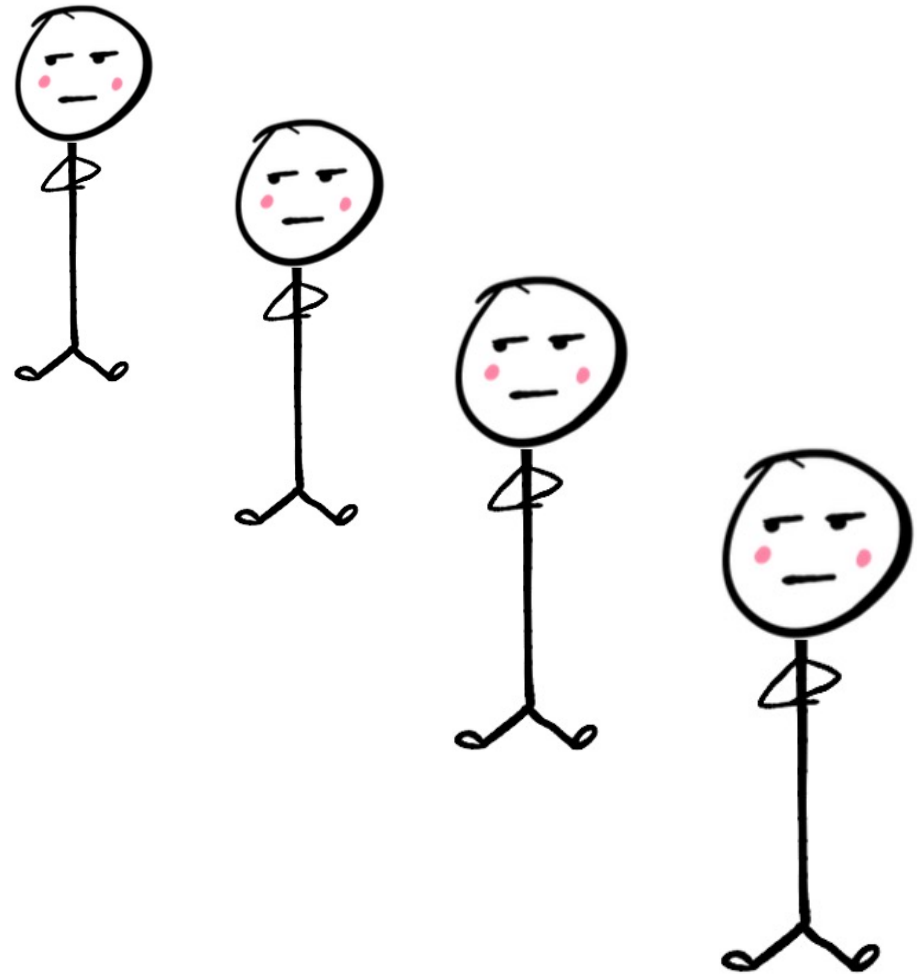
  ...helt til vi når **basistilfellet**

Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

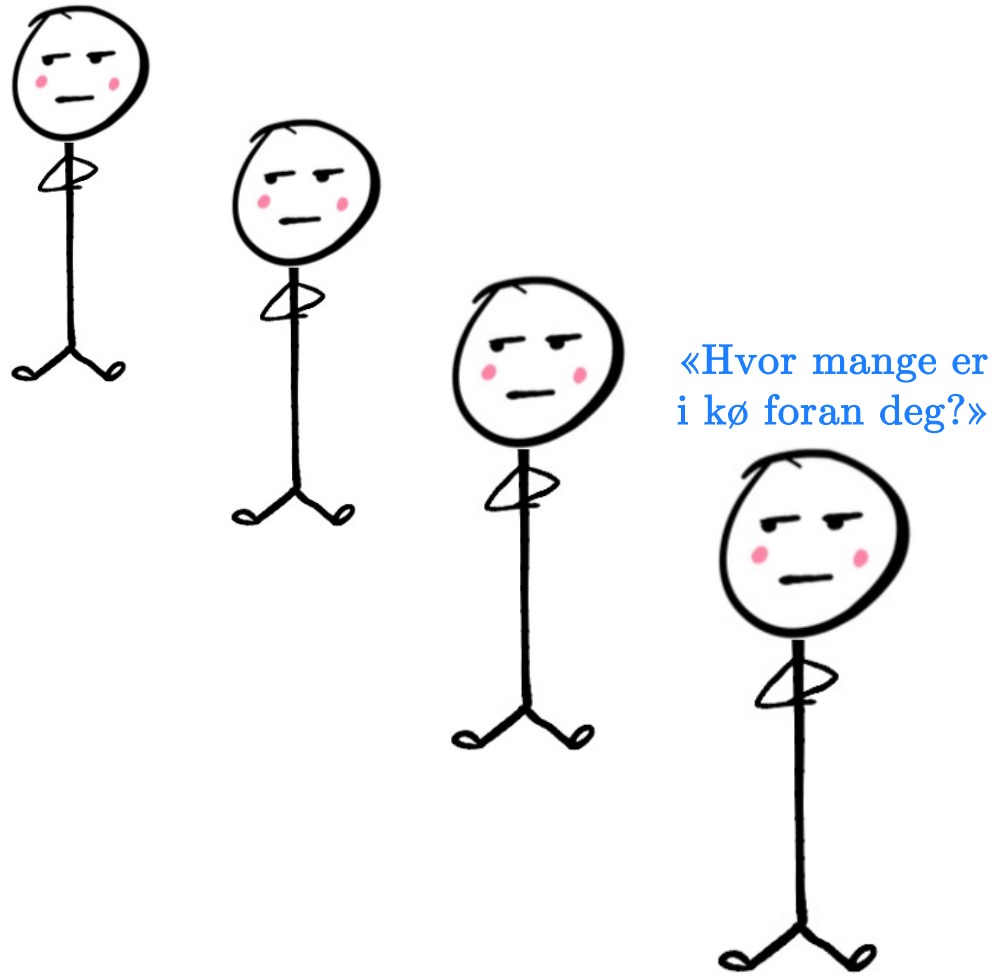


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

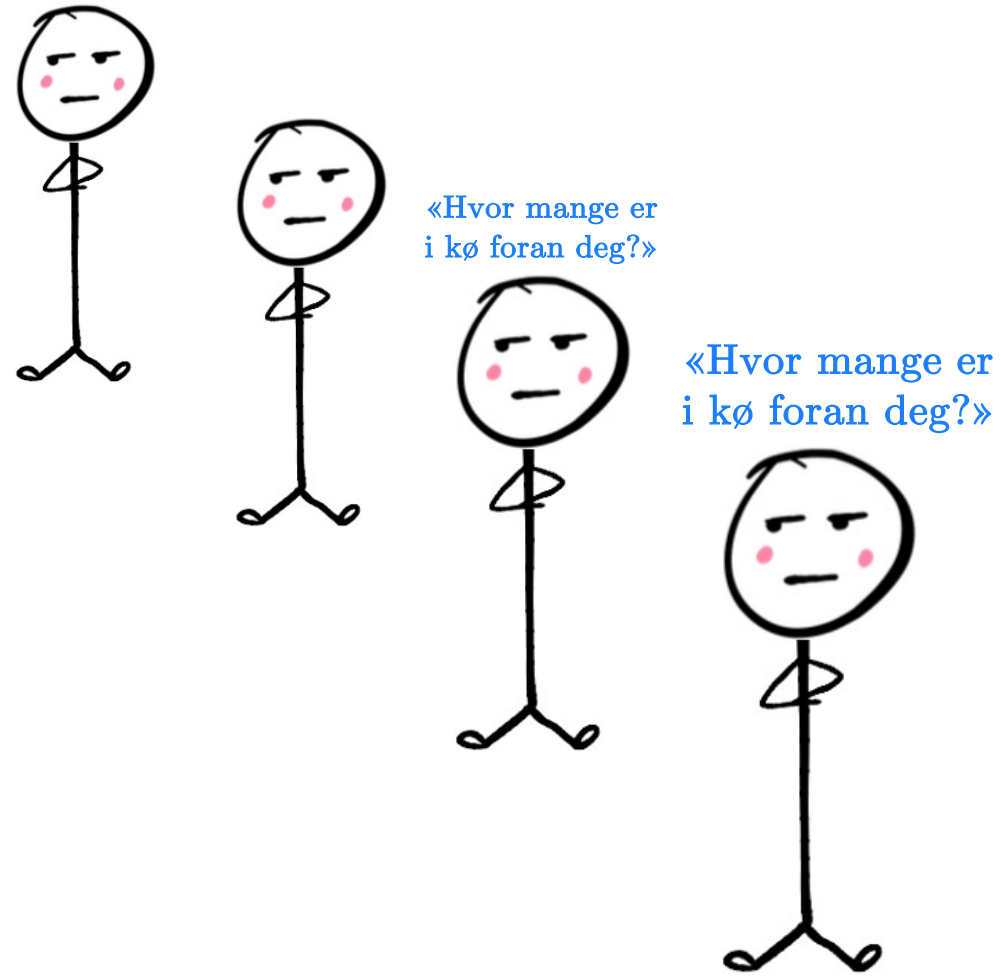


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

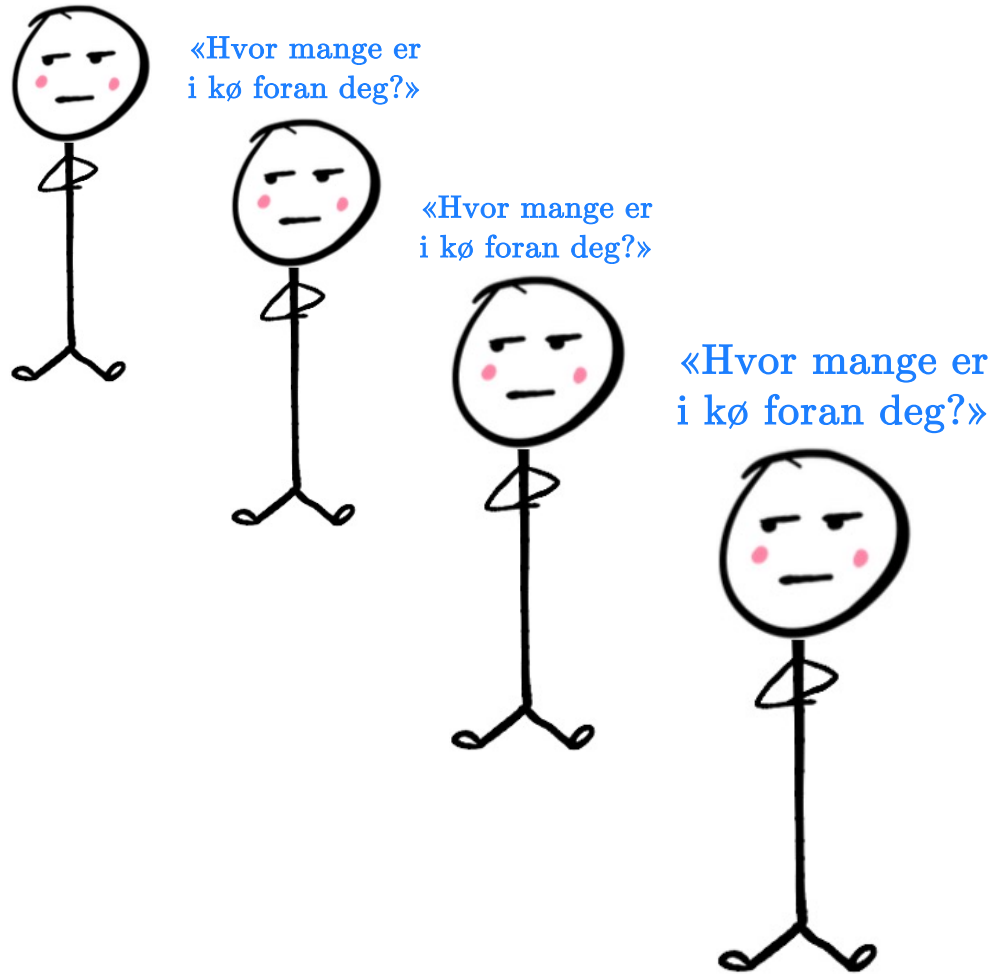


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

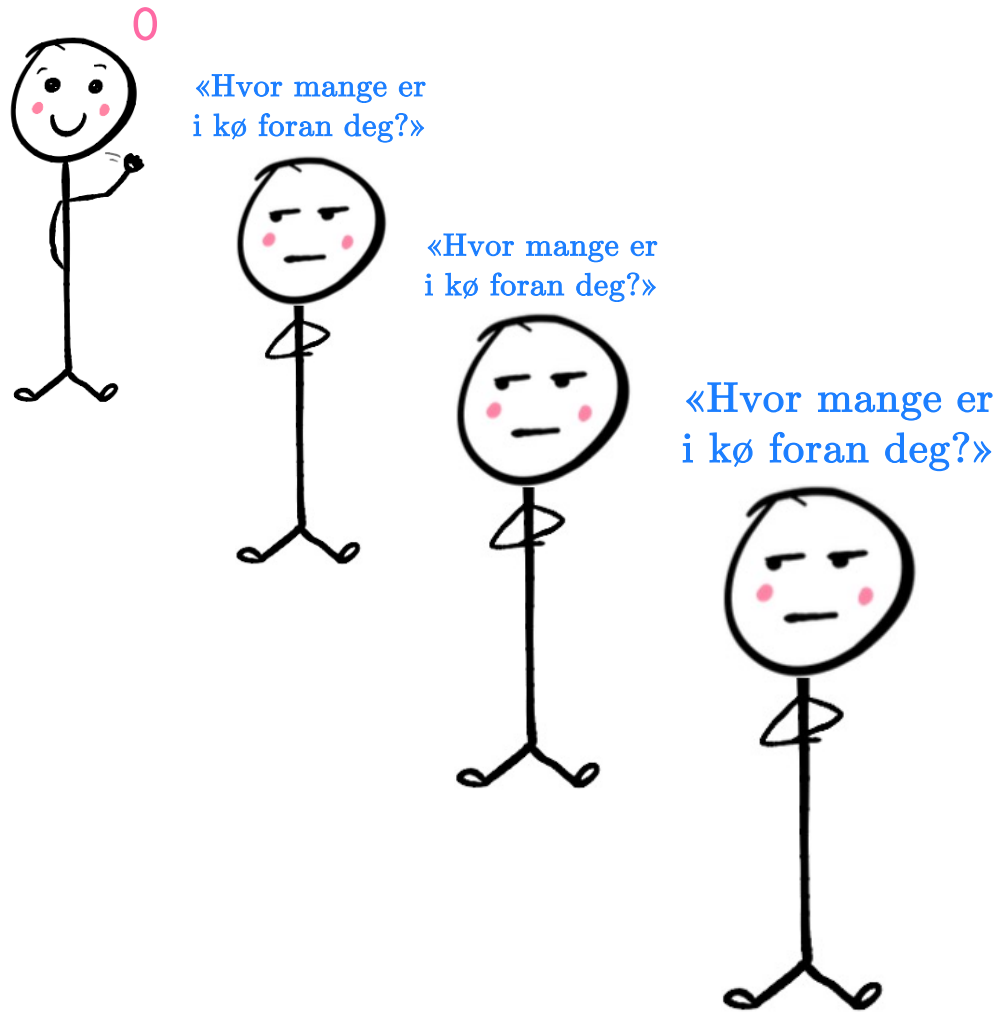


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

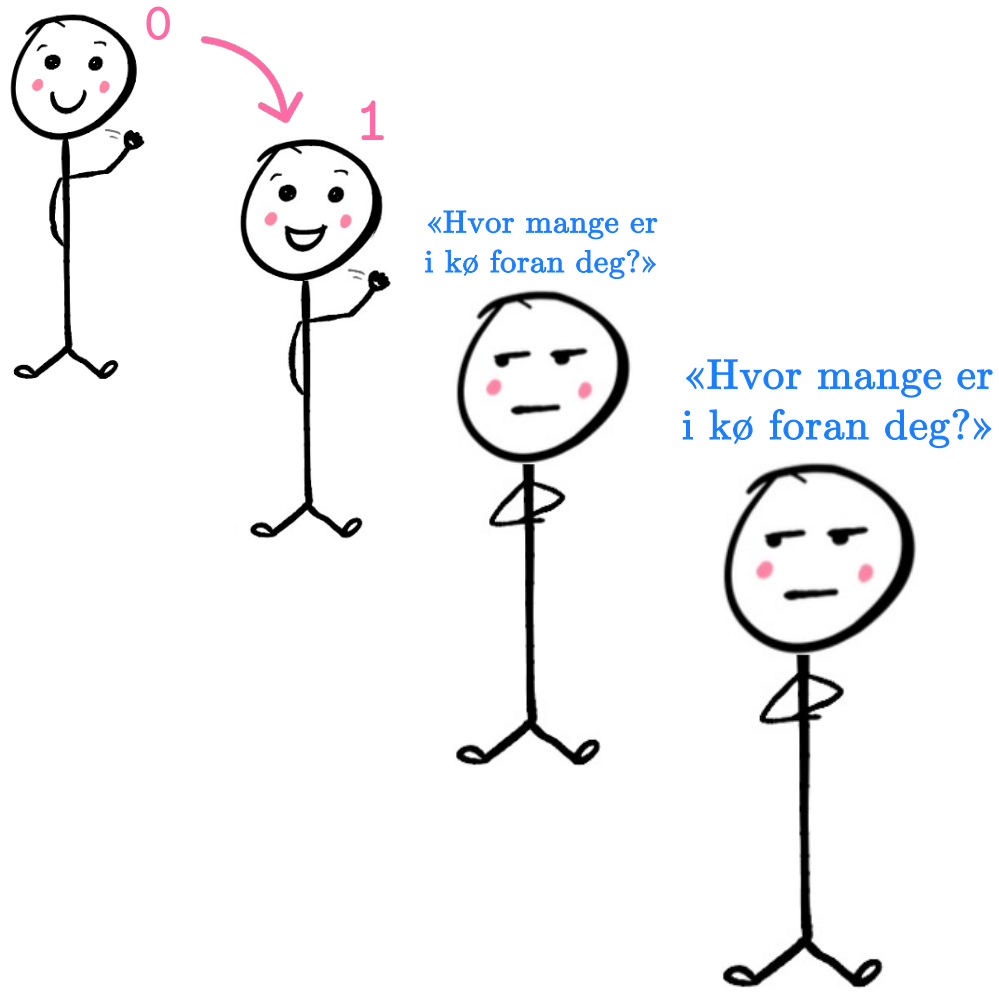


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

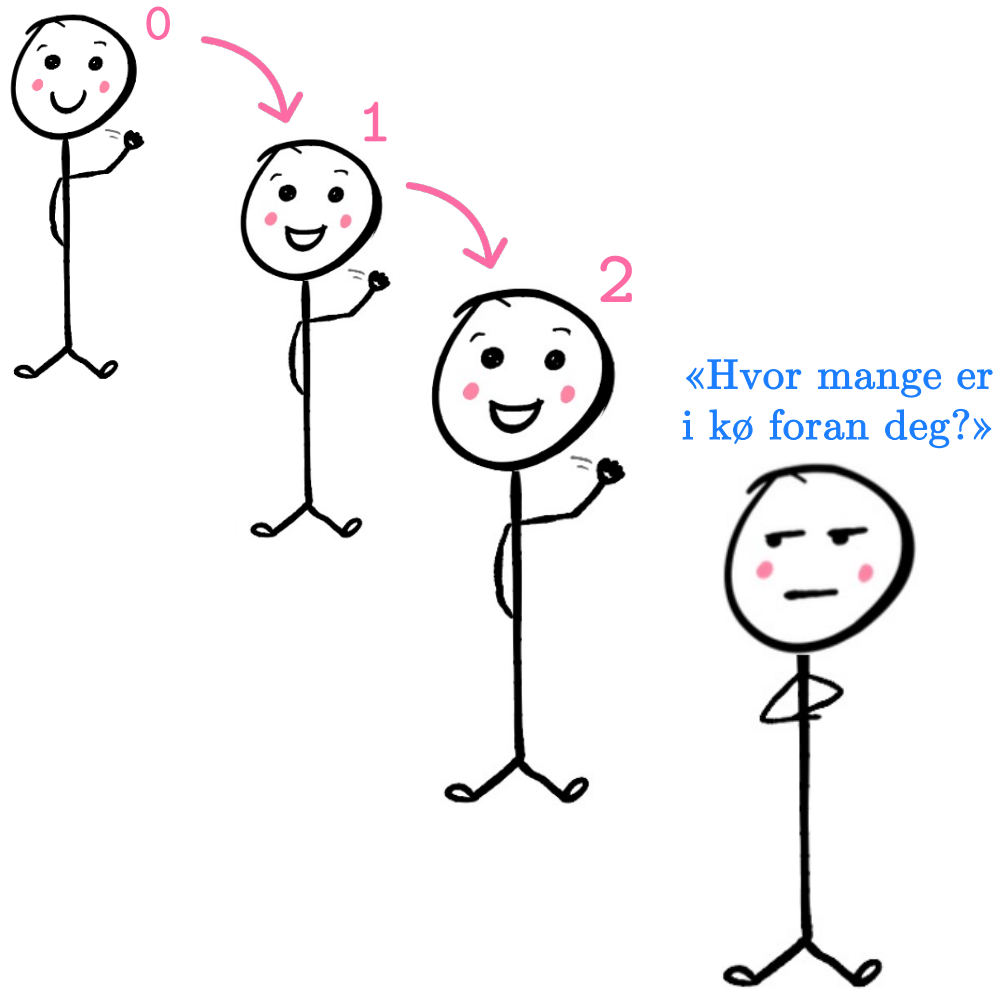


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**

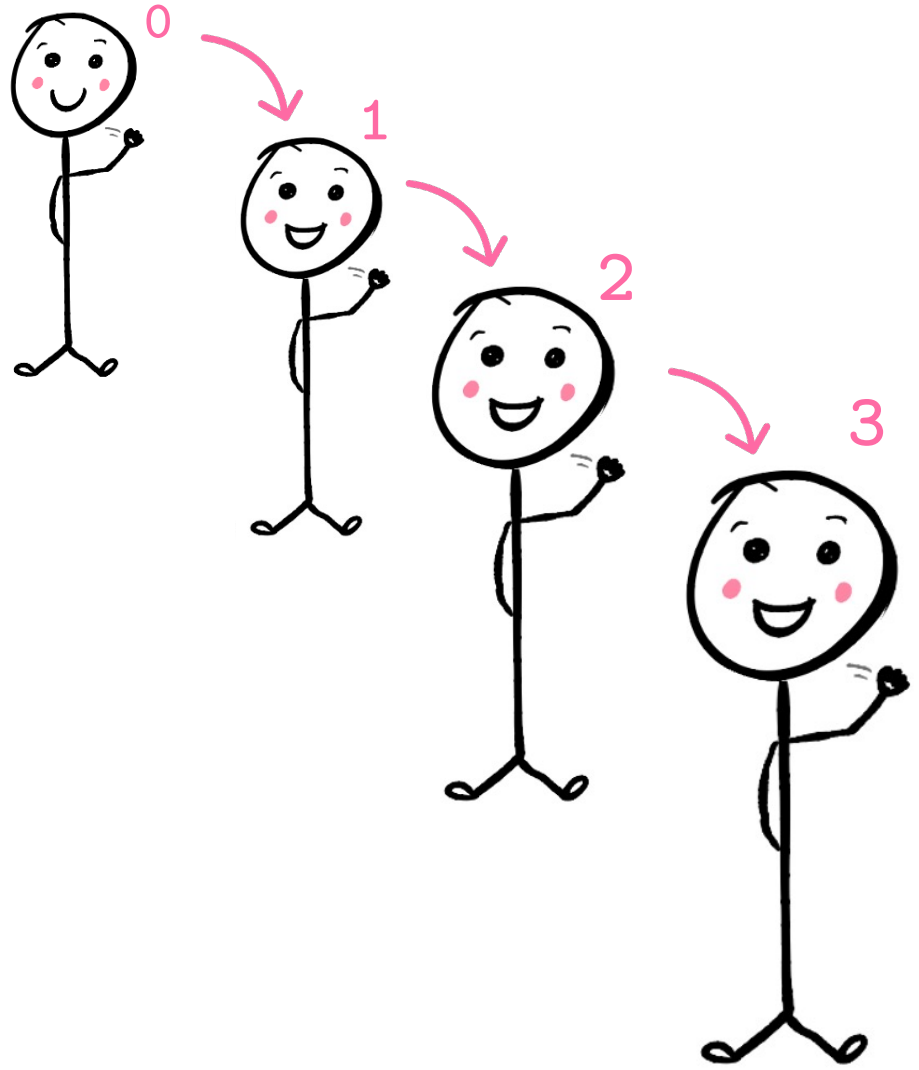


Rekursjon

- En **rekursiv** metode kaller seg selv

- Istedenfor å prøve å løse hele problemet, reduserer vi det til et enklere problem
  - Løser kun neste steg
    - Rekursivt kall

  ...helt til vi når **basistilfellet**



Rekursjon

- Tips:
  - Forsøk først å finne basistilfellet
    - Unngå «evig» rekursjon
  - Tegn opp call stacken for å f.eks. se hva som skrives ut

Rekursjon - eksempel

```
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```
fakultet(3)
```

Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```
fakultet(3)
```

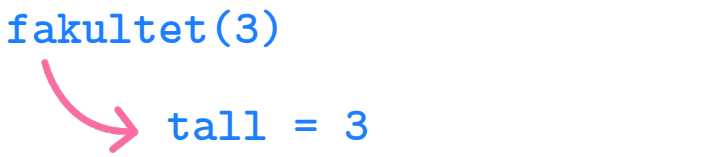
`fakultet(3)`

Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```java
fakultet(3)
```

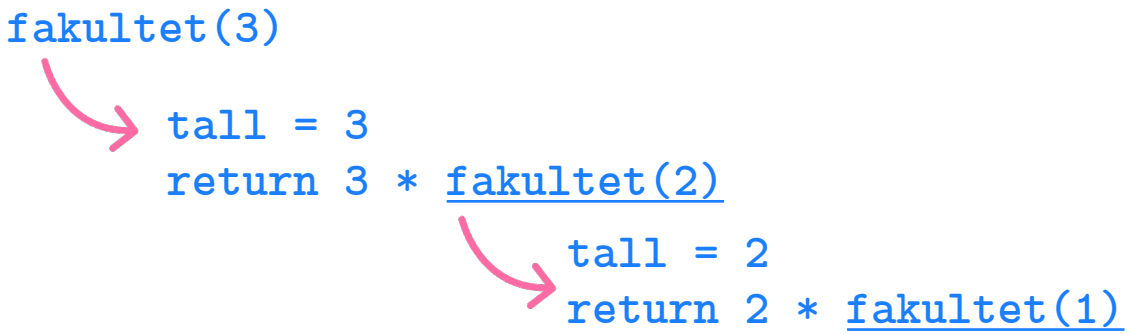


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```
fakultet(3)
```

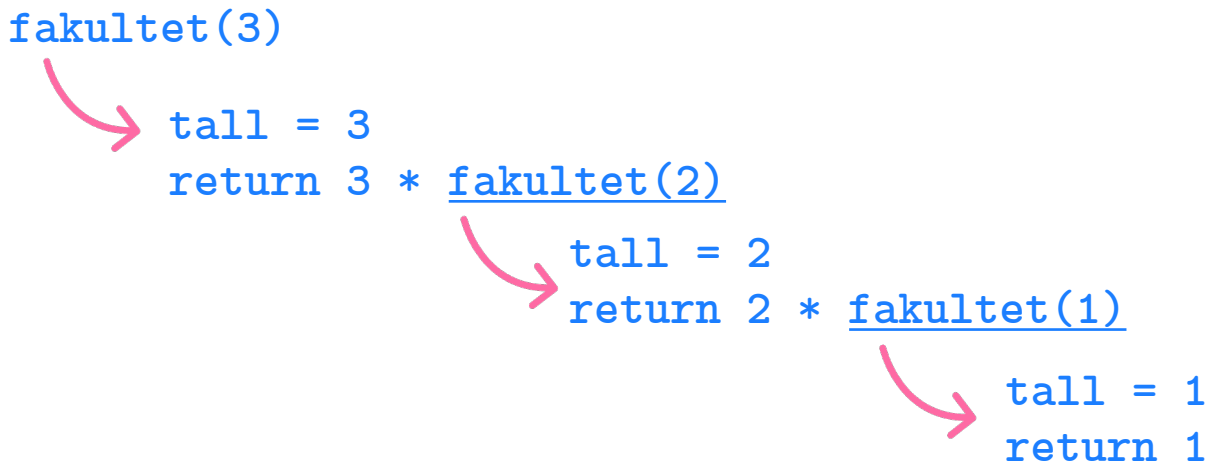


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```java
fakultet(3)
```

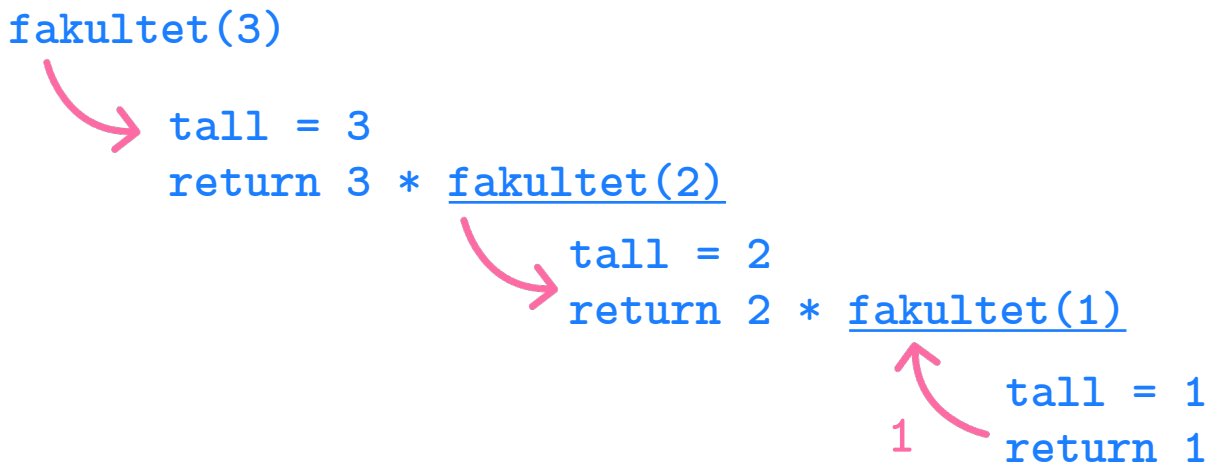


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```java
fakultet(3)
```

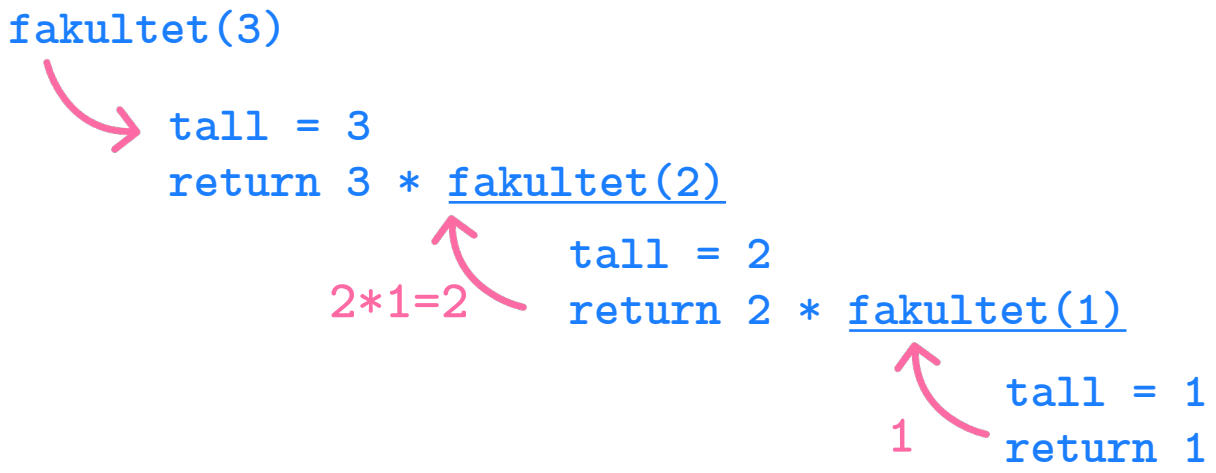


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```java
fakultet(3)
```

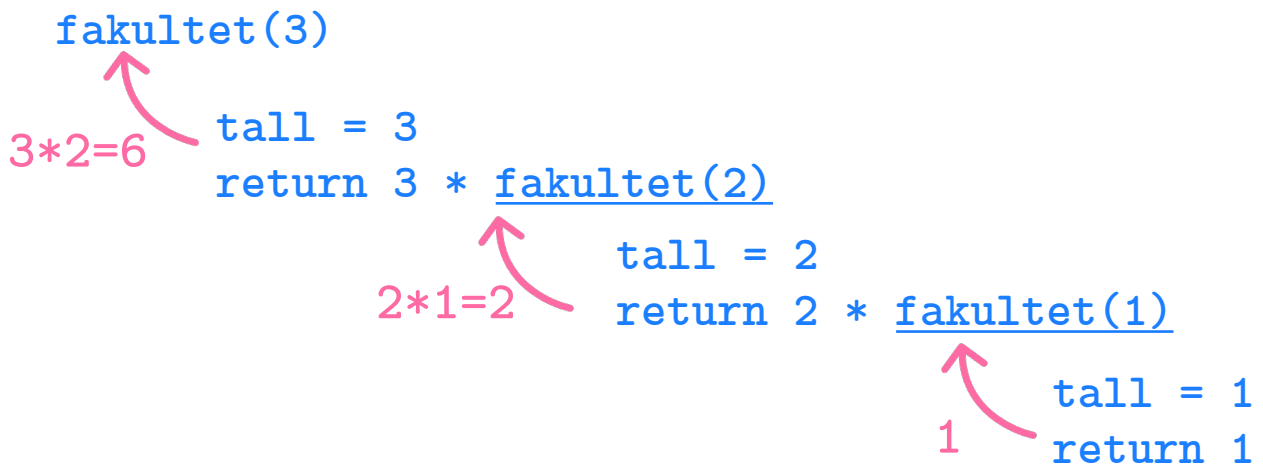


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

```java
public int fakultet(int tall){
    if (tall <= 1){
        return 1;
    }
    return tall*fakultet(tall-1);
}
```

```java
fakultet(3)
```



Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Oppsummering

- **Arv og subklasser**
  - Arv
  - Komposisjon
  - Polymorfi
  - Interface
  - Casting
  - `final` og `static`
- **Beholdere**
  - array
  - ArrayList
  - Lenkeliste
  - Generiske klasser
  - Kø, stabel, prioritetskø
  - Comparable
- **Rekursjon**

Oppsummering

- Andre relevante temaer:
  - Exceptions
  - Scanner
    - Fillesing/-skriving
    - Input
  - Datastrukturtegning
  - Iterator
  - Overloading

Lykke til!