Repetisjon

IN1010

Annika Willoch Olstad

14.05.24

Hva skjer fremover?

- Siste uke med ordinære gruppetimer
  - Siste gruppetime er **16.mai**
  - Repetisjonstimer **21.-24.mai**

- Mer eksamensinfo neste gang
  - I dag er fokus på repetisjon

Repetisjonstimer

Siste ordinære gruppetime i IN1010 denne våren er **16.mai**. Etter dette holdes det repetisjonstimer med fokus på enkelttemaer frem til og med **24.mai**. Du finner en oversikt over tidspunkt og rom nedenfor. Endringer kan forekomme.

Du kan møte opp på så mange repetisjonstimer du ønsker. :)

Tidspunkt	**Tirsdag 21.mai**	**Onsdag 22.mai**	**Torsdag 23.mai**	**Fredag 24.mai**
kl.10.15-12.00	**Tema:** Arv **Rom:** C **Hvem:** Victor	**Tema:** Beholdere **Rom:** Prolog **Hvem:** Sander	**Tema:** GUI **Rom:** Caml **Hvem:** Robert	**Tema:** Gjennomgang av midtveis **Rom:** C **Hvem:** Victor
kl.12.15-14.00	**Tema:** Beholdere **Rom:** Caml **Hvem:** Sander	**Tema:** Tråder **Rom:** Caml **Hvem:** Ahmed	**Tema:** Tråder **Rom:** Fortress **Hvem:** Ahmed	**Tema:** GUI **Rom:** C **Hvem:** Robert
kl.14.15-16.00	Forelesning	**Tema:** Rekursjon **Rom:** Caml **Hvem:** Bushra	**Tema:** Rekursjon **Rom:** C **Hvem:** Bushra	Fredagsjava🎉

Dagens plan

1. Læringsmål
2. Gjennomgang av temaer
   - Tidligere repetisjon av første halvdel
3. Eksempler på hvordan det kan se ut på eksamen
   - Basert på tidligere eksamener

Læringsmål

Etter å ha tatt IN1010:

- har du god oversikt over programmeringsspråket Java og du kan bruke det til å løse reelle problemer av middels størrelse
- behersker du avanserte objektorienterte mekanismer som subklasser, polymorfi og interface
- har du oversikt over noen grunnleggende datastrukturer (spesielt lenkete lister) og du kan programmere de viktigste operasjonene på dem
- kan du utvikle robuste og pålitelige programmer med godt objektorientert design, og du kan finne alternative løsninger for et gitt problem og vurdere fordeler og ulemper ved de ulike løsningene
- har du kunnskap om parallelle programmer med tråder og du kan benytte dette i enkle programmer
- kjenner du til hendelseshåndtering og kan skrive enkle programmer som håndterer hendelser

https://www.uio.no/studier/emner/matnat/ifi/IN1010/#learning_outcomes

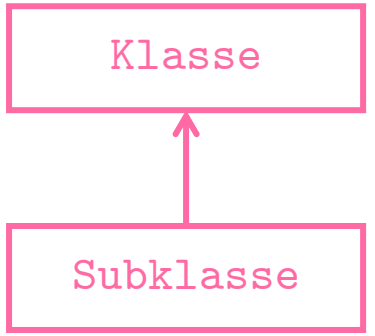
Oversikt

1. Arv og subklasser
2. Beholdere
3. Rekursjon
4. Tråder
5. GUI

Arv og Subklasser

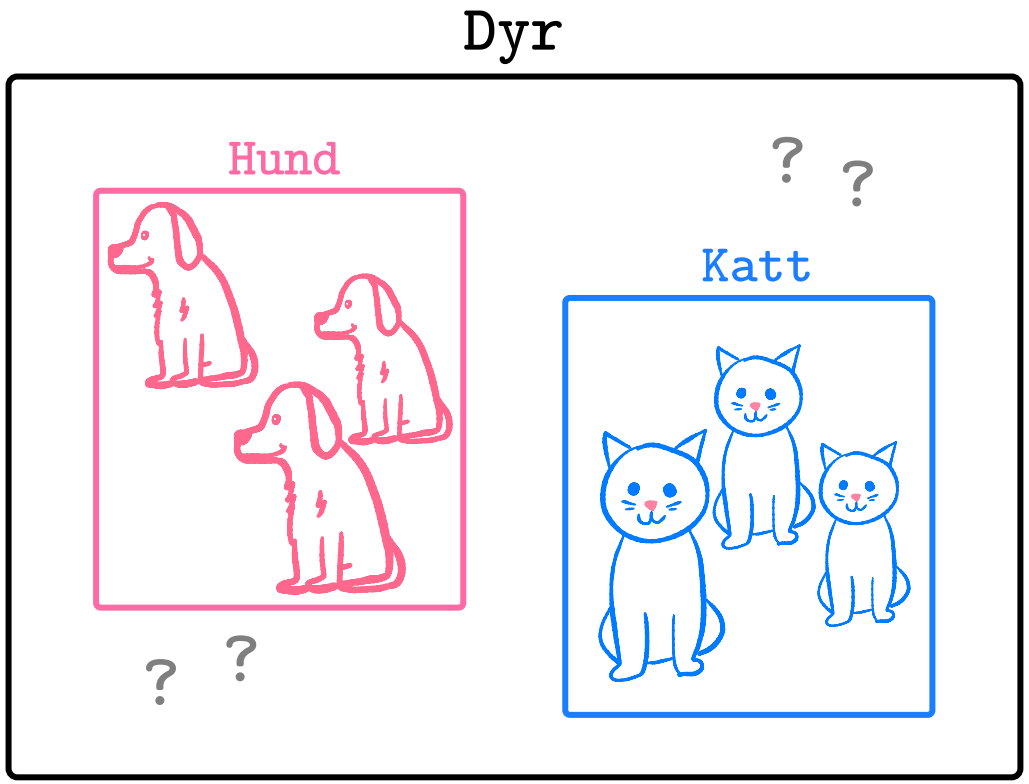
Arv og Subklasser 1

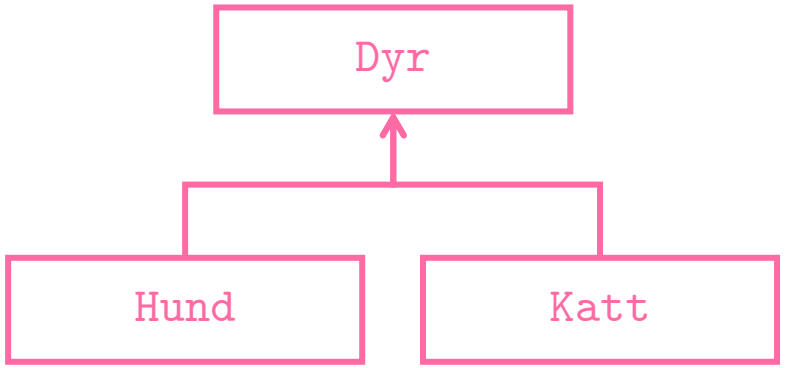
- **Klasse**: beskriver objekter med visse egenskaper
- **Subklasse**: beskriver objekter med samme egenskaper som beskrevet av **Klasse** (*superklassen)*
  - Men har i tillegg noe mer/spesielle egenskaper
- Nøkkelord: `extends`
  - Arve fra en klasse
- Nøkkelord: `super`
  - Aksessere variabler/metoder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




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

Arv og Subklasser 1 - Eksempel






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

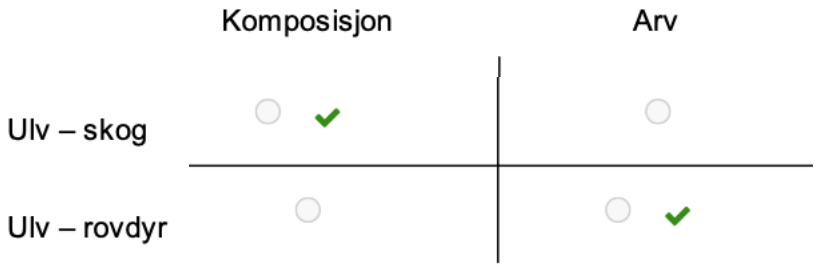
Arv og Subklasser 2

- **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 prøvemidtveis
  - 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 prøvemidtveis

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!");}
}

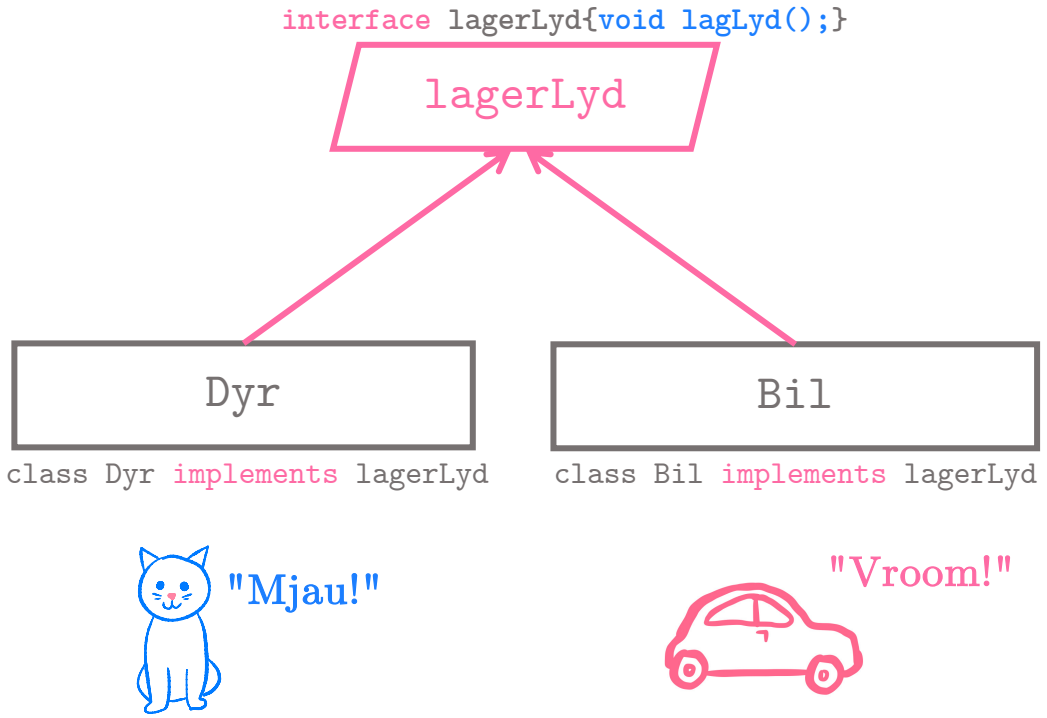
class Hund extends Dyr {
    @Override
    public void lagLyd(){System.out.println("Voff!");}
}

class Katt extends Dyr {
    @Override
    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`



Casting

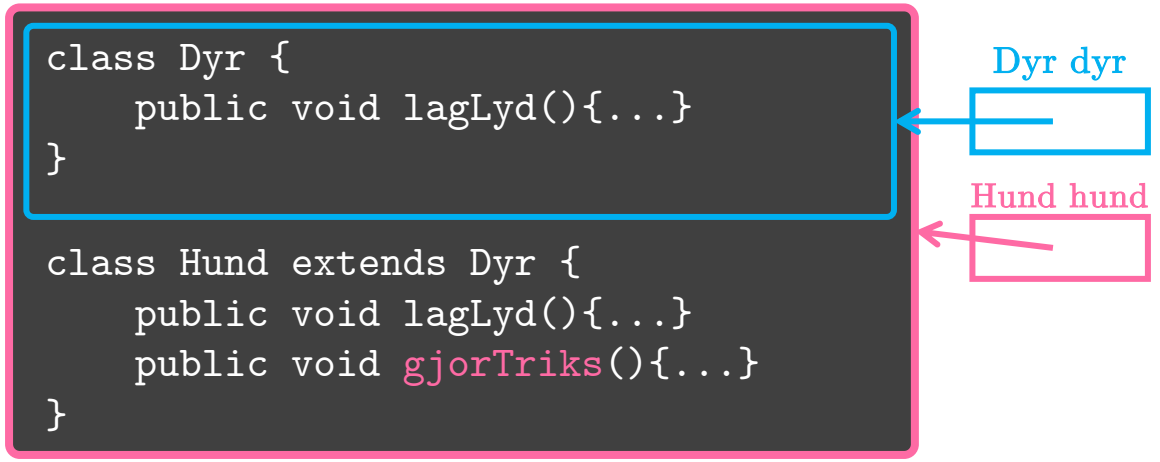
- *Referansens type* avgjør hvordan objektet fremtrer
  - Ser på objektet med ulike «briller»
  - Hvilke egenskaper har vi tilgang til?
- **Casting/typekonvertering**
  - Lar oss «bytte briller»

Casting - eksempel

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

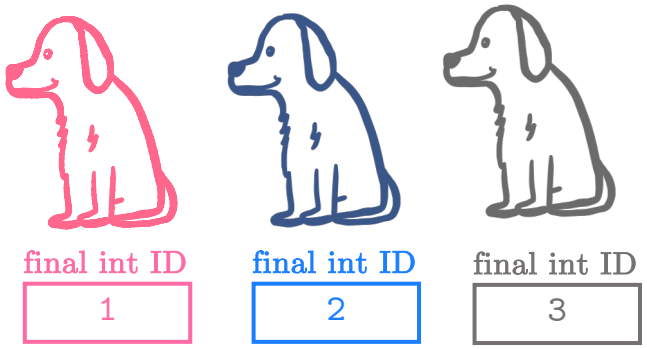
    Hund hund = (Hund) dyr;
    hund.gjorTriks();

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



`final` og `static`

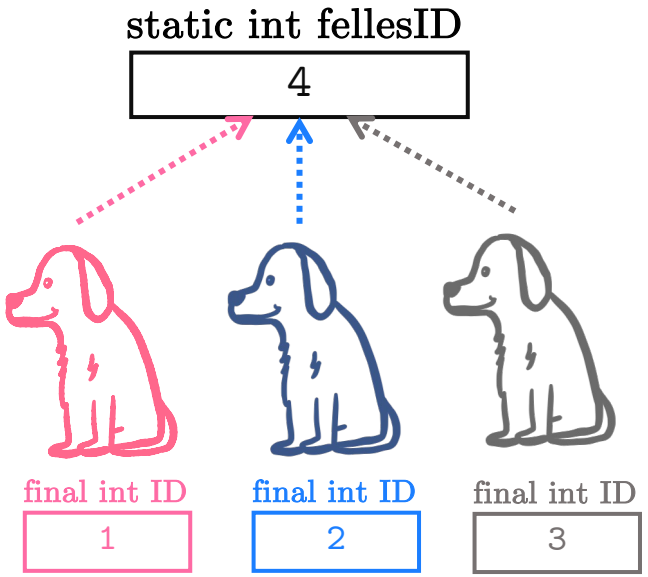
- `final`: nøkkelord for **konstanter**
  - Kan ikke endres etter initialisering



`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



På eksamen

Utsatt eksamen 2023

Oppgave 1 Tegning av klassehierarki (5 poeng)

⇨ Tegn et klassehierarki som kan representere ulike typer biler som beskrevet over. Begrunn valgene dine kort som kommentarer i programmet du skriver i oppgave 2.

Oppgave 2 Programmering av klassehierarki (15 poeng)

Bilene i en kolonne skal kunne hektes sammen i en dobbeltlenket liste ved hjelp av instansvariabler i bil-objektene. Hver bil skal også inneholde en referanse til den kolonnen den kjører i. Denne referansen skal være **null** når bilen ikke er i noen kolonne.

Enhver bil har en maksimal fart. Enhver lastebil har en maksimal lastevekt og personbiler har et maksimalt antall passasjerer. Disse instansvariablene skal initieres av klassenes konstruktører. Lederbiler har en egenskap kalt egnethet som er et heltall. Lederbilenes egnethet skal initieres av konstruktørene.

⇨ Programmer klassene og interfacet du tegnet i oppgave 1

På eksamen

Eksamen 2023

Oppgave 1. Klassehierarki for skinnegående materiell (5 poeng)

Alle vogner og annet materiell som kan kjøre på en skinnegang, representeres av klassen **Skinnegående** som har følgende metoder:

hentId returnerer en String som identifiserer dette skinnegående objektet.

hentSporvidde returnerer et heltall som angir hvor brede spor det kan kjøre på.

Sporvidden og identifikatoren skal være parametere til klassens konstruktør. Skinnegående objekter skal kunne hektes sammen i en dobbeltlenket liste i en senere oppgave, så du skal også ha med instansvariabler i klassen som gjør dette mulig.

I dette eksamenssettet er alt skinnegående materiell enten av klassen **Lokomotiv** eller klassen **Vogn**. Alle Vogn-objekter er enten av klassen **Godsvogn** eller klassen **Passasjervogn**.

Skinnegående materiell kan implementere et interface **Motordrevet** med følgende grensesnitt:

fossilt som returnerer true dersom fossilt brennstoff (bensin, diesel, kull, ved) benyttes, ellers false.
trekkraft som returnerer antall kW (kilowatt) som et heltall.

Alle vogner har en lengde angitt som heltall i centimeter. Godsvogner har en maks tillatt lastevekt, angitt som et desimaltall i kilogram, og passasjervogner har et heltall som representerer maks antall passasjerer det er plass til i vognen. Disse instansvariablene skal være parametere til konstruktørene til klassene.

- Tegn et klassehierarki for situasjonen beskrevet over.

Oppgave 2. Programmere klasser og interface (15 poeng)

I denne oppgaven skal du programmere klasser og interface som beskrevet i oppgave 1.

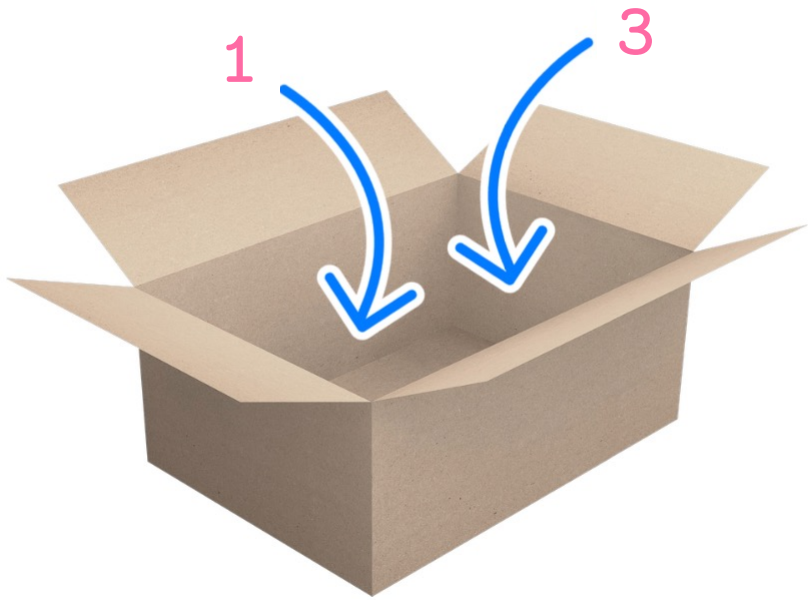
- Programmer klassene **Skinnegående, Lokomotiv, Vogn**, **Godsvogn**, **Passasjervogn** og interface-et **Motordrevet**.

Beholdere

og generiske klasser

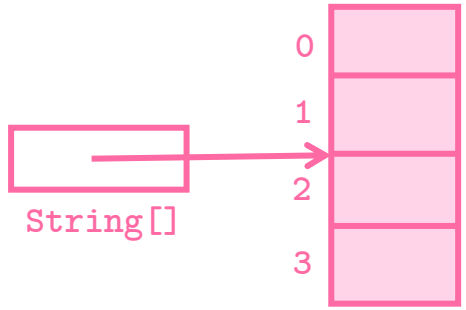
Beholdere

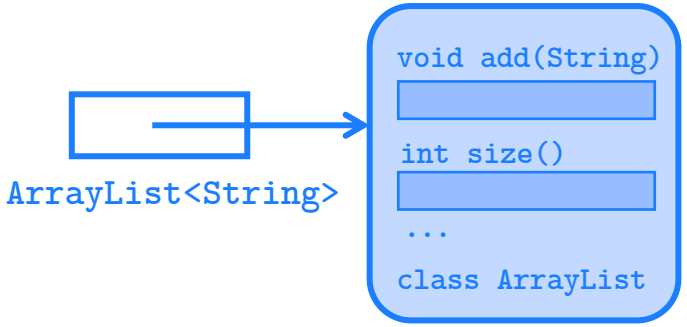
- Samling elementer
- I Java:
  - array
  - ArrayList
  - HashMap
  - ...
- Lenkeliste



Array og ArrayList

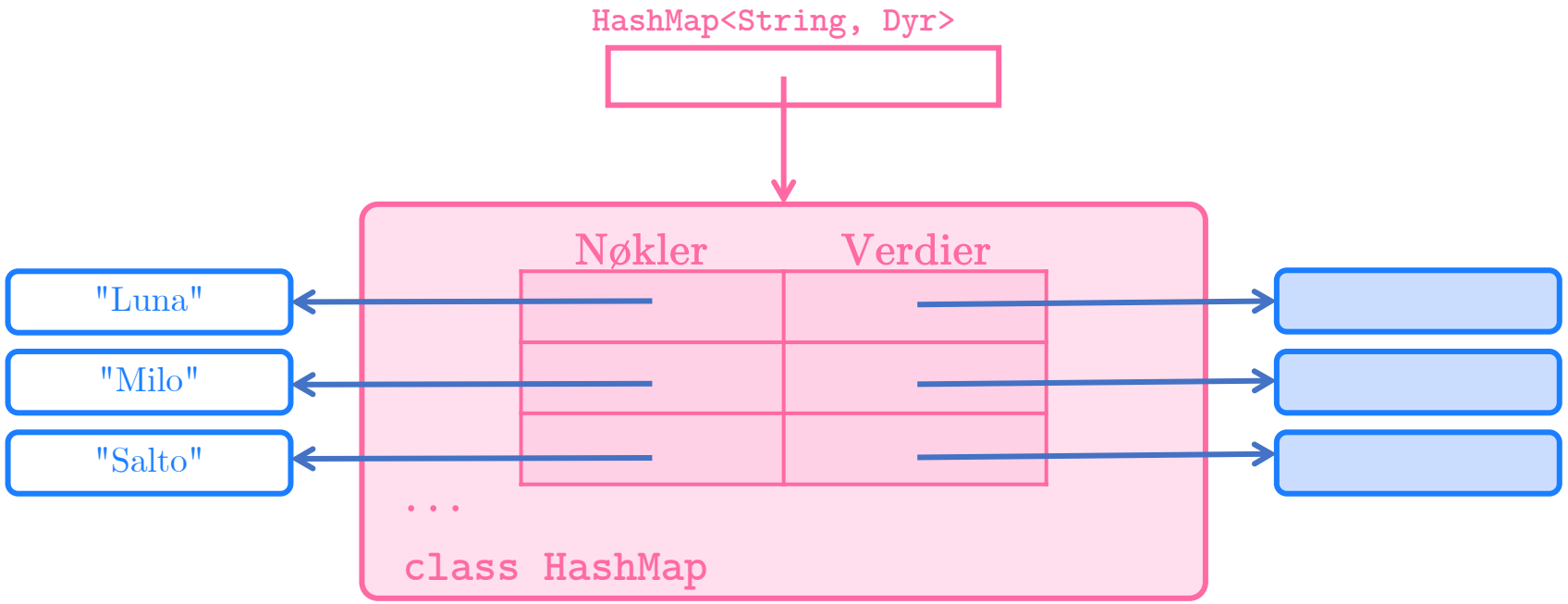
- **array**: sammenhengende gruppe i minnet som holder på et gitt antall verdier av samme type
  - Konstant størrelse
  - Tilordner og henter verdier med indekser (plassering i arrayet)
- **ArrayList**: fleksibel listeklasse i Java-biblioteket
  - Kan utvides/trekkes sammen etter behov
  - Men tar mer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





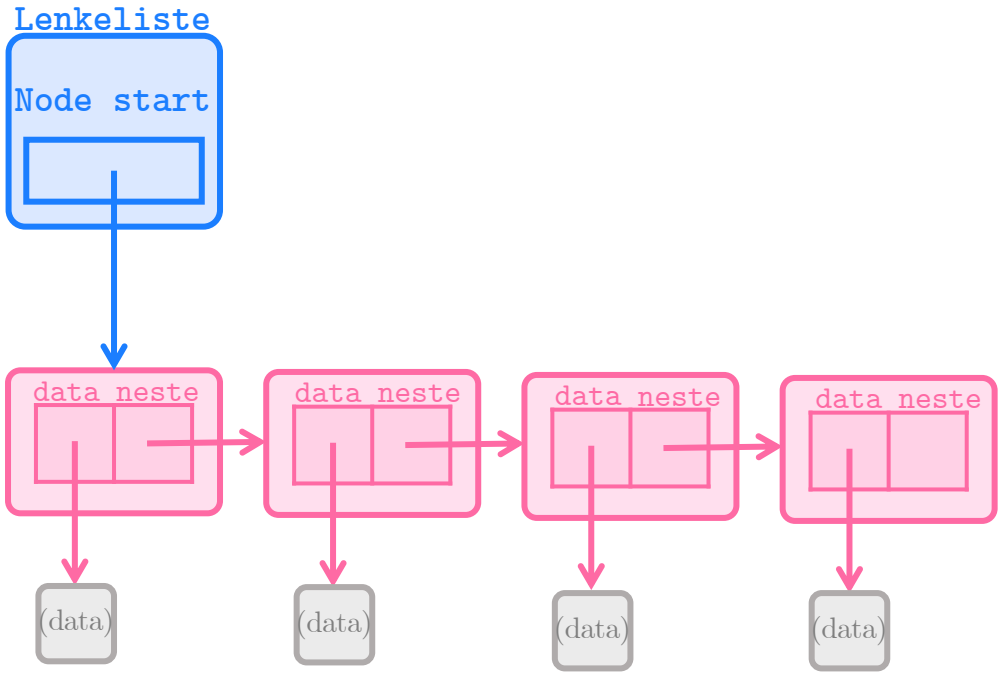
HashMap

- Samling hvor elementer identifiseres med en nøkkel



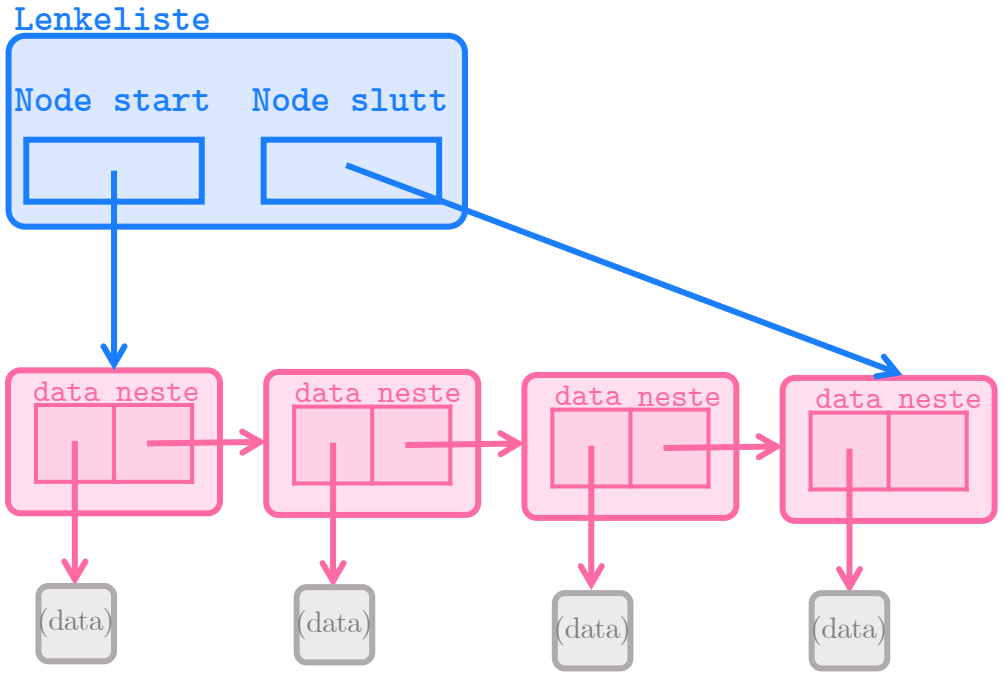
Lenkeliste

- Listestruktur med noder som er lenket sammen
  - **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



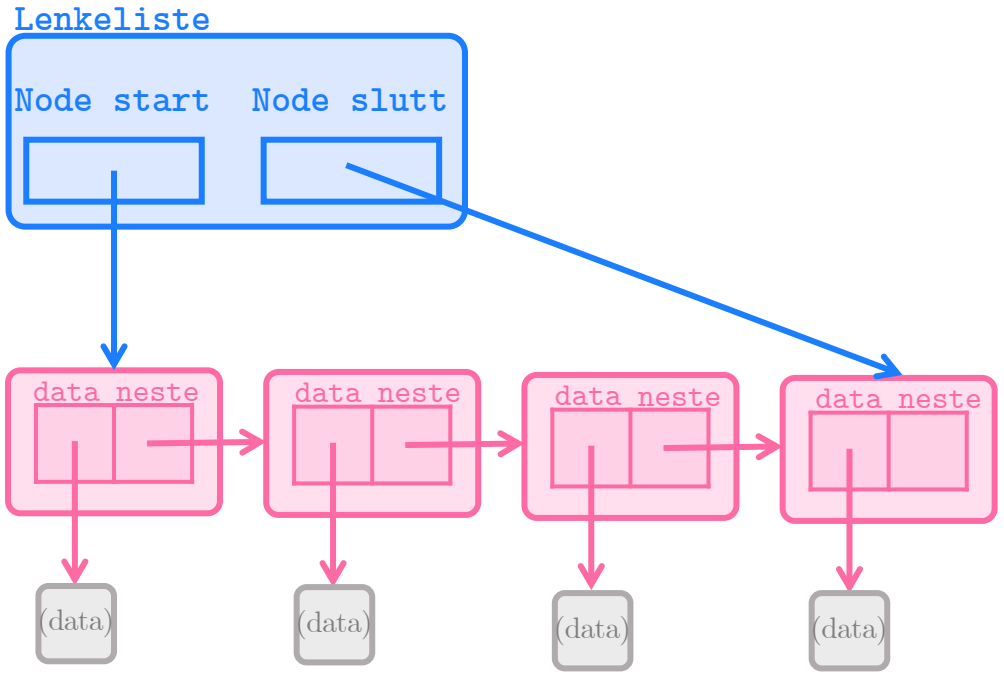
Lenkeliste

- 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

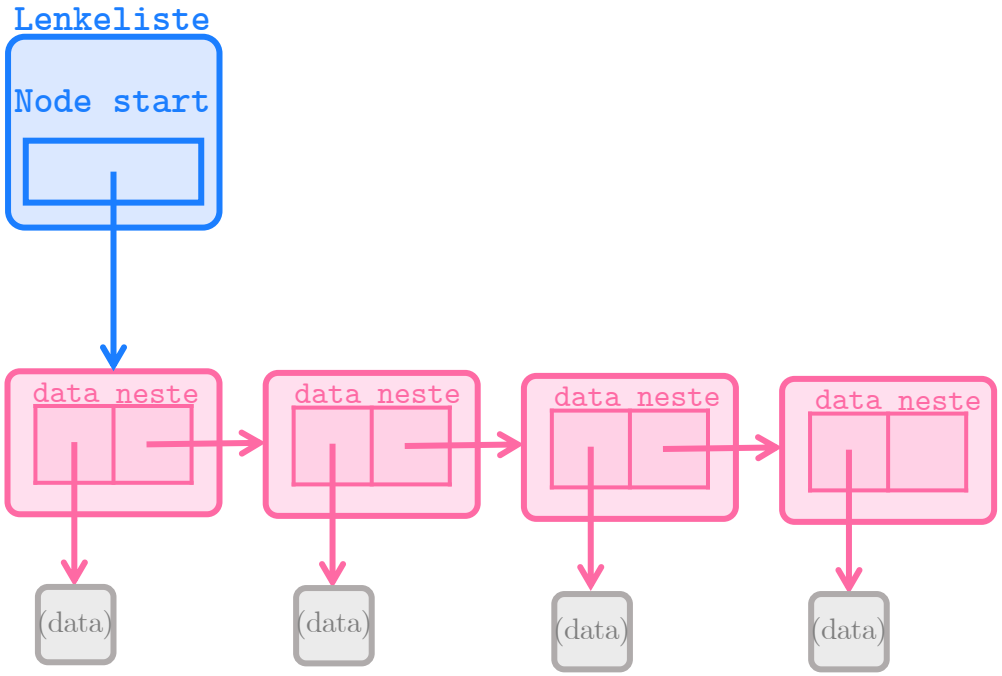
- Listestruktur med noder som er lenket sammen
  - **Node**: indre klasse
    - Holder på **data** og **referanse til neste node** i listen (*enkellenket*)
- Lenkelisten har en **start-**variabel
  - Refererer til **første** element i lenken
  - Kan også ha til **siste** element
- Kan være **enkel-**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



Lenkeliste: Enkellenket

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

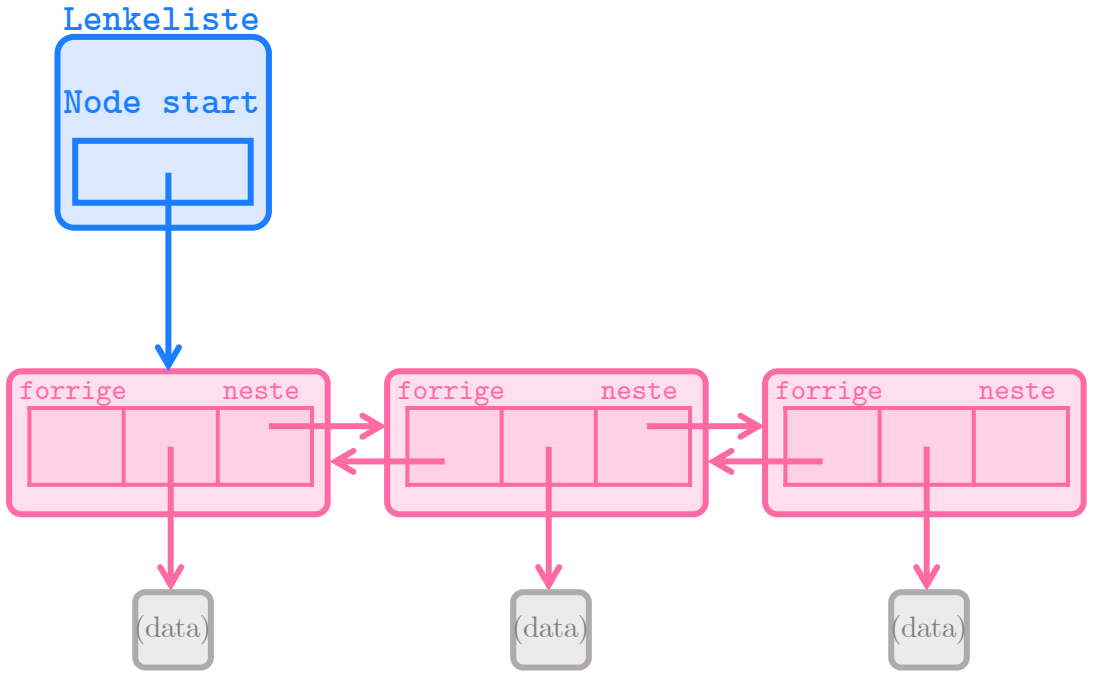
```
class Lenkeliste {
    protected class Node {
        Node neste = null;
        //...data
    }
    protected Node start;
}
```



Lenkeliste: Dobbellenket

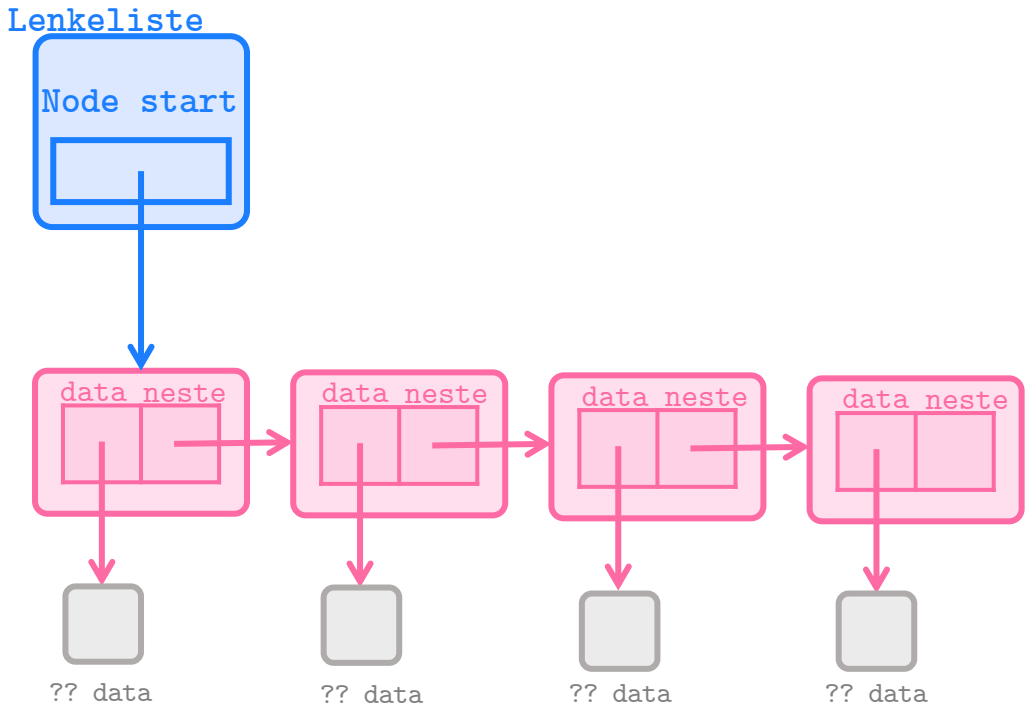
- Dobbellenket liste:
  - **neste**-peker
  - **forrige**-peker

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



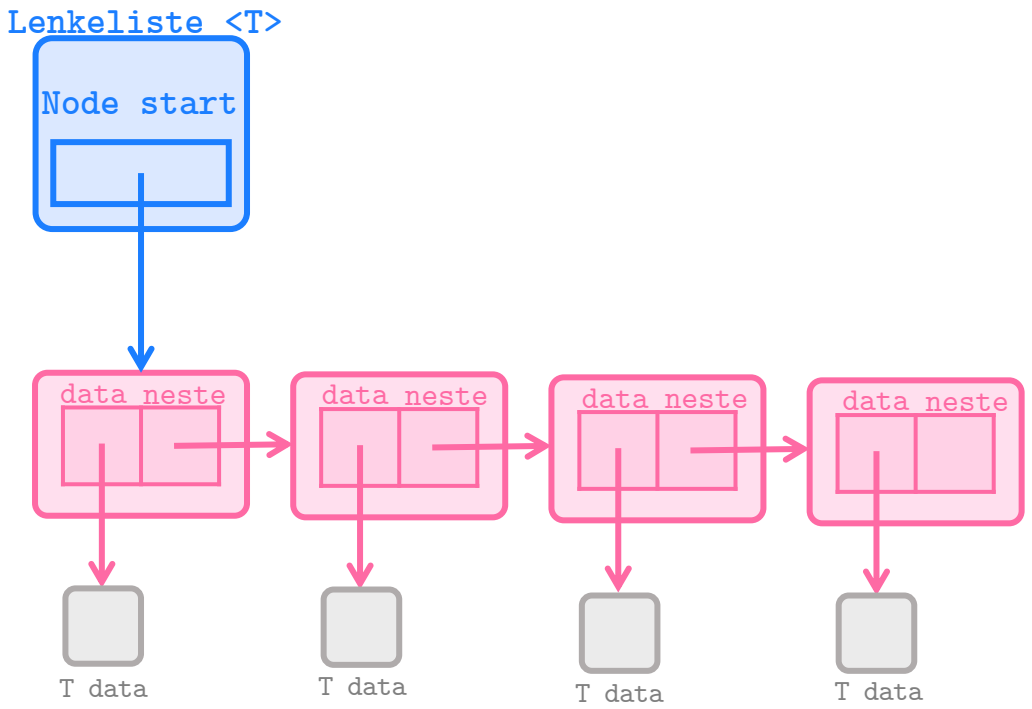
Generiske klasser

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

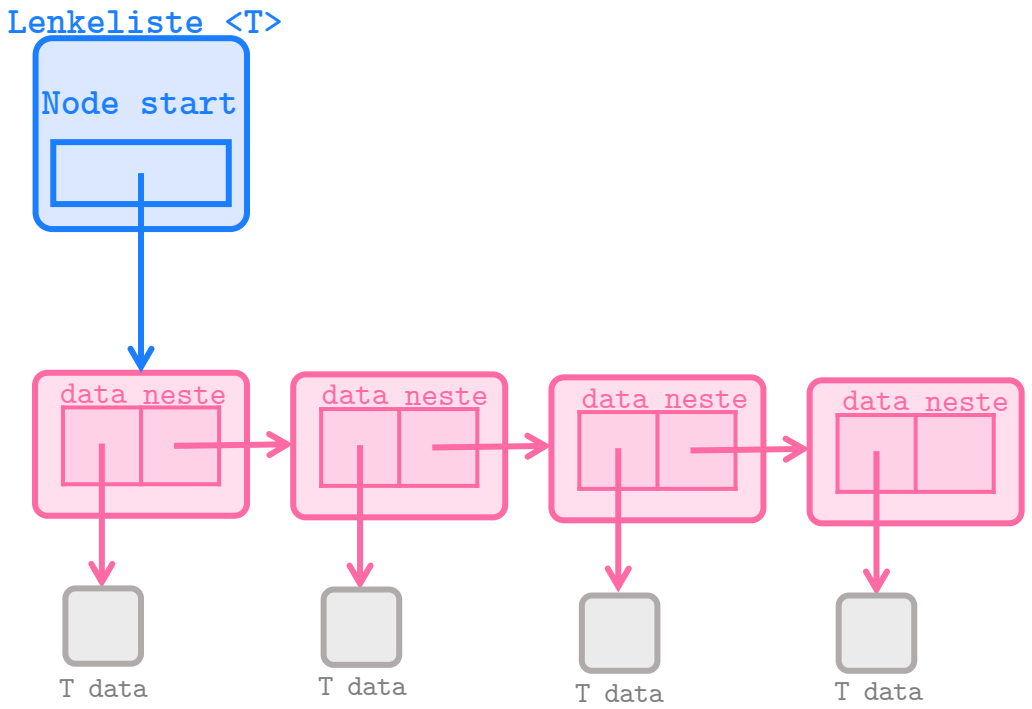
- Hvilken type har dataen vi skal lagre?
  - → **Generiske klasser**
- Klasser med parametere som angir typen til elementene inni klassen
  - **Typeparametre/klasse-parametre**
    - Markeres med f.eks. <T>
  - Nyttig for bl.a. beholdere



Generiske klasser

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

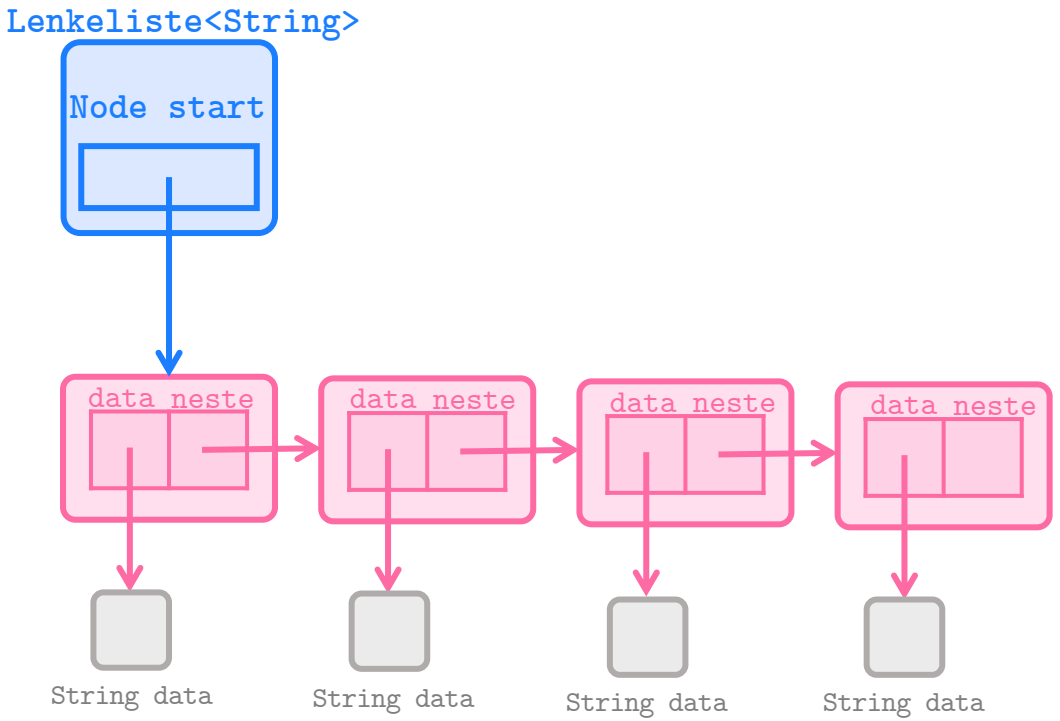
        Node (T x) {
            data = x;
        }
    }
    protected Node start = null;
}
```



Generiske klasser - Eksempel

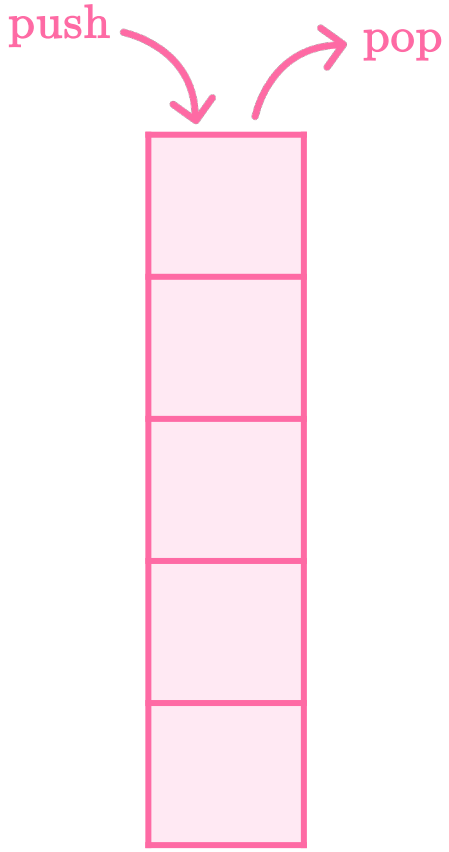
```
class Lenkeliste<T>{
    protected class Node {
        Node neste = null;
        T data;

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



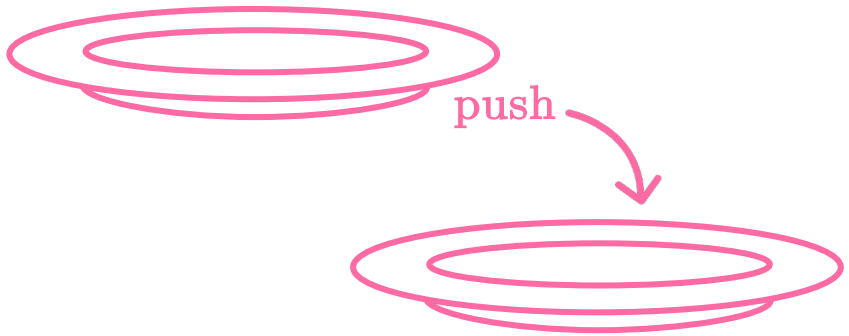
Stabel

- 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



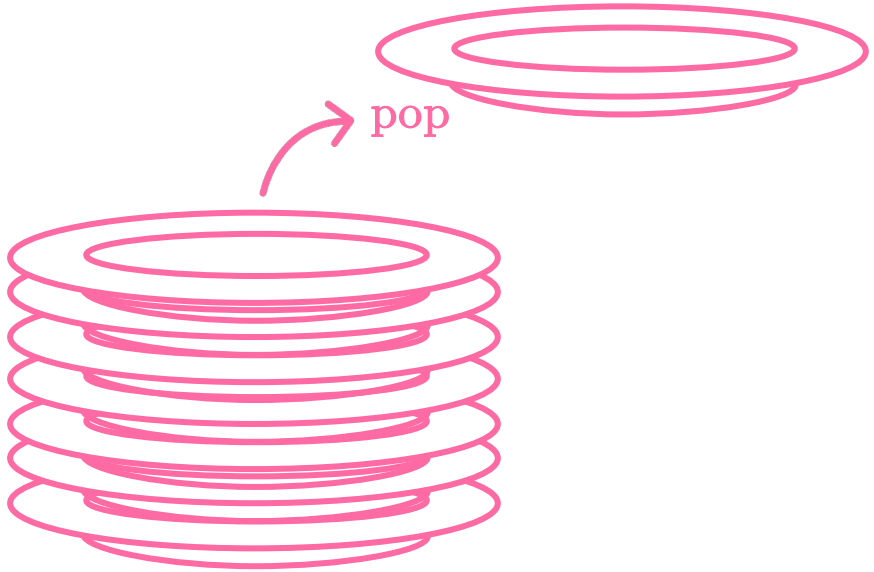
Stabel - Sammenlikning

- 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



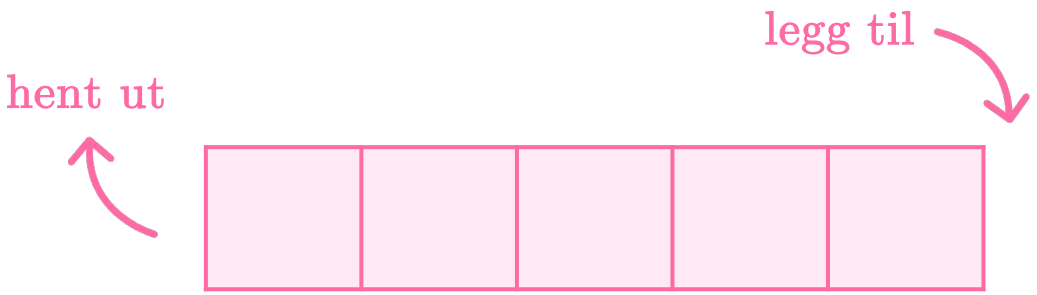
Stabel - Sammenlikning

- 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



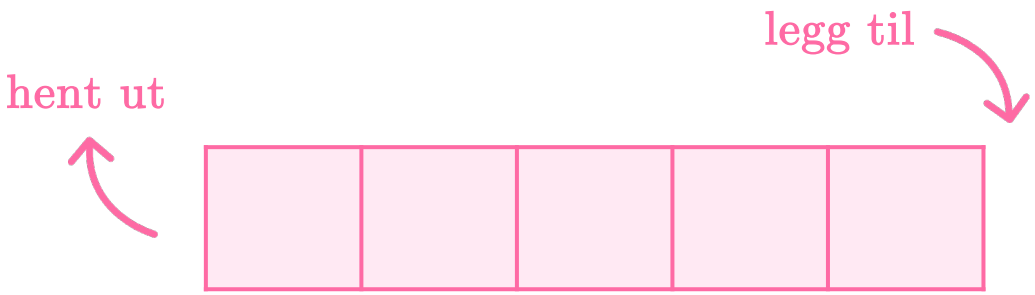
Kø

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



Kø

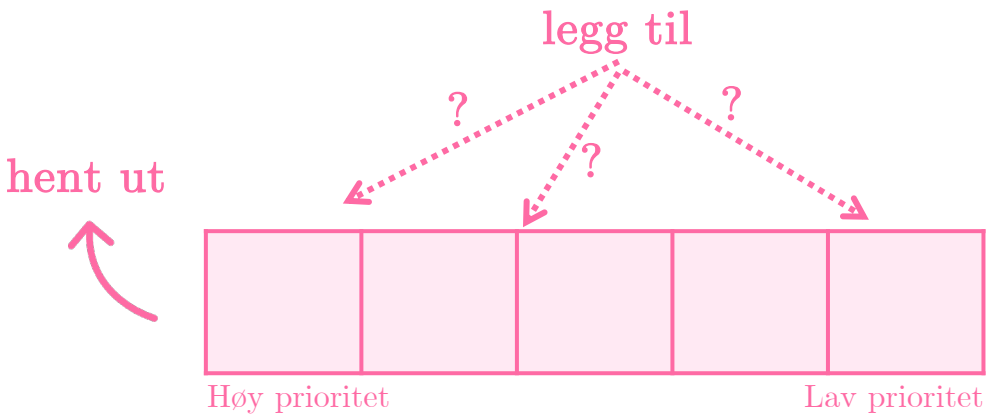
- FIFO: First In First Out
- Legger til elementer bakerst og henter ut fremst





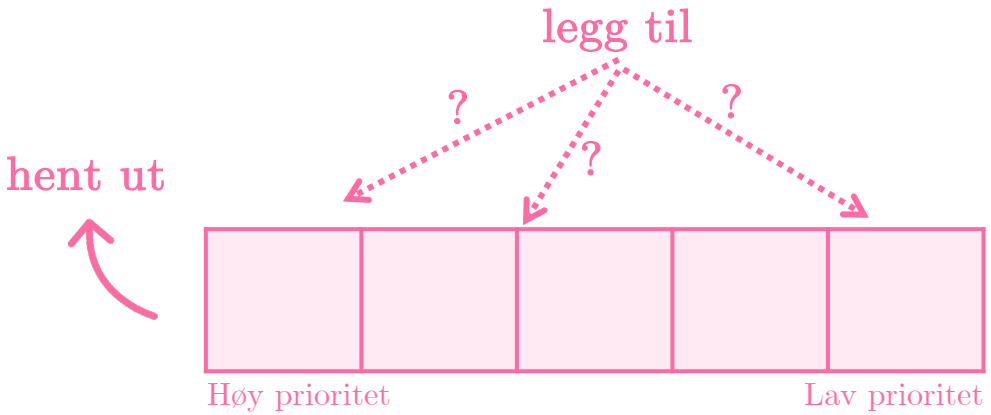
Prioritetskø

- Elementer med høyest prioritet hentes ut først



Prioritetskø

- Elementer med høyest prioritet hentes ut først





På eksamen

Eksamen 2023

Oppgave 3. Klassen Tog (40 poeng)

I oppgave 3 skal du skrive en klasse **Tog** med variabler og metoder. Et tog er en dobbeltlenket liste av Skinnegående-objekter og du skal bruke instansvariablene i disse objektene til å lage denne listen. I denne oppgaven kan du aksessere instansvariabler i Skinnegående-klassen og dens subklasser.

3a) 1 poeng

⇨ Skriv konstruktør og instansvariabler for klassen Tog med pekere til første og siste Skinnegående-objekt i listen.

3b) 4 poeng

⇨ Skriv metoden **leggTil** i klassen Tog som legger til et Skinnegående objekt bakerst i listen. Parameter til metoden er en referanse til objektet som skal legges til.

3c) 7 poeng

⇨ Skriv metoden **taUt** i klassen Tog som tar ut et element i listen. Parameter til metoden er en referanse til objektet som skal tas ut. Du kan anta at objektet som tas ut, ligger i toget. Metoden skal returnere en referanse til det objektet som tas ut.

3d) 7 poeng

⇨ Skriv metoden **finnOgTaUt** i klassen Tog som finner et element i listen og tar det ut. Parameter til metoden er en String som er id-en til det Skinnegående-objektet som skal tas ut (du kan anta at id-en er unik). Metoden skal returnere en referanse til det objektet som tas ut.

3g) 7 poeng

⇨ Skriv metoden **Passasjervogn[] hentPassasjervogner ( )** i klassen Tog som returnerer en array med referanser til alle Passasjervogn-objekter i toget.
Hint: Tell antall passasjervogner før du oppretter arrayen.

3e) 7 poeng

⇨ Skriv metoden **leggTilForan** i klassen Tog som har to parametere av typen Skinnegående. Den første parameteren er en referanse til et objekt du kan anta allerede ligger i toget, den andre er en referanse til et objekt som skal settes inn i listen foran objektet som refereres av den første parameteren.

3f) 7 poeng

⇨ Skriv en iterator over alle Skinnegående-objektene i et Tog.

På eksamen

Eksamen 2019

3(a) KullListe.settInn

Subklassene KullListe og KullArray

I oppgave 3 skal du programmere to subklasser av klassen Kull for å holde oversikt over hundene i et kull: KullListe og KullArray. Begge disse klassene skal ha en metode settInn for å legge nye hunder til et kull, men de skal ikke ha noen metode for å fjerne hunder. Metoden iterator skal lage en iterator for hundene i kullet, men du skal bare lage den for KullListe. (Klassen KullArray skal også inneholde en iterator-metode, men du skal *ikke* skrive den.)

I klassen KullListe skal alle hundene i kullet være lenket sammen i en liste, og du skal bruke variabelen neste i klassen Hund til dette. Når du itererer over hundene i kullet, skal de yngste komme først, så du må la listen være sortert slik at yngre hunder kommer foran eldre. Du kan ikke anta noe om rekkefølgen hundene blir lagt inn i kullet, så programmet ditt må sørge for at listen av hunder er sortert. I oppgavene 3a og 3b skal du til sammen skrive hele klassen KullListe, og i oppgavene 3c og 3d skal du skrive hele klassen KullArray (unntatt iteratoren i KullArray).

Oppgave

Programmer klassen KullListe med metoden settInn.

Maks poeng: 15

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

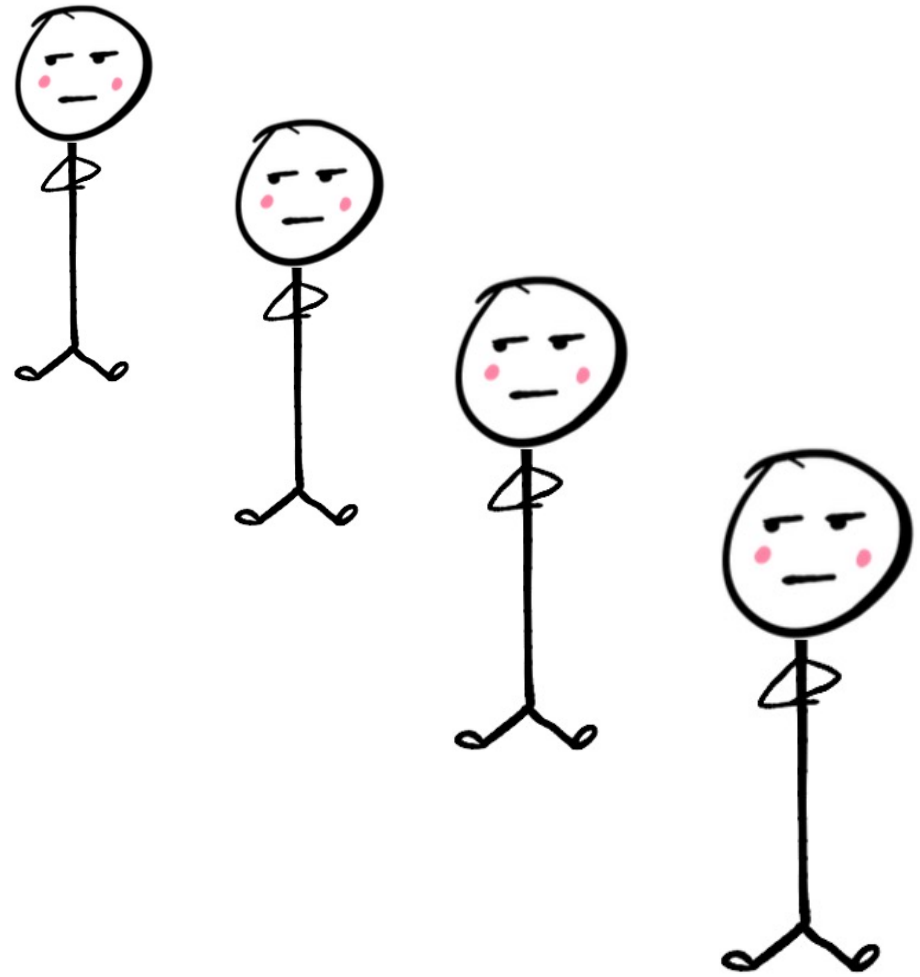
  ...helt til vi når **basistilfellet**

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**

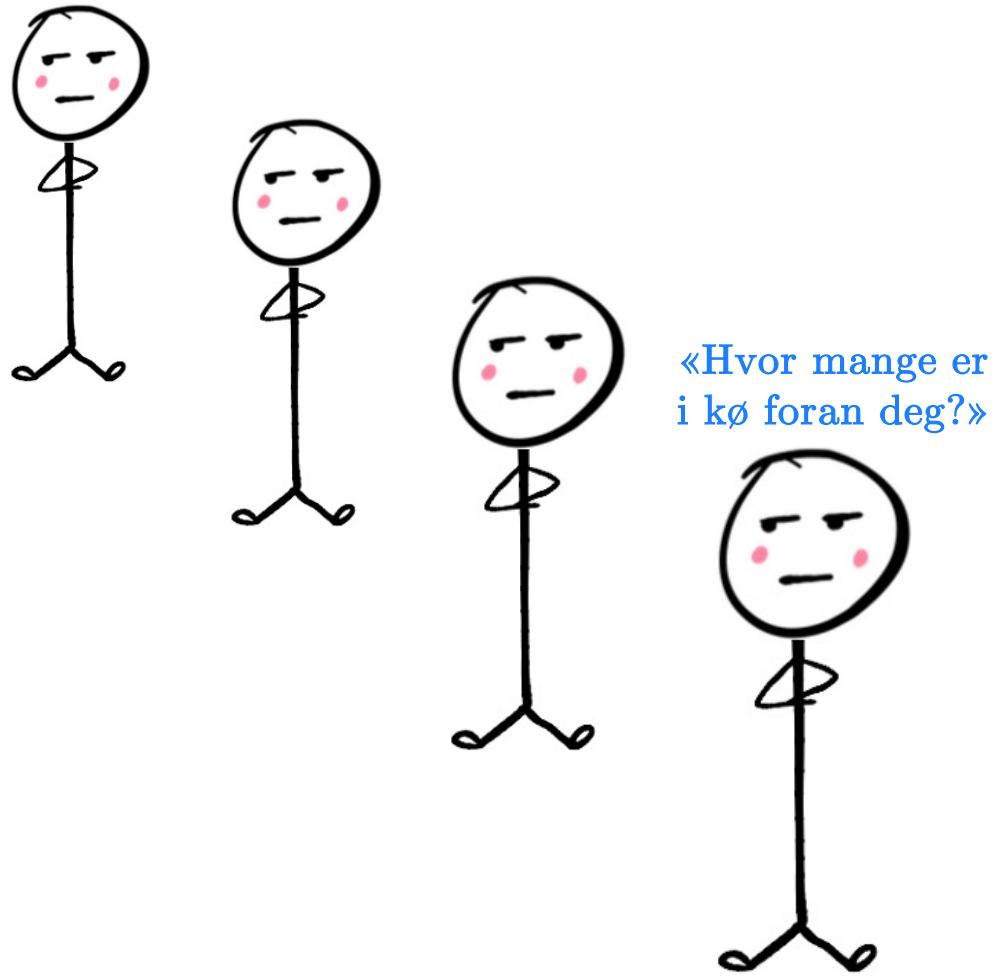


Rekursjon

- 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**

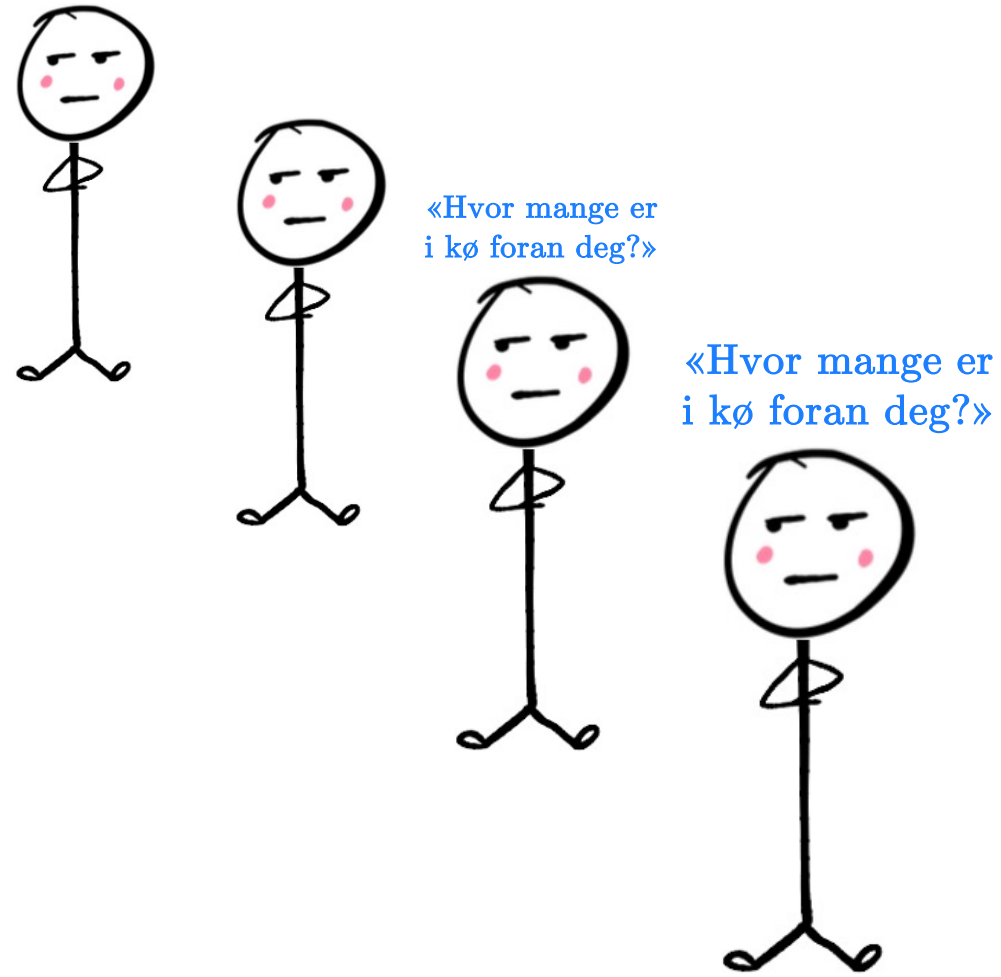


Rekursjon

- 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**

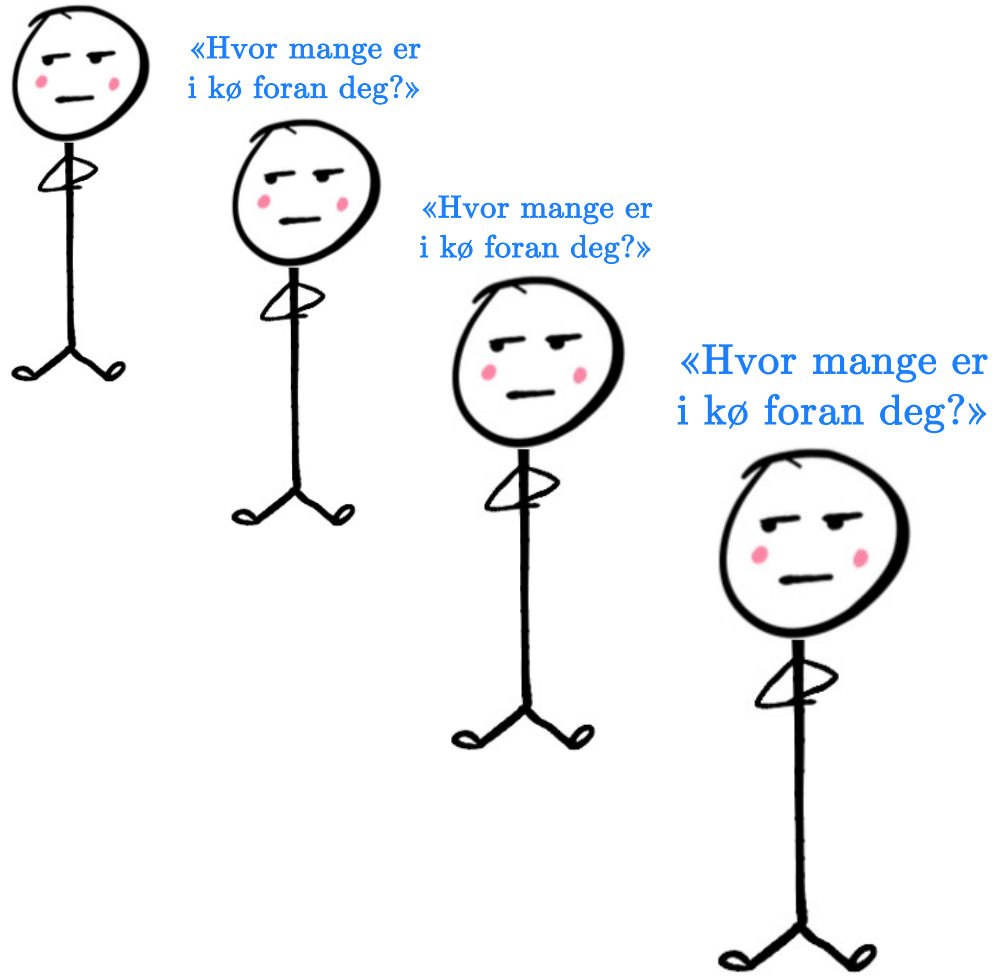


Rekursjon

- 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**

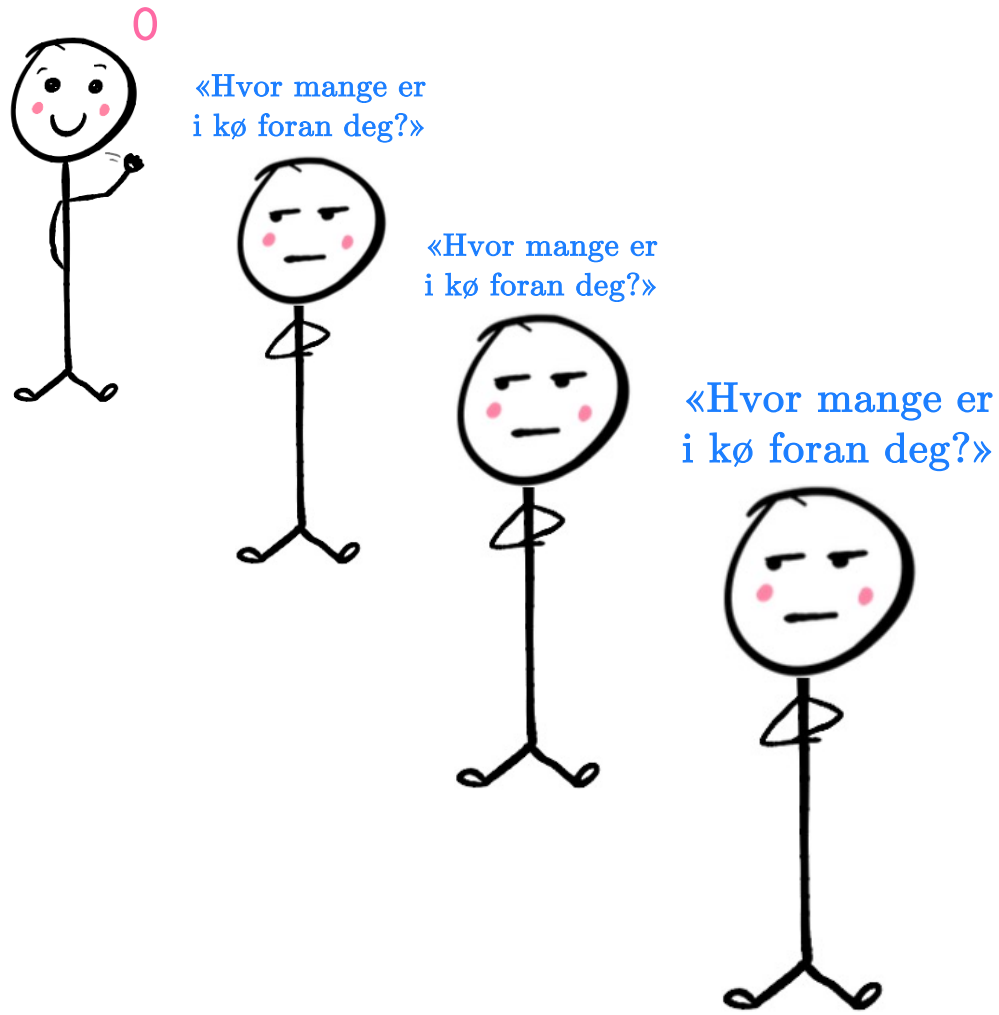


Rekursjon

- 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**

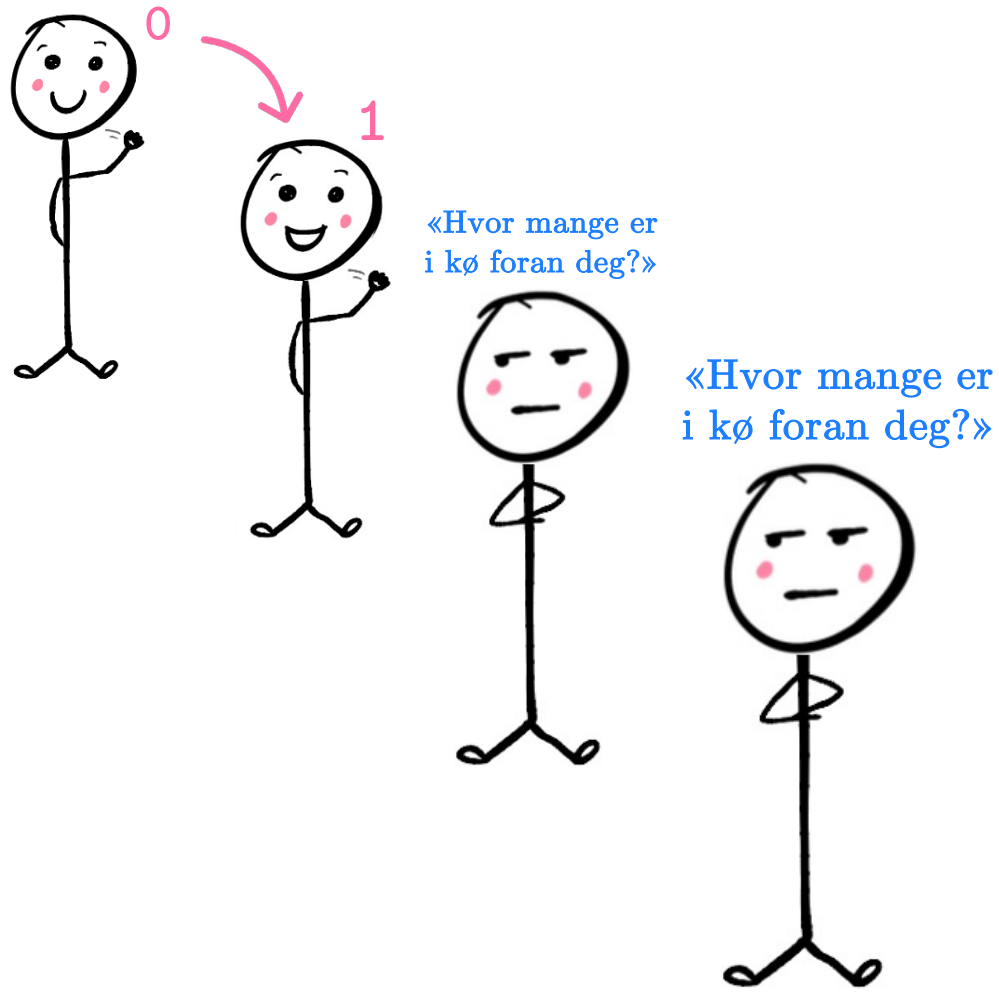


Rekursjon

- 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**

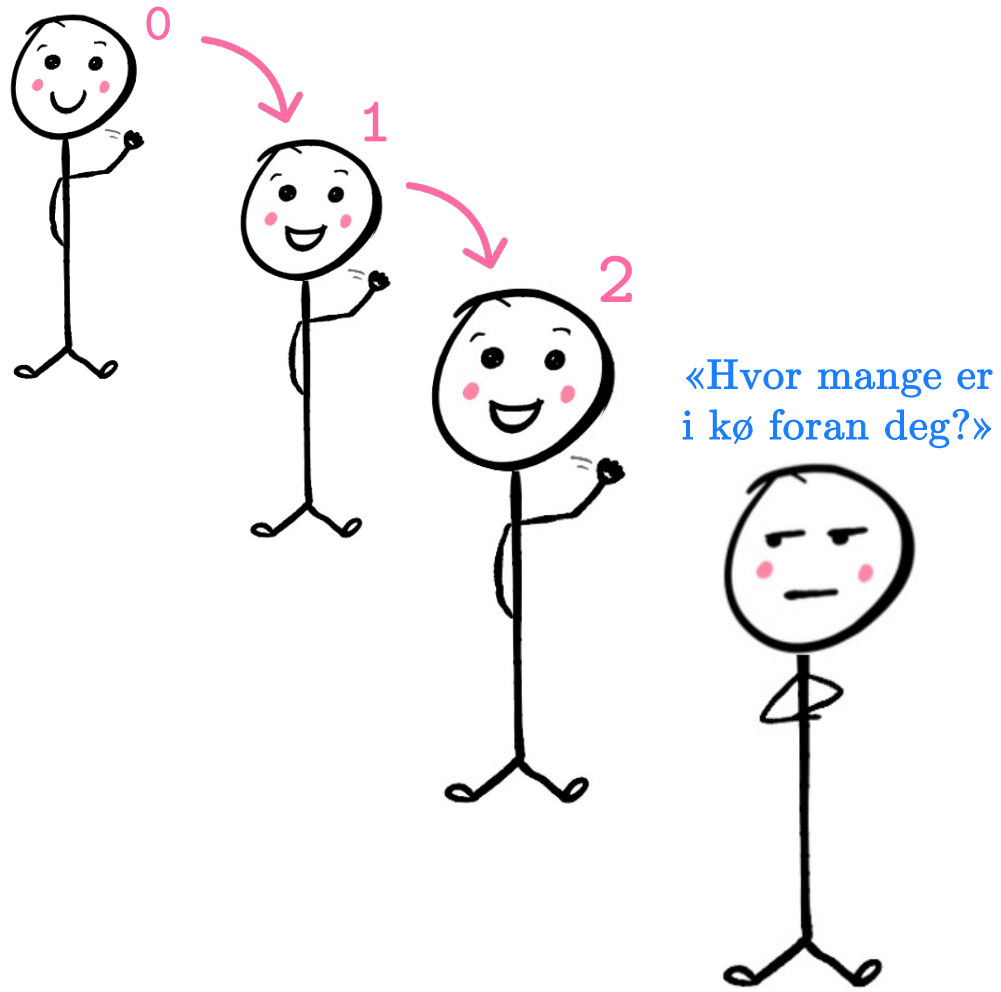


Rekursjon

- 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**

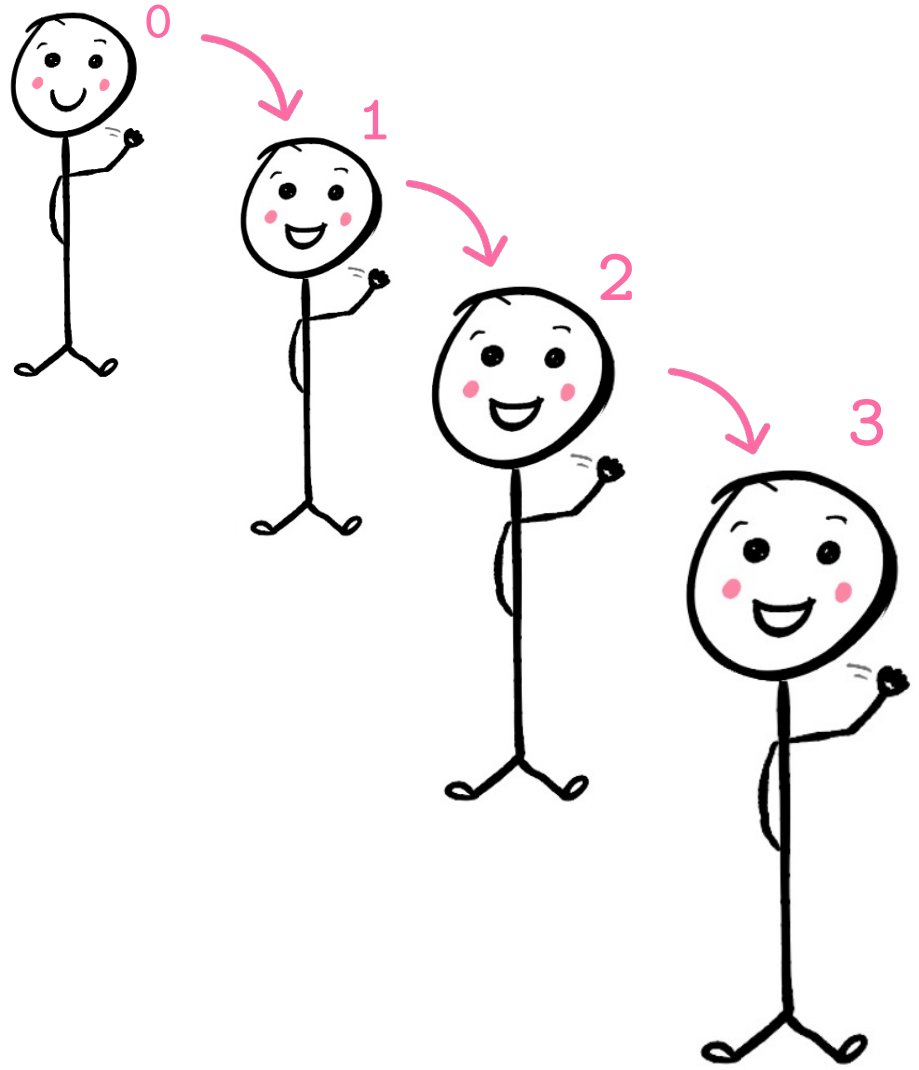


Rekursjon

- 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)
```

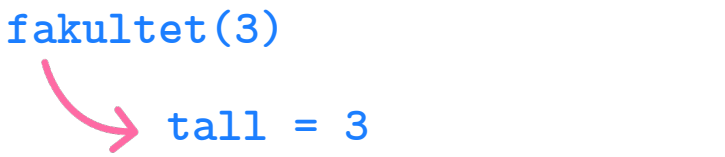
`fakultet(3)`

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)
```

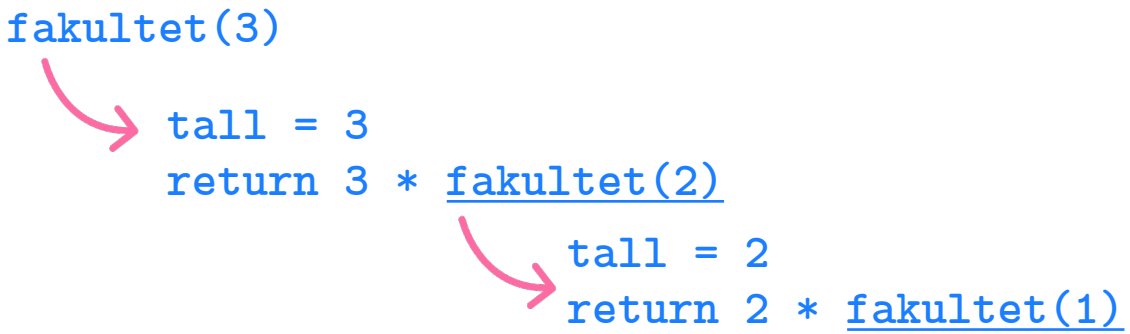


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

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

```
fakultet(3)
```

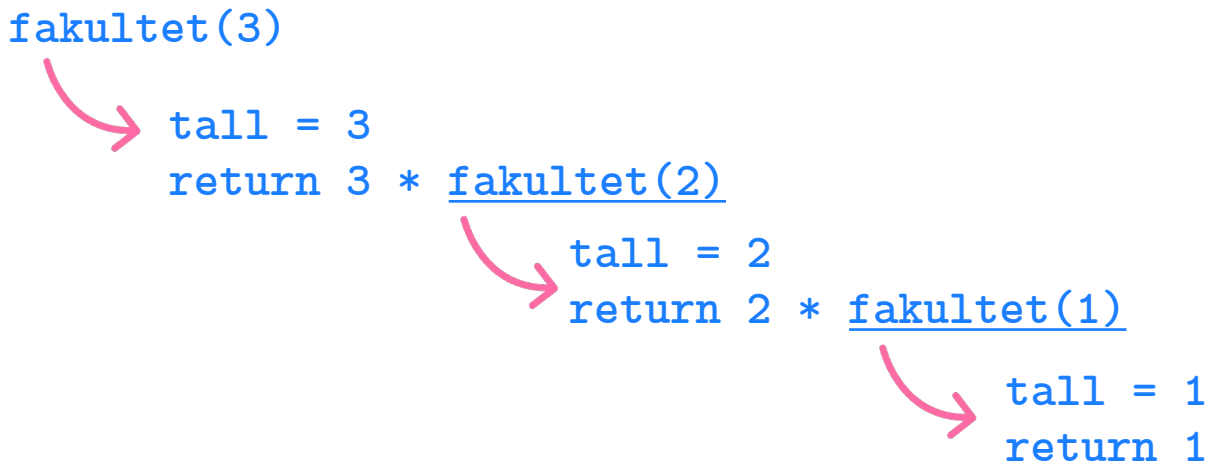


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

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

```java
fakultet(3)
```

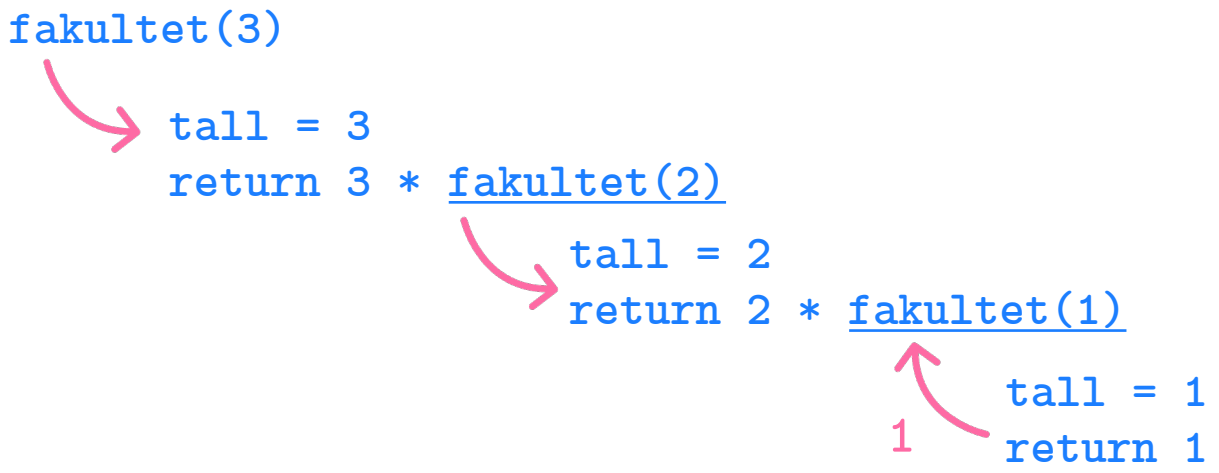


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

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

```java
fakultet(3)
```

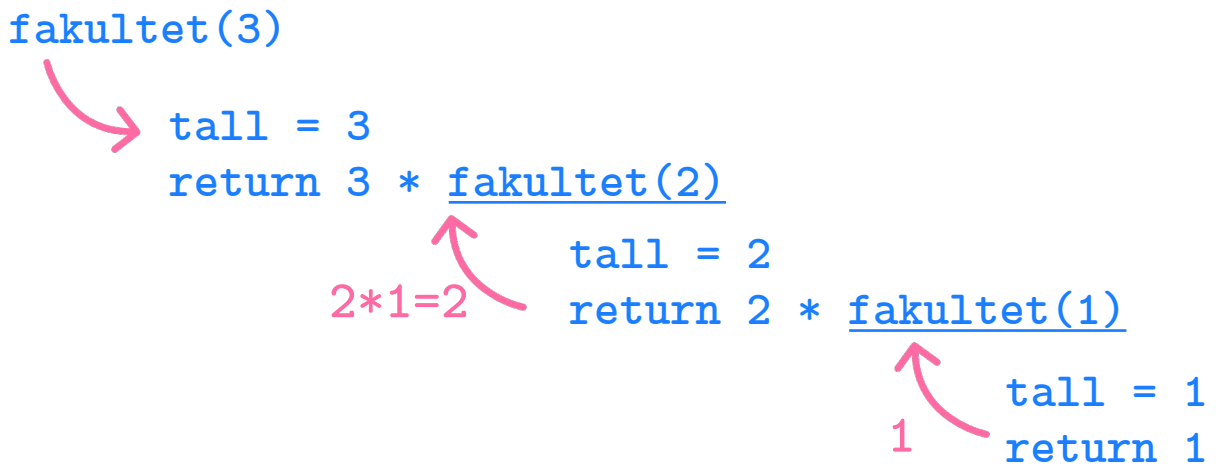


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

Rekursjon - eksempel

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

```java
fakultet(3)
```

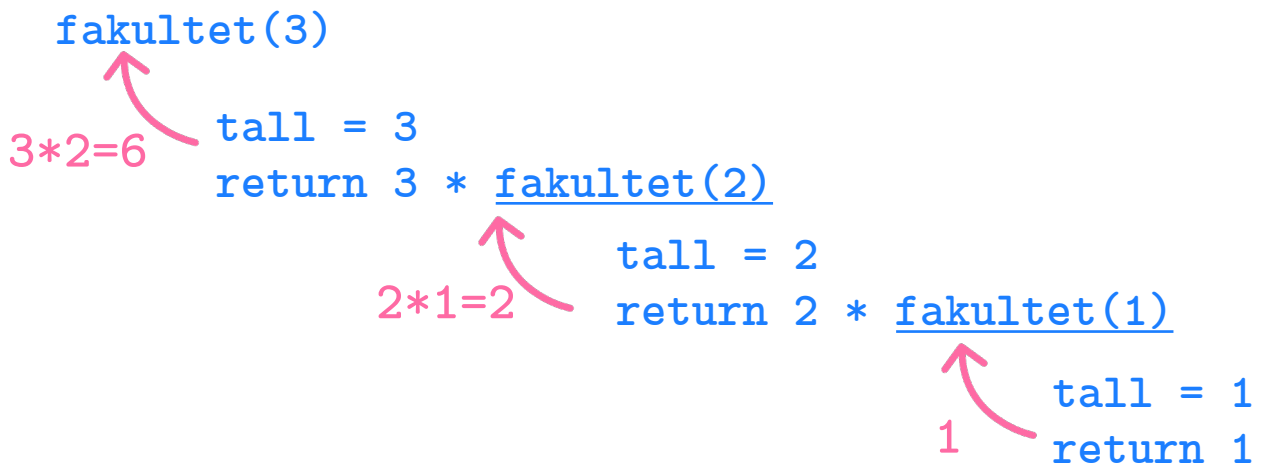


Inspirert av gruppelærer Felicia Norén sine foiler for uke 7

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

På eksamen

Eksamen 2019

2(d) **Eldste kjente opphav**

Skriv den rekursive metoden finnEldsteKjenteOpphav. En hunds opphav er dens mor og dens far samt deres mødre og fedre osv. Om både moren og faren til en hund er ukjent, er det eldste kjente opphavet hunden selv.

Maks poeng: 20

Eksamen 2023

4d) 8 poeng

⇨ Skriv en rekursiv løsning for å sjekke at alle objektene i et tog har samme sporvidde, inkludert eventuelle endringer eller tillegg du vil gjøre i svar på tidligere oppgaver.

Tråder

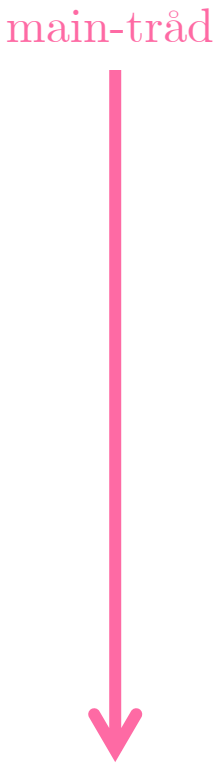
Parallelle programmer

Tråder

- **Parallelle programmer**
  - Flere programmer kjører parallelt/samtidig
  - I motsetning til **sekvensielle** programmer
    - Programmet gjør én ting om gangen

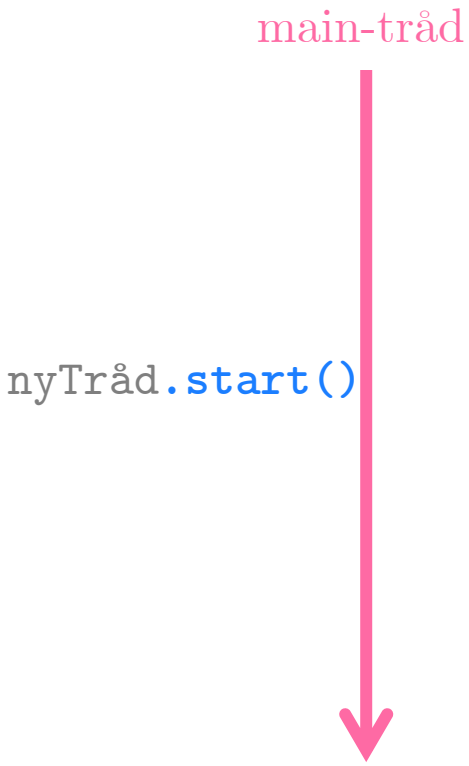
Tråder

- **Parallelle programmer**
  - Flere programmer kjører parallelt/samtidig
  - I motsetning til **sekvensielle** programmer
    - Programmet gjør én ting om gangen



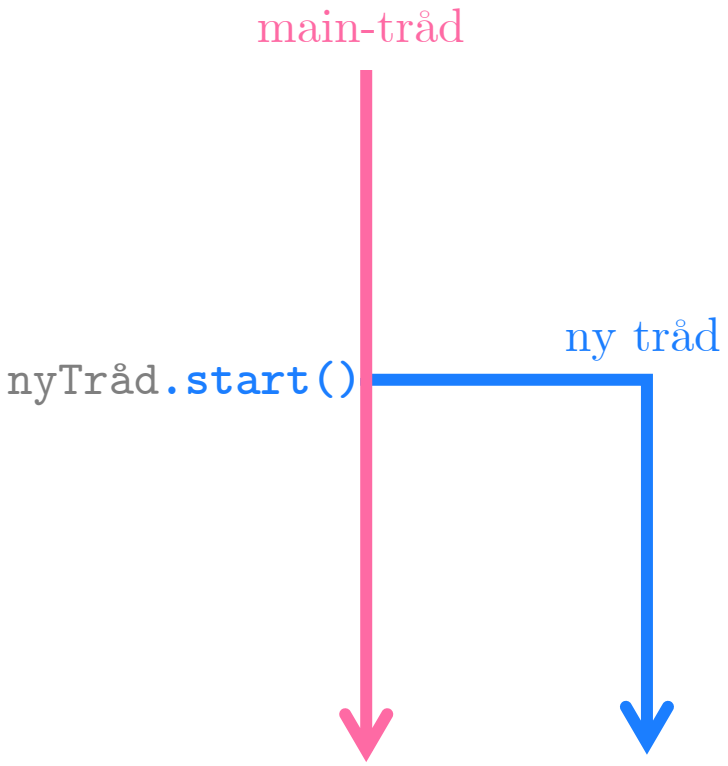
Tråder

- **Parallelle programmer**
  - Flere programmer kjører parallelt/samtidig
  - I motsetning til **sekvensielle** programmer
    - Programmet gjør én ting om gangen



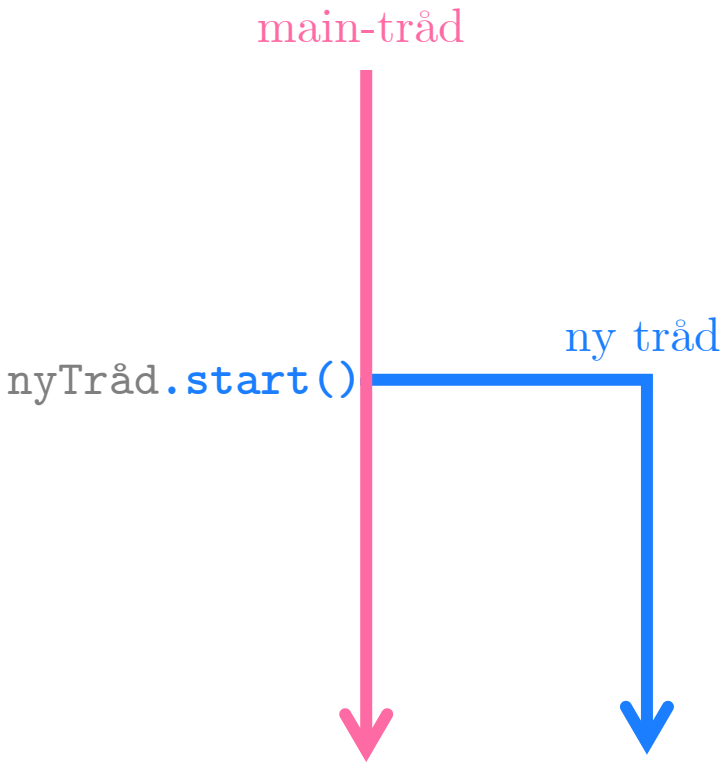
Tråder

- **Parallelle programmer**
  - Flere programmer kjører parallelt/samtidig
  - I motsetning til **sekvensielle** programmer
    - Programmet gjør én ting om gangen



Tråder

- **Parallelle programmer**
  - Flere programmer kjører parallelt/samtidig
  - I motsetning til **sekvensielle** programmer
    - Programmet gjør én ting om gangen
- I IN1010:
  - `class Thread`
  - `interface Runnable`



Tråder

1. Definer en klasse som implementerer `Runnable`
   - Skriv `run()`-metoden
     = hva skjer når tråden kjører

```
public class Oppgave implements Runnable {



}
```

```

```

Tråder

1. Definer en klasse som implementerer `Runnable`
   - Skriv `run()`-metoden
     = hva skjer når tråden kjører

```
public class Oppgave implements Runnable {
    public void run() {
        for (int i = 1; i < 10; i++) {
            System.out.printf("Hei for %d. gang!\n",i);
        }
    }
}
```

Tråder

1. Definer en klasse som implementerer `Runnable`
   - Skriv `run()`-metoden
     = hva skjer når tråden kjører
2. Lag et objekt av klassen

```java
public class Oppgave implements Runnable {
    public void run() {
        for (int i = 1; i < 10; i++) {
            System.out.printf("Hei for %d. gang!\n",i);
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Runnable oppgave = new Oppgave();



}
```

Tråder

1. Definer en klasse som implementerer `Runnable`
   - Skriv `run()`-metoden
     = hva skjer når tråden kjører
2. Lag et objekt av klassen
3. Send objektet som parameter til et objekt av `Thread`-klassen

```
public class Oppgave implements Runnable {
    public void run() {
        for (int i = 1; i < 10; i++) {
            System.out.printf("Hei for %d. gang!\n",i);
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Runnable oppgave = new Oppgave();
        Thread traad1 = new Thread(oppgave);
        Thread traad2 = new Thread(oppgave);

}
```

Tråder

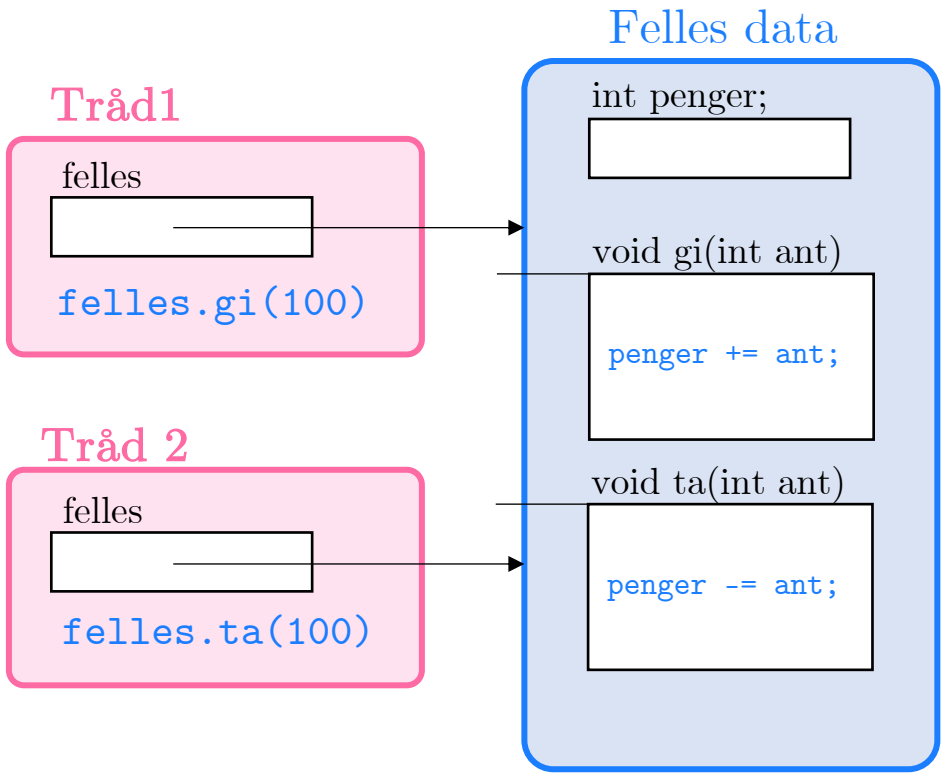
1. Definer en klasse som implementerer `Runnable`
   - Skriv `run()`-metoden
     = hva skjer når tråden kjører
2. Lag et objekt av klassen
3. Send objektet som parameter til et objekt av `Thread`-klassen
4. Kall `start()` på `Thread`-objektet

```java
public class Oppgave implements Runnable {
    public void run() {
        for (int i = 1; i < 10; i++) {
            System.out.printf("Hei for %d. gang!\n",i);
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Runnable oppgave = new Oppgave();
        Thread traad1 = new Thread(oppgave);
        Thread traad2 = new Thread(oppgave);
        traad1.start(); traad2.start();}
}
```

Tråder – Datadeling

- Flere tråder kan utføre endringer på felles data
  - Uten synkronisering: kan gi feil
  - Løsning: **låser**

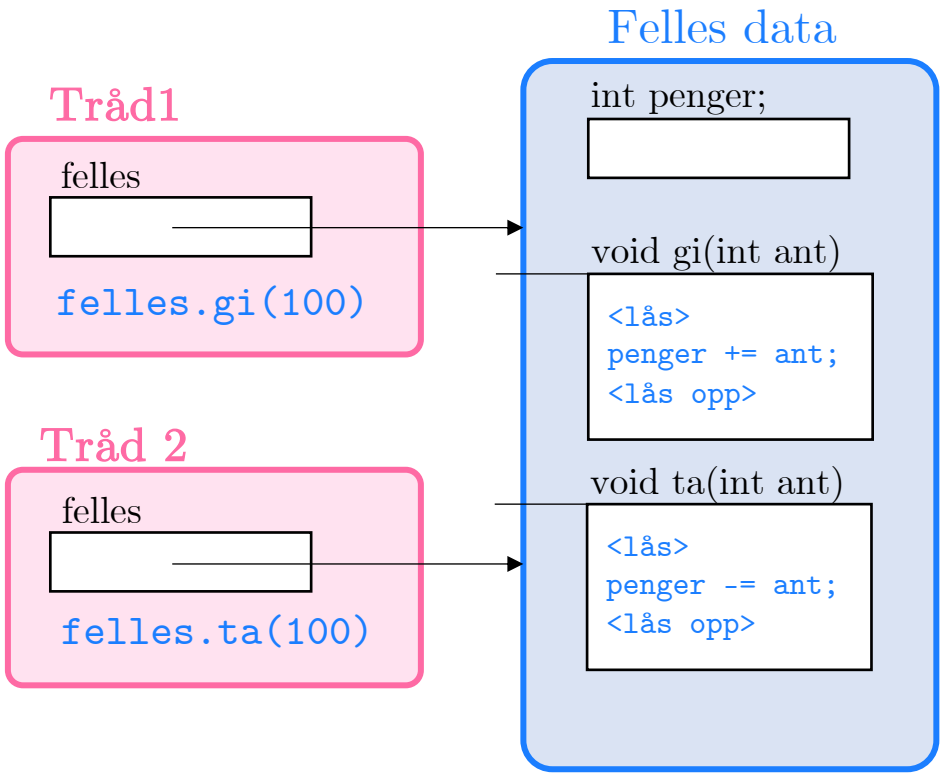


Eksempel fra forelesningen Tråder I

Tråder – Datadeling

- Flere tråder kan utføre endringer på felles data
  - Uten synkronisering: kan gi feil
  - Løsning: **låser**

- **Låser:** vi lar kun én tråd aksessere et delt objekt til en gitt tid
  - **Kritisk region**: kodebit som utføres ferdig før en annen tråd får lov
  - Andre tråder må vente på sin tur

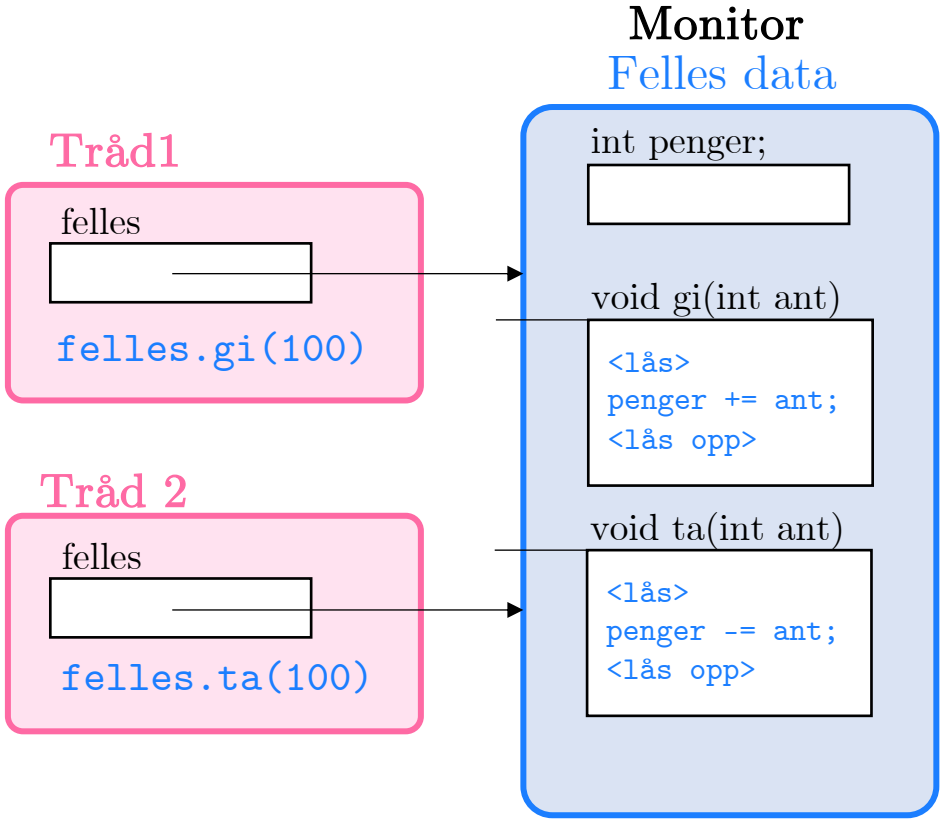


Eksempel fra forelesningen Tråder I

Tråder – Datadeling

- Flere tråder kan utføre endringer på felles data
  - Uten synkronisering: kan gi feil
  - Løsning: **låser**

- **Låser:** vi lar kun én tråd aksessere et delt objekt til en gitt tid
  - **Kritisk region**: kodebit som utføres ferdig før en annen tråd får lov
  - Andre tråder må vente på sin tur



Eksempel fra forelesningen Tråder I

Tråder – Låsing

```
public class Monitor {
    private int teller = 0;
    private final Lock lock = new ReentrantLock();

    public void økTeller() {
        lock.lock();
        try {
            teller++;
        } finally {
            lock.unlock();
        }
    }
}
```

Tråder – Låsing

```
public class Monitor {
    private int teller = 0;
    private final Lock lock = new ReentrantLock();

    public void økTeller() {
        lock.lock();
        try {
            teller++;
        } finally {
            lock.unlock();
        }
    }
}
```

// Opprett en lås

Tråder – Låsing

```
public class Monitor {
    private int teller = 0;
    private final Lock lock = new ReentrantLock();

    public void økTeller() {
        lock.lock();
        try {
            teller++;
        } finally {
            lock.unlock();
        }
    }
}
```

```
// Opprett lås


// Lås
// Forsøk å utføre noe
```

Tråder – Låsing

```
public class Monitor {
    private int teller = 0;
    private final Lock lock = new ReentrantLock();

    public void økTeller() {
        lock.lock();
        try {
            teller++;
        } finally {
            lock.unlock();
        }
    }
}
```

```
// Opprett lås


// Lås
// Forsøk å utføre noe
// Lås alltid opp igjen
```

Tråder – Conditions

- **Aktiv** vs. **passiv** venting

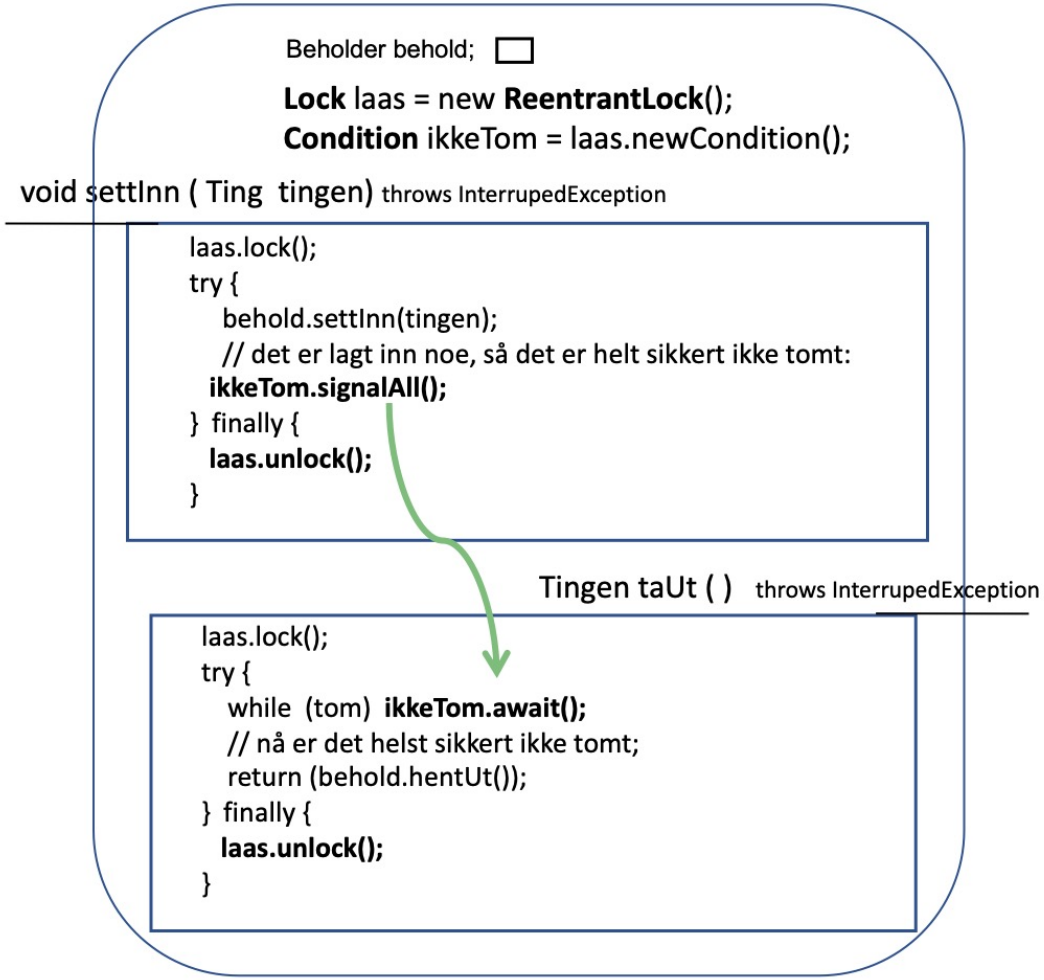
Tråder – Conditions

- **Aktiv** vs. **passiv** venting
- Passiv venting med **signalering**

Tråder – Conditions

- **Aktiv** vs. **passiv** venting
- Passiv venting med **signalering**
  - **Conditions** opprettes på låsen med `newCondition()`
    - `await()` – får tråden til å vente på signal (eller interruption)
    - `signal()` – vekker én tråd
    - `signalAll()`– vekker alle trådene

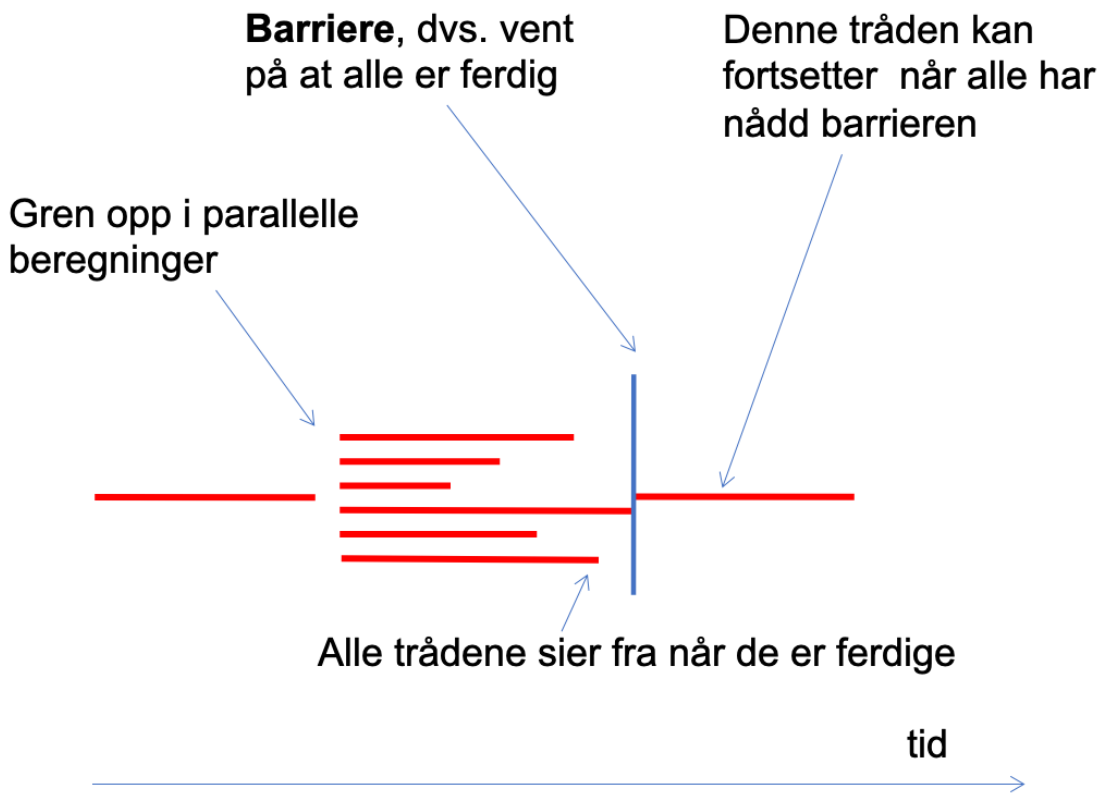
Tråder – Conditions



Fra forelesningen Tråder I

Tråder - Barrierer

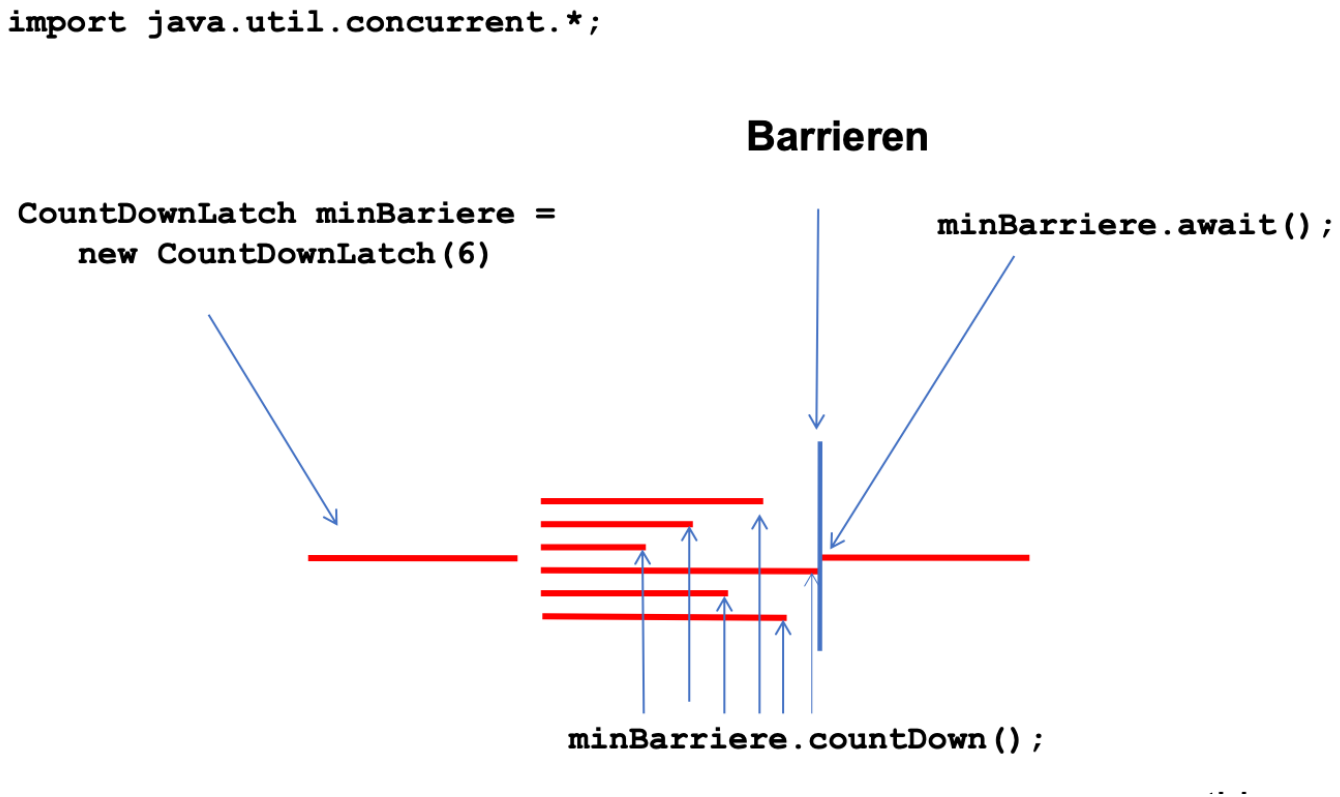
- Barrierer – annen måte å vente på



Fra forelesningen Tråder II (2023)

Tråder - Barrierer

- Barrierer – annen måte å vente på
  - `CountDownLatch` – venter på at et gitt antall tråder har kalt `countDown()`



Fra forelesningen Tråder II (2023)

På eksamen

Eksamen 2023

Oppgave 5. Tråder (20 poeng)

I oppgave 5 skal du skrive én trådklasse som heter **Leter** og én som heter **Resultat**. Begge disse skal implementere Runnable. Du skal også skrive en klasse **Monitor** som skal være en monitor.

5 a) 5 poeng

Leter-trådene skal lete gjennom hvert sitt tog. De skal lete etter alle skinnegående objekter der Id-en begynner med en bestemt tekststreng. Hver gang en Leter-tråd har funnet et skinnegående objekt som passer, skal den legge en referanse til dette inn i monitoren ved å kalle metoden **leggTil** i monitoren. Når en Leter-tråd har lett gjennom hele sitt tog, skal den si fra til monitoren om dette ved å kalle metoden **ferdigLeting** i monitoren og så terminere.

Som parameter til Leter-klassens konstruktør skal du ha en referanse til toget tråden skal lete i, en referanse til monitoren og en referanse til String-objektet det skal letes etter.

Hint: Metoden startsWith(String s) i String-klassen returnerer true hvis String-objektet den kalles på, begynner med s.

⇨ Skriv trådklassen Leter

5 b) 3 poeng

Det skal bare være én instans av tråden Resultat. Resultat-tråden skal hente Skinnegående-objekter fra monitoren ved å kalle metoden **hentNeste** i monitoren og skrive ut Id-en i terminalvinduet ved hjelp av System.out.println. Hvis hentNeste returnerer null er det ikke flere Skinnegående- objekter igjen å skrive ut, og tråden kan terminere. Resultat-tråden skal kunne startes opp samtidig med Leter-trådene og således kunne utføres i parallell med Leter-trådene.

Resultat-klassens konstruktør skal bare ha en referanse til monitoren som parameter

⇨ Skriv trådklassen Resultat

5 c) 12 poeng

Monitoren skal inneholde de tre metodene leggTil, ferdigLeting og hentNeste som beskrevet over. Som parameter til monitorens konstruktør skal du bare ha hvor mange Leter-tråder som finnes.

⇨ Skriv Monitor-klassen

(Du skal altså ikke skrive noe fullstendig program som starter opp noen tråder – du skal bare skrive disse tre klassene med metoder og variabler.)

På eksamen

Eksamen 2018

10

Søk etter terrorister

Det er et krav at flyreservasjonsprogrammer må kunne brukes til hjelpe politiet med å finne mulige terrorister.

Du skal skrive to trådklasser samt en monitorklasse og et hovedprogram til dette bruk. Den første trådklassen skal hete **Forstelinjevokter**. Alle Forstelinjevokter-trådene skal samtidig søke gjennom hver sin flygning. Når en Forstelinjevokter finner en mistenkelig person (hva det vil si skal vi definere senere), skal tråden legge den mistenkelige personen inn i en monitor av klassen **MistenkeligePersoner**. Objekter av den andre trådklassen, kalt **Andrelinjevokter**, skal hente personer ut av denne monitoren og kalle en statisk metode i hovedprogramklassen din med navn **mistenkelig()** der disse personene kommer til å bli nøye undersøkt av politiet. (Du skal ikke skrive denne metoden.)

Hovedprogrammet skal altså brukes når politiet får tips om en sammenhengende del av navnet til en terrorist som skal ut og fly. Denne navnedelen gis som parameter til programmet og blir dermed tilgjengelig i parameter **args[0]** i hovedprogrammet når det er deklarert som

```
public static void main(String[] args) { ... }
```

La programmet gå gjennom alle flygningene og opprette og starte en Forstelinjevokter-tråd for hver flygning. For enkelthets skyld kan du anta at programmet kan få tak i en iterator over alle flygningene til flyselskapet ved å kalle den statiske metoden alleFlygninger() i hovedprogramklassen (du skal ikke programmere denne metoden).

Den sammenhengende delen av et navn som politiet får tips om, kalles et *mønster*. Jo tidligere i navnet til den mistenkte personen dette mønsteret forekommer, jo mer sannsynlig er det at personen er en terrorist. Når en Forstelinjevokter-tråd legger en mistenkt person inn i monitoren, skal avstanden fra starten av navnet til starten av mønsteret brukes som prioritet. F.eks. vil STEIN fra oppgave 1 få prioritet 2 når mønsteret er EI. Som i oppgave 2 betyr et lite tall høy prioritet.

Programmet ditt skal også starte 100 Andrelinjevokter-tråder som tar mistenkte ut av monitoren, og da skal de mistenkte med høyest prioritet tas ut først. Om monitoren er tom, skal trådene vente på at en ny person blir lagt inn.

Det er ikke nødvendig å tenke på at trådene skal terminere.

Maks poeng: 22

GUI

Graphical User Interface

GUI

- Grafisk brukergrensesnitt
  - Visuelt og mer intuitiv bruk



GUI

- Grafisk brukergrensesnitt
  - Visuelt og mer intuitiv bruk
- **Hendelsesorientert** programmering



GUI

- Grafisk brukergrensesnitt
  - Visuelt og mer intuitiv bruk
- **Hendelsesorientert** programmering
- Eksempel på parallellisering og tråder



GUI

- Grafisk brukergrensesnitt
  - Visuelt og mer intuitiv bruk
- **Hendelsesorientert** programmering
- Eksempel på parallellisering og tråder
- I IN1010: **AWT** og **Swing**



GUI

- Sentrale GUI-elementer
  - JFrame: hovedvinduet
  - JPanel: tegneflaten – en del av vinduet vi fyller med innhold
    - Her kan vi plassere ulike «bokser»
  - JLabel: tekst (eller bilde)
  - JButton: klikkbar knapp

GUI

- Sentrale GUI-elementer
  - JFrame: hovedvinduet
  - JPanel: tegneflaten – en del av vinduet vi fyller med innhold
    - Her kan vi plassere ulike «bokser»
  - JLabel: tekst (eller bilde)
  - JButton: klikkbar knapp
- Mulighet for ulike variasjoner
  - Farge, plassering, font, mm.
  - Se GUI 2-forelesningen

GUI - Eksempel

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```



GUI - Eksempel

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```

// Importer

GUI - Eksempel

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```

// Importer

// Angi standard Swing-utseende

GUI - Eksempel

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```

```
// Importer

// Angi standard
  Swing-utseende

// Deklarer vinduet vårt
// Stopp programmet når
  vinduet lukkes
```

GUI - Eksempel

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```

```
// Importer

// Angi standard
  Swing-utseende

// Deklarer vinduet vårt
// Stopp programmet når
  vinduet lukkes
// Opprett tegneflaten
og sett inn i vinduet
```

GUI - Eksempel

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```

```
// Importer


// Angi standard
  Swing-utseende


// Deklarer vinduet vårt
// Stopp programmet når
  vinduet lukkes
// Opprett tegneflaten
og sett inn i vinduet

// Pakk innholdet sammen
// Plasser vindu midt på
skjermen
// Gjør vinduet synlig
```

GUI – Eksempel med tekst

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        JLabel hilsen = new JLabel("God tirsdag!");
        panel.add(hilsen);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```



GUI – Eksempel med tekst

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
            System.exit(1);
        }

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        JLabel hilsen = new JLabel("God tirsdag!");
        panel.add(hilsen);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```



GUI - Knapper

- Trykknapper – vanlig form for interaksjon
  - Jbutton: klikkbar knapp

GUI - Knapper

- Trykknapper – vanlig form for interaksjon
  - **Jbutton**: klikkbar knapp
- Fremgangsmåte:
  1. Opprett en **JButton**

GUI - Knapper

- Trykknapper – vanlig form for interaksjon
  - Jbutton: klikkbar knapp
- Fremgangsmåte:
  1. Opprett en **JButton**
  2. Sett knappen på en **Jpanel**

GUI - Knapper

- Trykknapper – vanlig form for interaksjon
  - Jbutton: klikkbar knapp
- Fremgangsmåte:
  1. Opprett en **JButton**
  2. Sett knappen på en **Jpanel**
  3. Definer hva som utføres ved et trykk: ActionListener

GUI - Knapper

- Trykknapper – vanlig form for interaksjon
  - Jbutton: klikkbar knapp
- Fremgangsmåte:
  1. Opprett en **JButton**
  2. Sett knappen på en **Jpanel**
  3. Definer hva som utføres ved et trykk: ActionListener
  4. Koble ActionListener til knappen med addActionListener

GUI – Eksempel med knapp

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
// import

class Mini {
    public static void main (String[] args) {
        // (trycatch)

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        JButton exitKnapp = new JButton("Avslutt");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0); }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```



GUI – Eksempel med knapp

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
// import

class Mini {
    public static void main (String[] args) {
        // (trycatch)

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        JButton exitKnapp = new JButton("Avslutt");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0); }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```




```
// Opprett en knapp
```

GUI – Eksempel med knapp

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
// import

class Mini {
    public static void main (String[] args) {
        // (trycatch)

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        JButton exitKnapp = new JButton("Avslutt");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0); }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```




```
// Opprett en knapp
// Hva skal skje når
action gjøres?
```

GUI – Eksempel med knapp

Modifisert eksempel fra GUI 1-forelesning (Dag Langmyhr)

```
// import

class Mini {
    public static void main (String[] args) {
        // (trycatch)

        JFrame vindu = new JFrame("Mitt program");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        vindu.add(panel);

        JButton exitKnapp = new JButton("Avslutt");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0); }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
}
```




```
// Opprett en knapp
// Hva skal skje når
action gjøres?

// Legg til Action-
Listener til knapp
//Legg til knapp på
panel
```

På eksamen

Utsatt eksamen 2021

Grafisk brukergrensesnitt - GUI

I oppgave 1i skal du skrive et GUI som samhandler med beholderen med reiser. Her trenger du tre Swing-klasser som ikke er gjennomgått på forelesningene, men som er beskrevet i læreboken og selvfølgelig i Java-dokumentasjonen.

JTextField opprettes slik: JTextField t = new JTextField("initiell tekst");
Den fungerer omtrent som JLabel men lar brukeren redigere teksten. Programmet kan få tak i teksten med metoden *getText()*. (Hint: Denne klassen er velegnet om brukeren skal oppgi et tall.)

JTextArea opprettes slik: JTextArea a = new JTextArea(10,50);
(Parametrene er antall rader og kolonner det settes av plass til.)
Den brukes til å skrive ut flere linjer tekst på skjermen. De to mest aktuelle metodene er:

* *setText(null)* vil blanke området.
* *append("xxx" + "\n")* vil skrive ut tekst i området; husk linjeskift.

JScrollPane opprettes slik: JScrollPane s = new JScrollPane(a) der a er en referanse til et JTextArea. Den brukes til å gi tekstvinduet muligheter til å kunne rulles.

GUI-vinduet skal inneholde et område (f.eks. et JTextArea) der reiser kan vises frem. I tillegg skal vinduet inneholde knapper og felt som begrenser hvilke reiser som vises.
Brukeren skal oppgi en nedre grense og en øvre grense for hva reisene som vises skal koste (med mva). I tillegg skal brukeren kunne velge å bare få vist all-inclusive-reiser eller bare-fly-reiser. Du må sørge for at ikke «all inclusive» og «bare fly» kan velges samtidig.

Oppgave 1i vekt 17
Skriv klassen ReisebyraVindu som inneholder GUI-delen av programmet som beskrevet over. Parameteren til konstruktøren av ReisebyraVindu er en referanse til HashListen som inneholder alle reisene. Når brukeren trykker på en knapp «Finn» skal programmet lage et filter basert på feltene som er satt i GUI-et. Deretter skal det gå gjennom alle reisene i beholderen men bare vise frem i vinduet de reisene som passerer filteret.
Hint: Opprett vinduet og alle komponentene og koble komponentene sammen med hendelseshåndteringsobjektene i konstruktøren til ReisebyraVindu.
Slutt oppgave 1i

På eksamen

Utsatt eksamen 2022

GUI
I tillegg til å legge inn data via appen, skal det være mulig å bruke et GUI-vindu til å registrere støy.

Du skal skrive en klasse GUI med en metode void lagStøyVindu() som kan brukes til å opprette et vindu der en bruker kan registrere støymålinger. Konstruktøren til GUI-klassen skal ta en referanse til By-objektet der målingene gjøres, som parameter.

Når metoden lagStøyVindu() kalles, skal det opprettes et vindu der brukeren kan taste inn navnet på et sted, et tidspunkt og en støyverdi (i dB).
Når brukeren har tastet inn disse dataene i tre forskjellige felt og trykket på «OK»-knappen, skal programmet lage et Måling-objekt og sette dette inn i datastrukturen i byen.

Vinduet skal kunne ta imot flere målinger. Brukeren kan derfor taste inn nye data og trykke på «OK»-knappen på nytt for å registrere en ny måling.

Oppgave 14. Vekt 12
Skriv klassen GUI med metoden void lagStøyVindu() slik som beskrevet over. Gjør eventuelt andre forandringer i programmet ditt slik at GUI-et virker som beskrevet over.

Hint: For å lage et GUI-felt der brukeren kan skrive, kan vi bruke klassen JTextField. Den fungerer som JButton men har i tillegg metoden String getText() for å få tak i det som brukeren har skrevet.

På eksamen

Eksamen 2022 (1)

GUI-dialog

Nå skal du programmere `GUIDialog` som implementerer `Dialog` (se oppgave 3) og som gir brukeren mulighet til å kommunisere med programmet i et GUI-vindu som kan se slik ut:



Første gang `svarJaEllerNei()` kalles, skal metoden åpne et slikt GUI-vindu. Spørsmålet er parameter til metoden. Når brukeren klikker på JA- eller NEI-knappen, skal metoden returnere med henholdsvis **`true`** eller **`false`**.

Dette vinduet skal ikke lukkes så lenge programmet kjøres. Det betyr at neste gang `svarJaEllerNei()` kalles, skal metoden bruke samme vindu og bare endre spørsmålet.

Oppgave 11. 18 poeng
Skriv klassen GUIDialog med metoden `svarJaEllerNei()`.

Hint. Når metoden `svarJaEllerNei()` skal vente til brukeren klikker på JA eller NEI, må den sove. Dette kan gjøres med `Thread.sleep()` med en ganske stor verdi, for eksempel

```
try {
    Thread.sleep(1000000);
} catch (InterruptedException e) {}
```

Når `ActionListener`-objektet utføres (fordi brukeren har klikket på JA eller NEI), må den vekke hovedtråden. Dette kan gjøres med

```
hovedtrad.interrupt();
```

`GUIDialog`-objektet må ha en referanse til hovedtråden; dette kan den få ved å utføre

```
Thread hovedtrad = Thread.currentThread();
```

Oppsummering

Arv og subklasser	**Beholdere**	**Rekursjon**	**Tråder**	**GUI**
- Subklasser - Interface - Klassehierarki-tegning - Arv vs. komposisjon - Casting	- Array, ArrayList, HashMap mm. - Lenkelister - Generiske klasser - Stabel, kø og prioritetskø	- Rekursive metoder - Basistilfeller - Rekursive kall	- Runnable og Threads - Monitorer - Låser - Conditions - Barrierer	- Hendelses-orientert - AWT+Swing - JFrame, JPanel, JLabel, JButton

Oppsummering

- Andre relevante temaer:
  - Exceptions
  - Scanner
    - Fillesing/-skriving
    - Input
  - Datastrukturtegning
  - Iterator
  - Comparable
  - Overloading
  - Design og testbarhet