Velkommen!

•••

Johanna
johannph på mattermost
johannph@uio.no på mail!

Kort: Praktisk informasjon

- Undervisningstilbud
  - https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/undervisningstilbud/
  - Jeg har konkrete spørsmål/problemer med min kode -> Labtime!
  - Jeg vil ha mer liveprogrammering -> Plenumstime!
  - Jeg vil jobbe med andre (og kanskje en kjapp recap av forelesning) -> Gruppetime!
  - Jeg vil ha en recap av de vanskeligste konseptene fra forelesning -> Repetisjonsgruppe!
- Skriv alt dere lurer på i chatten enten til everybody eller bare til meg 😊

Oblig 3

Kort om oblig3

- Testprogrammene

  Jeg tror de testprogrammene bare er en ulempe hvis dere synes lenkeliste er vanskelig å forstå. Risikerer at dere kanskje ikke har skjønt det helt, men bare prøvd til testprogrammet kjører.

- Tips

  Lag en lenkeliste før eksamen der dere lager deres eget testprogram
  Testprogrammet må teste edge-cases: det første elementet, det siste elementet og feilmeldinger.

Tips oblig 4

Ikke lag en fil som heter Iterable med et interface Iterable! Iterable finnes i java biblio! Skriver man sitt eget interface Iterable overskriver man det i java sitt biblio!

Ikke lag en fil som heter Iterator med et interface Iterator! Iterator finnes i java biblio! Skriver man sitt eget interface Iterator overskriver man det i java sitt biblio!

Oblig 5

Oblig 5 legges ut i morgen, den har frist 5. april kl 23.59. Det er rett etter (eller er det i?) påsken!

I påsken er det ikke noe opplegg eller hjelp å få.

Så gjør den asap før påske!!!

Repetisjon forrige uke

Forrige gang

Casting

Exceptions

Hva printes? Send meg direktemelding i chatten!

```java
class Dyr{
    protected String navn;

    public Dyr(String navn){
        this.navn = navn;
    }
    public String hentNavn(){
        return navn;
    }
    public String hentLyd(){
        return "hello";
    }
}
```

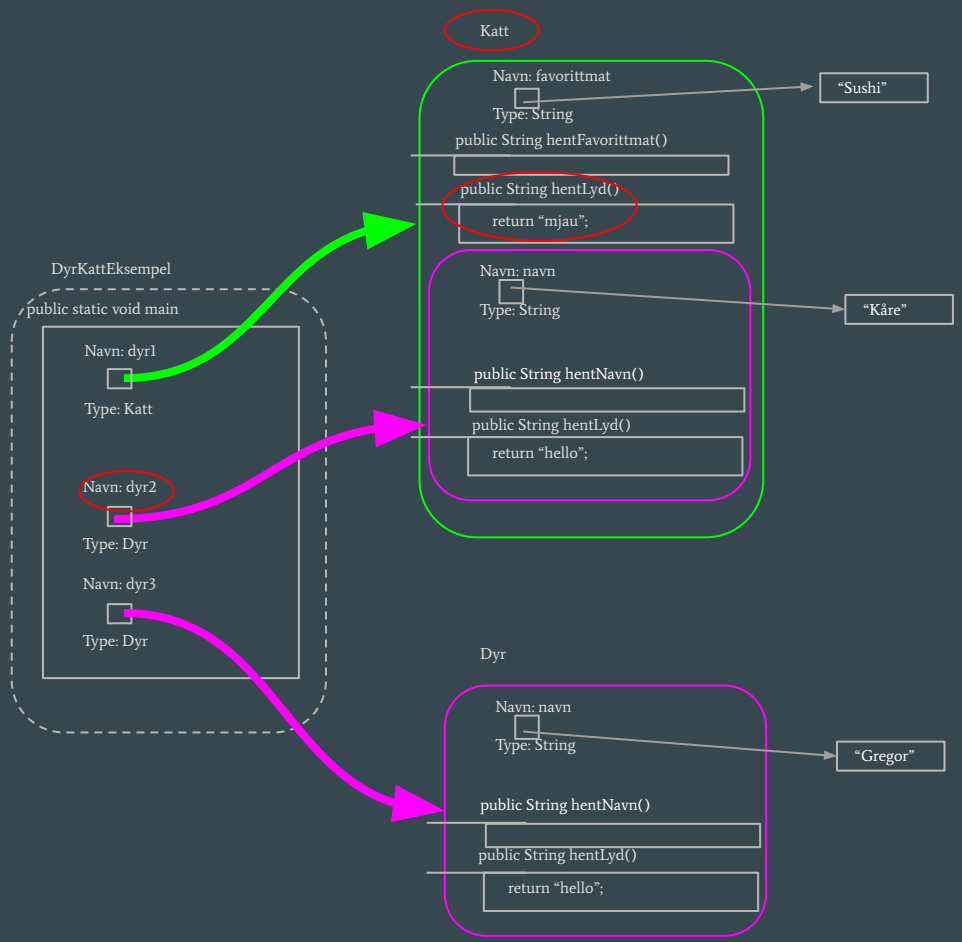
```java
class Katt extends Dyr{
    protected String favorittmat;

    public Katt(String navn, String favorittmat){
        super(navn);
        this.favorittmat = favorittmat;
    }
    public String hentFavorittMat(){
        return favorittmat;
    }
    public String hentLyd(){
        return "mjau";
    }
```

```java
class TestKatt{
    public static void main(String[] args){
        Katt dyr1 = new Katt("Kåre", "Sushi");
        Dyr dyr2 = (Dyr) dyr1;
        Dyr dyr3 = new Dyr("Gregor");
        System.out.println(dyr1.hentLyd());
        System.out.println(dyr2.hentLyd());
        System.out.println(dyr3.hentLyd());
        System.out.println(dyr1.hentNavn());
        System.out.println(dyr2.hentNavn());
        System.out.println(dyr3.hentNavn());
        System.out.println(dyr1.hentFavorittMat());
        System.out.println(dyr2.hentFavorittMat()); //ERROR
        System.out.println(dyr3.hentFavorittMat()); //ERROR
```

Casting

```
class TestKatt{
  public static void main(String[] args){
    Katt dyr1 = new Katt("Kåre", "Sushi");
    Dyr dyr2 = (Dyr) dyr1;
    Dyr dyr3 = new Dyr("Gregor");
    System.out.println(dyr1.hentLyd());   mjau
    System.out.println(dyr2.hentLyd());
    System.out.println(dyr3.hentLyd());
    System.out.println(dyr1.hentNavn());
    System.out.println(dyr2.hentNavn());
    System.out.println(dyr3.hentNavn());
    System.out.println(dyr1.hentFavorittMat());
    System.out.println(dyr2.hentFavorittMat()); //ERROR
    System.out.println(dyr3.hentFavorittMat()); //ERROR
```

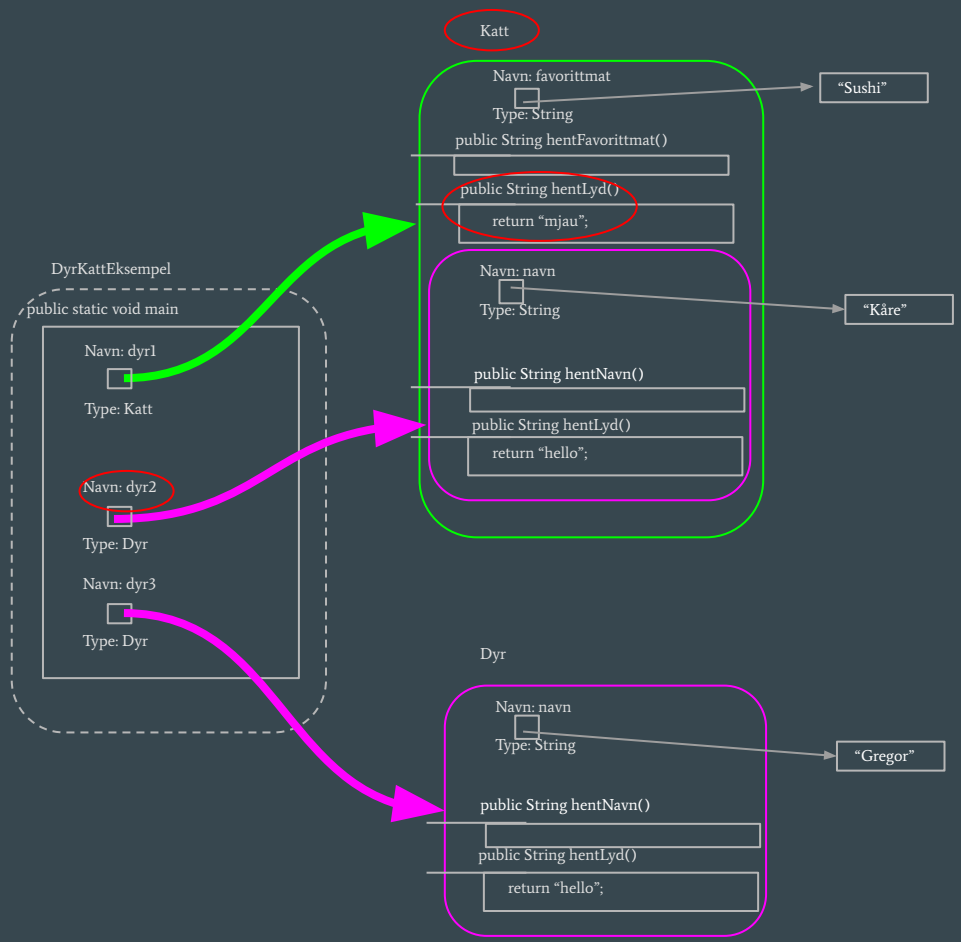


Vi må se på typen til objektet for å avgjøre hvilken hentLyd metode som kjører.

Casting

```
class TestKatt{
  public static void main(String[] args){
    Katt dyr1 = new Katt("Kåre", "Sushi");
    Dyr dyr2 = (Dyr) dyr1;
    Dyr dyr3 = new Dyr("Gregor");
    System.out.println(dyr1.hentLyd());
    System.out.println(dyr2.hentLyd());
    System.out.println(dyr3.hentLyd());
    System.out.println(dyr1.hentNavn());
    System.out.println(dyr2.hentNavn());
    System.out.println(dyr3.hentNavn());
    System.out.println(dyr1.hentFavorittMat());
    System.out.println(dyr2.hentFavorittMat()); //ERROR
    System.out.println(dyr3.hentFavorittMat()); //ERROR
```

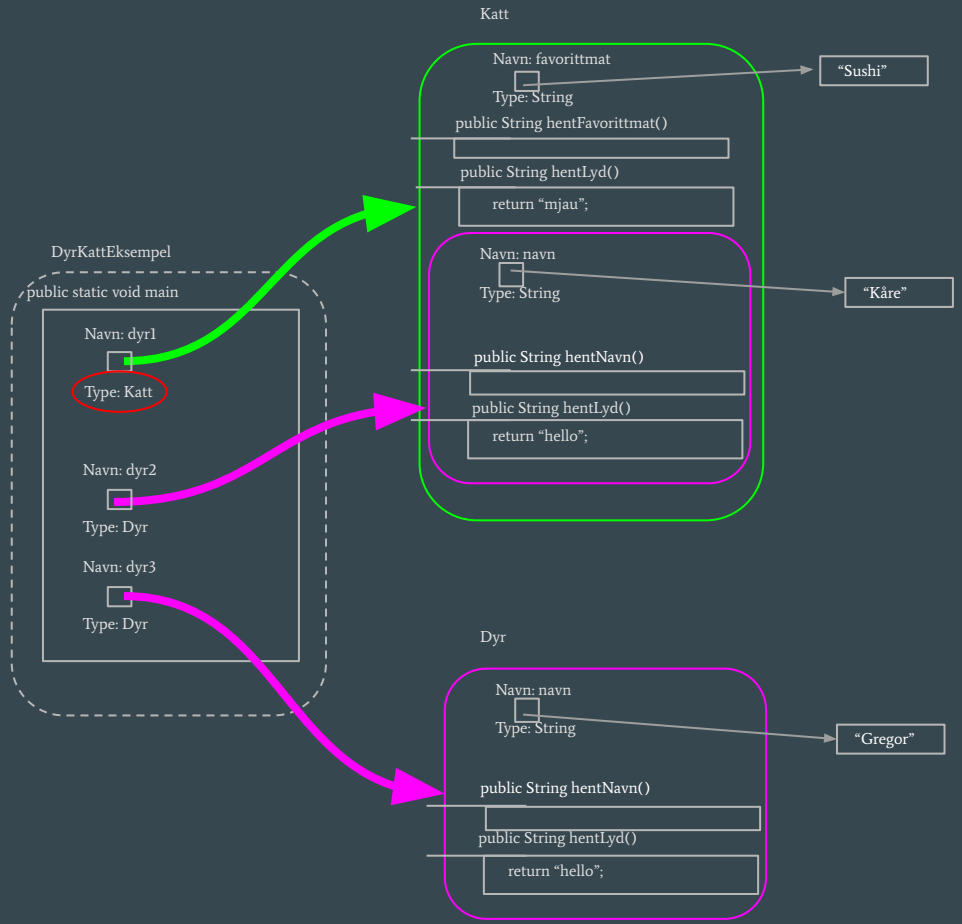
```
mjau
mjau
```



Vi må se på typen til objektet for å avgjøre hvilken hentLyd metode som kjører.

Casting

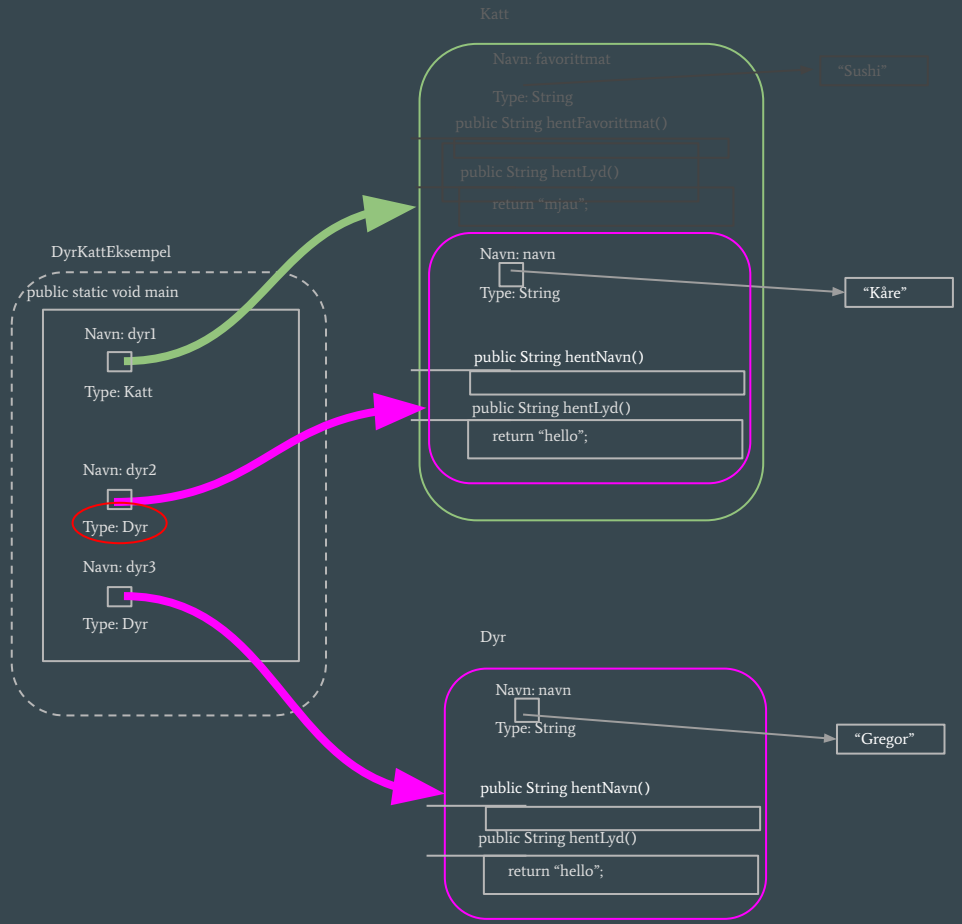
```
class TestKatt{
  public static void main(String[] args){
    Katt dyr1 = new Katt("Kåre", "Sushi");
    Dyr dyr2 = (Dyr) dyr1;
    Dyr dyr3 = new Dyr("Gregor");
    System.out.println(dyr1.hentLyd());          mjau
    System.out.println(dyr2.hentLyd());          mjau
    System.out.println(dyr3.hentLyd());          hello
    System.out.println(dyr1.hentNavn());         Kåre
    System.out.println(dyr2.hentNavn());         Kåre
    System.out.println(dyr3.hentNavn());         Gregor
    System.out.println(dyr1.hentFavorittMat());  Sushi
    System.out.println(dyr2.hentFavorittMat()); //ERROR
    System.out.println(dyr3.hentFavorittMat()); //ERROR
```



Vi må se på typen til referansen for å avgjøre hva vi kan aksessere.

Casting

```
class TestKatt{
  public static void main(String[] args){
    Katt dyr1 = new Katt("Kåre", "Sushi");
    Dyr dyr2 = (Dyr) dyr1;
    Dyr dyr3 = new Dyr("Gregor");
    System.out.println(dyr1.hentLyd());          mjau
    System.out.println(dyr2.hentLyd());          mjau
    System.out.println(dyr3.hentLyd());          hello
    System.out.println(dyr1.hentNavn());         Kåre
    System.out.println(dyr2.hentNavn());         Kåre
    System.out.println(dyr3.hentNavn());         Gregor
    System.out.println(dyr1.hentFavorittMat());  Sushi
    System.out.println(dyr2.hentFavorittMat()); //ERROR
    System.out.println(dyr3.hentFavorittMat()); //ERROR
```

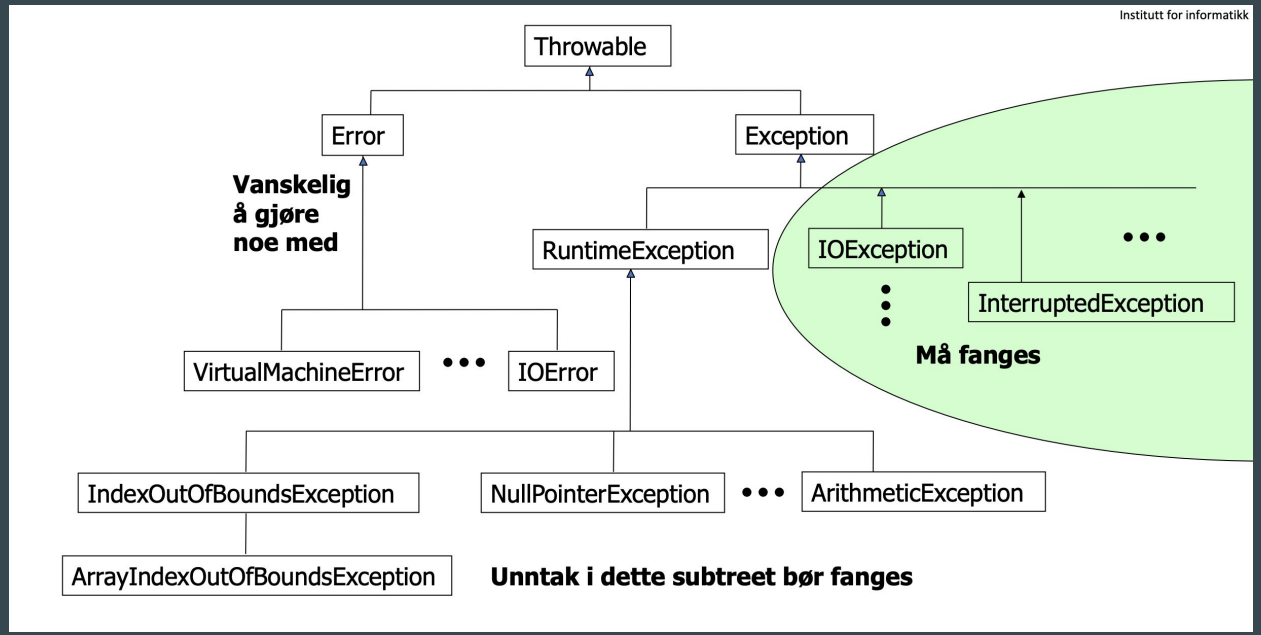


Vi må se på typen til variabelen for å avgjøre hva vi kan aksessere.

Vi tar på oss dyr-brillene

Feilmeldinger og feilhåndtering

Vi kan lage våre egne feilmeldinger eller bruke de som allerede finnes:



Repetisjon denne uken

I dag

Tråder

Lock

Monitor

Condition

Tips og triks

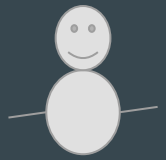
Les boka:
https://higheredbcs.wiley.com/legacy/college/horstmann/1118087887/web_ch/ch20.pdf

Les dette notatet:
https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/notater/trader-8mars.pdf

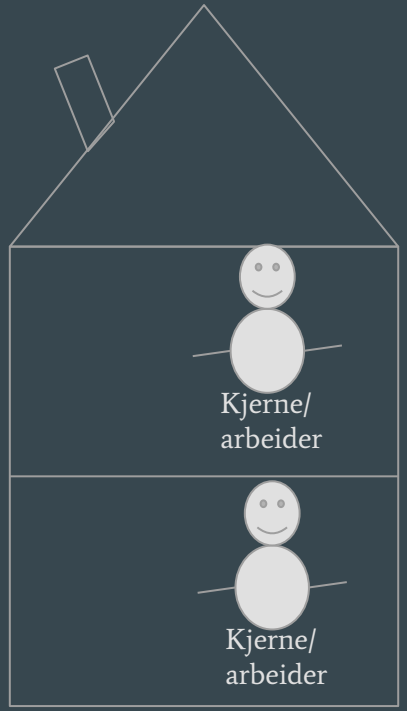
Det er viktig å programmere masse, men du blir en enda bedre utvikler hvis du skjønner teorien bak det du driver med også.

Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).



Utviklere

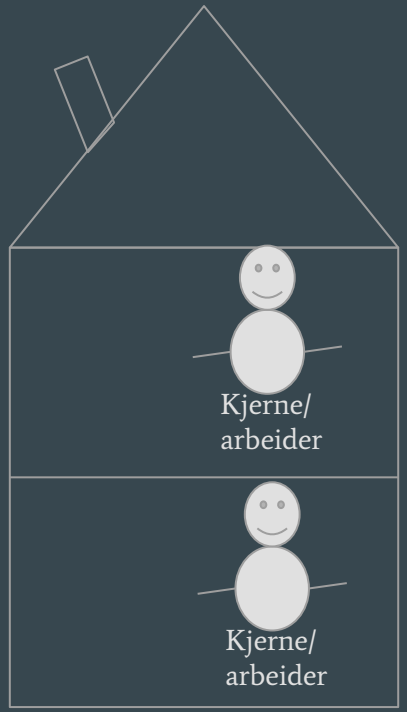


CPU/kontoret/fabrikken

Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

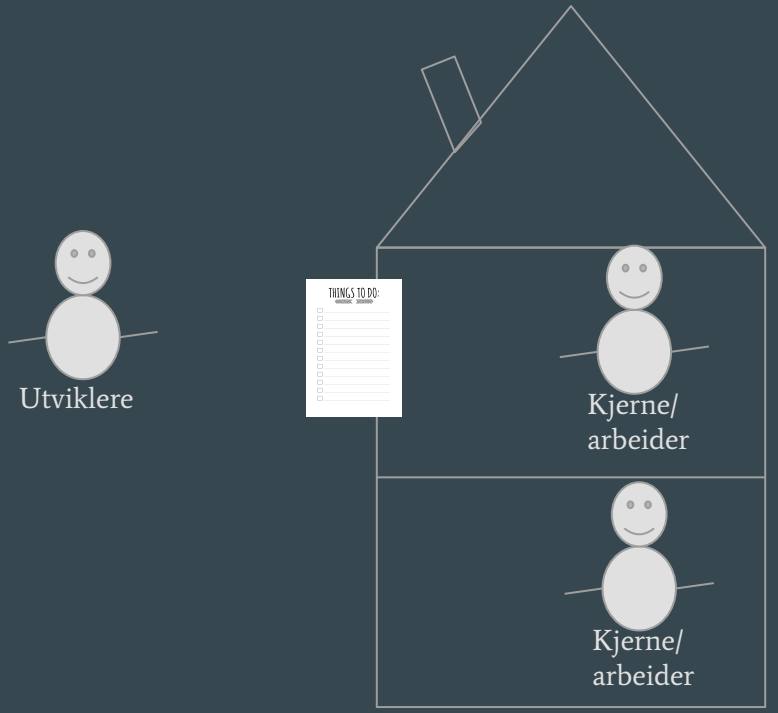
Utviklere



CPU/kontoret/fabrikken

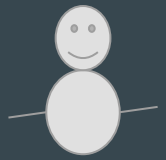
Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

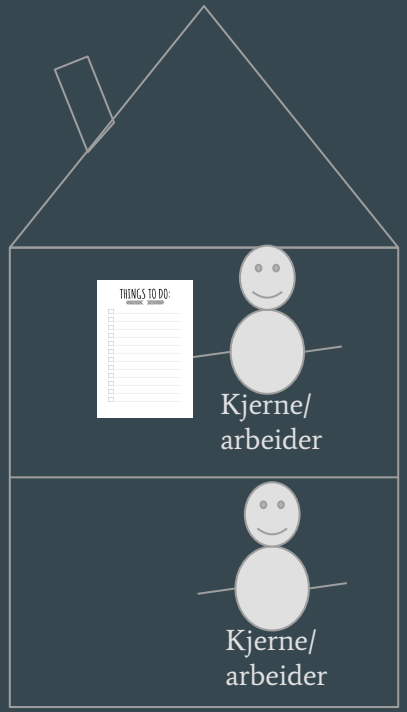


Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).



Utviklere



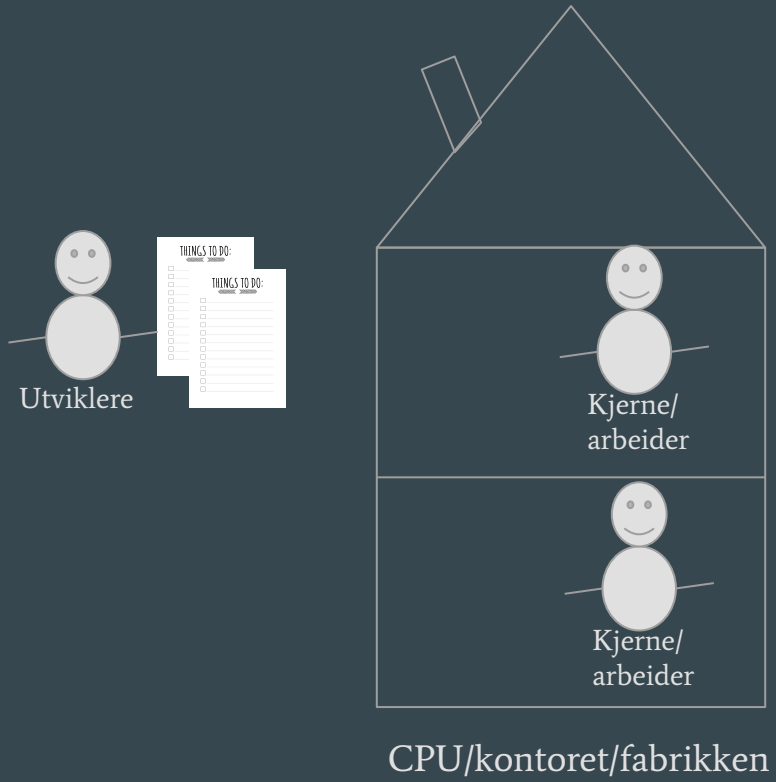
CPU/kontoret/fabrikken

Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.

Da må vi dele opp programmet vårt!

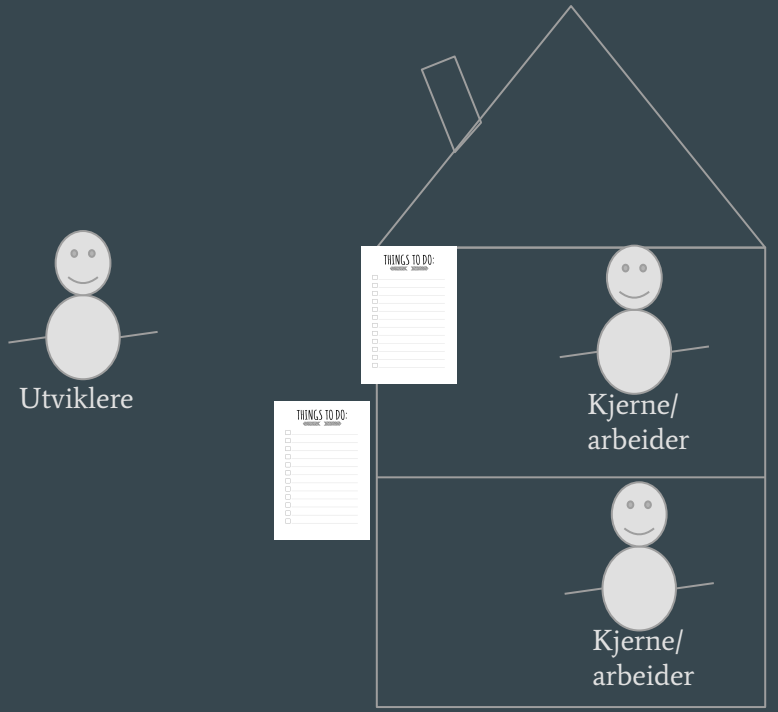


Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.

Da må vi dele opp programmet vårt!

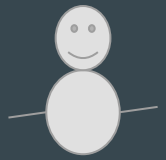


Hvorfor trenger vi tråder?

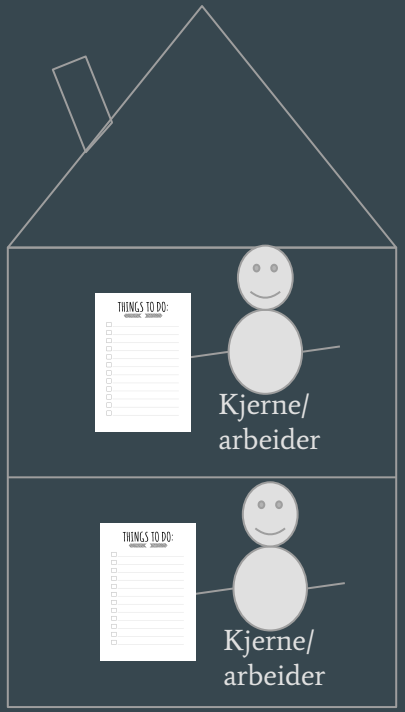
Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.

Da må vi dele opp programmet vårt!



Utviklere



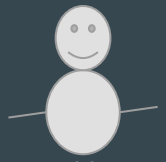
CPU/kontoret/fabrikken

Hvorfor trenger vi tråder?

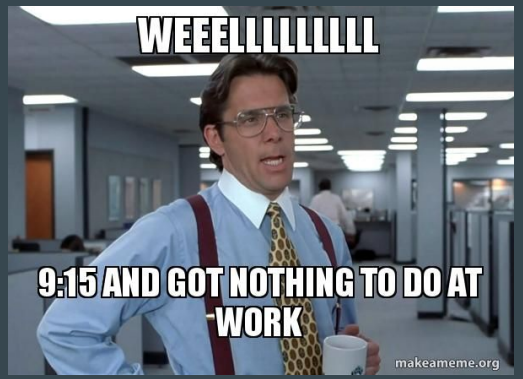
Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

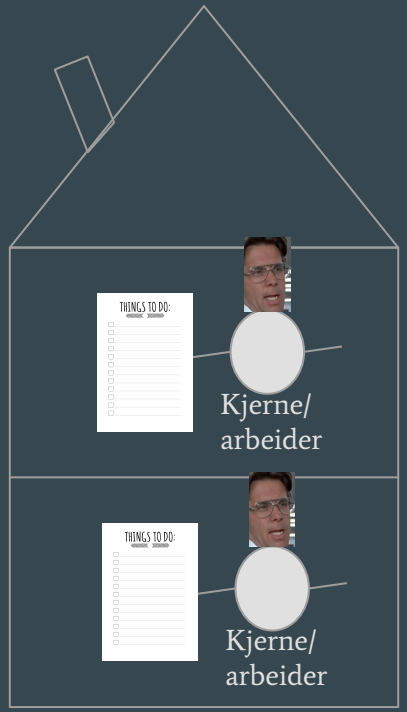
1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.

Noen ganger er det dødtid for kjernene, de må vente, f.eks. på at ting blir hentet fra minnet



Utviklere



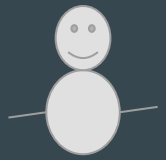


CPU/kontoret/fabrikken

Hvorfor trenger vi tråder?

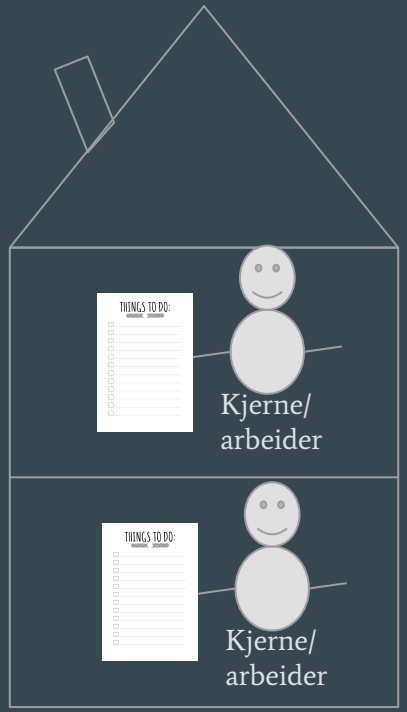
Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.
2. Hvis en kjerne kan jobbe på ulike deler av programmet vårt “samtidig” går det raskere.



Utviklere

Skjer ikke egentlig samtidig, de bytter på hva de jobber med



CPU/kontoret/fabrikken

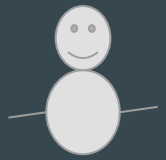
Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

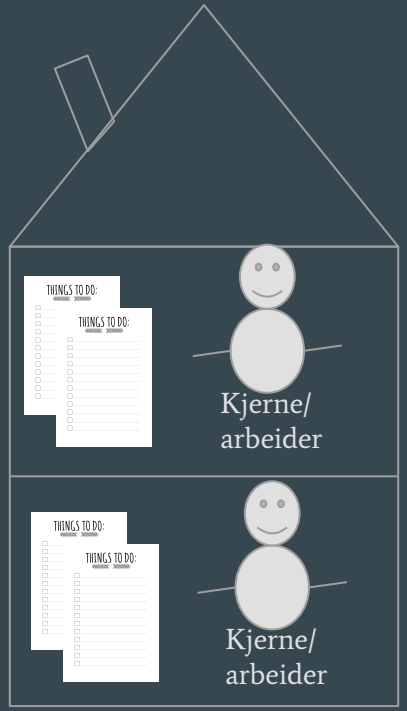
1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.
2. Hvis en kjerne kan jobbe på ulike deler av programmet vårt “samtidig” går det raskere.

Skjer ikke egentlig samtidig, de bytter på hva de jobber med

Da må vi dele opp programmet vårt!



Utviklere



CPU/kontoret/fabrikken

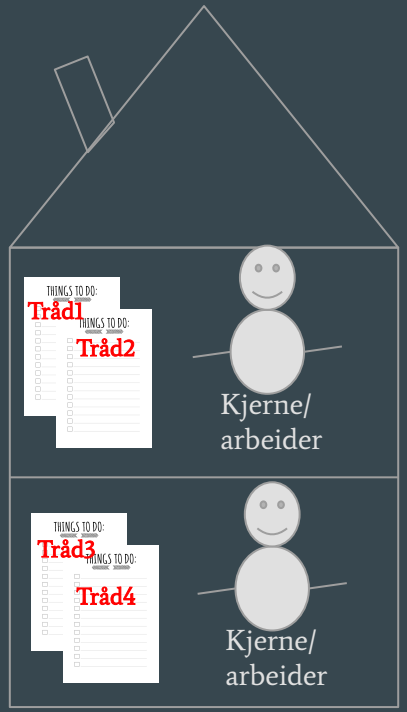
Hvorfor trenger vi tråder?

Kjernene finnes i CPUen, en CPU kan ha flere kjerner(vanlig), og en datamaskin kan ha flere CPUer(ikke så vanlig).

1. Hvis flere kjerner kan jobbe på programmet vårt samtidig går det raskere.
2. Hvis en kjerne kan jobbe på ulike deler av programmet vårt “samtidig” går det raskere.

Skjer ikke egentlig samtidig, de bytter på hva de jobber med

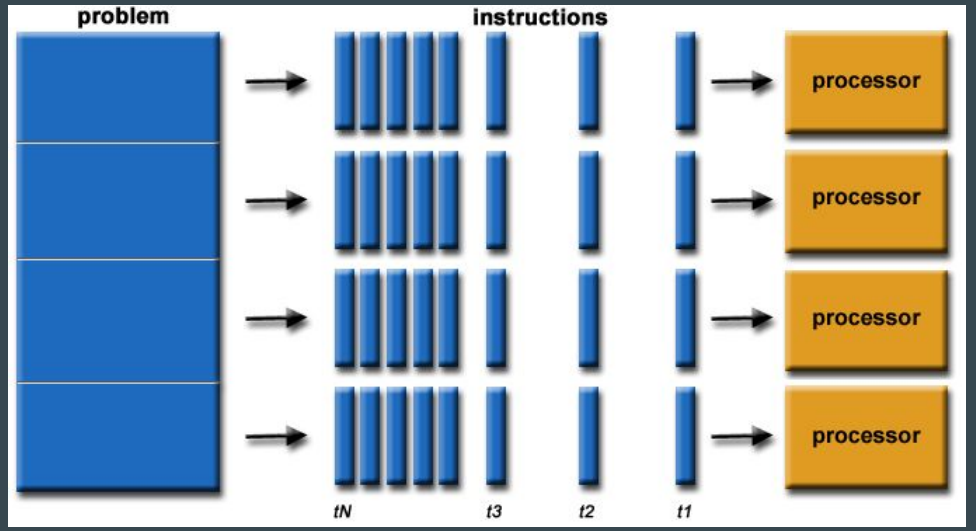
Både 1) og 2) løses med tråder(gjøre det mulig å dele opp programmet).



CPU/kontoret/fabrikken

Hva er et parallelt program og hvorfor bruker vi det

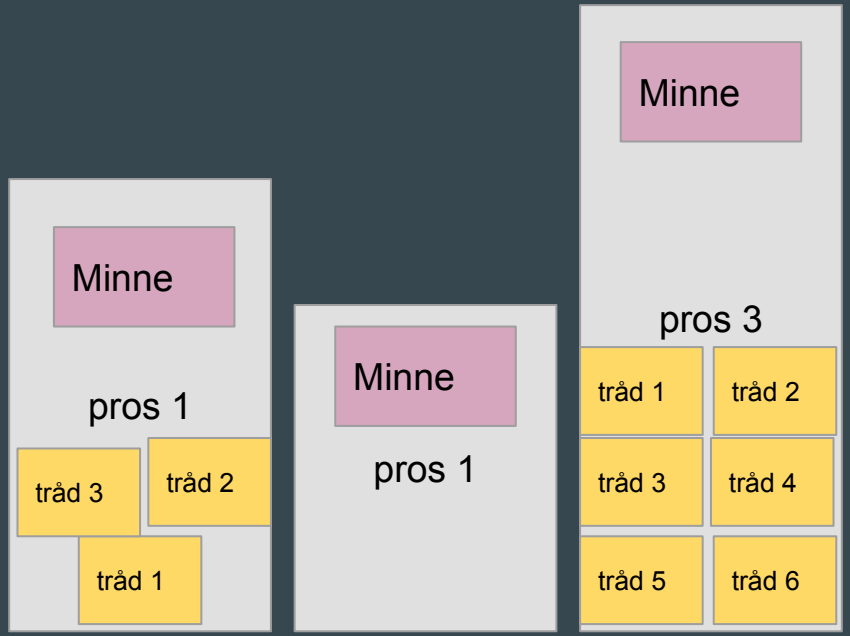
- CPU/ Kjerner
- Flere programmer får kjørt
- Maskinen har kapasiteten
- Dele opp problemet
- Raskere, men
  - Det tar tid å starte opp en tråd
  - Synkronisering



Bildekilde: https://computing.llnl.gov/tutorials/parallel_comp/

Hva er en tråd?

- En parallell eksekvering inne i en prosess
  - En prosess er utføringen av et program
- Deler minne til prosessen
- Tråder kan gi i ekte parallel
- “Små”-prosesser i en vanlig prosess
- En sekvens med instruksjoner



Tråder step by step (fra boka)

1. Write a class that implements the `Runnable` interface. That interface has a single method called `run`:

```
public interface Runnable
{
   void run();
}
```

2. Place the code for your task into the `run` method of your class:

```
public class MyRunnable implements Runnable
{
   public void run()
   {
      Task statements
      . . .
   }
}
```

3. Create an object of your subclass:

```
Runnable r = new MyRunnable();
```

4. Construct a `Thread` object from the runnable object:

```
Thread t = new Thread(r);
```

5. Call the `start` method to start the thread:

```
t.start();
```

Tråder step by step

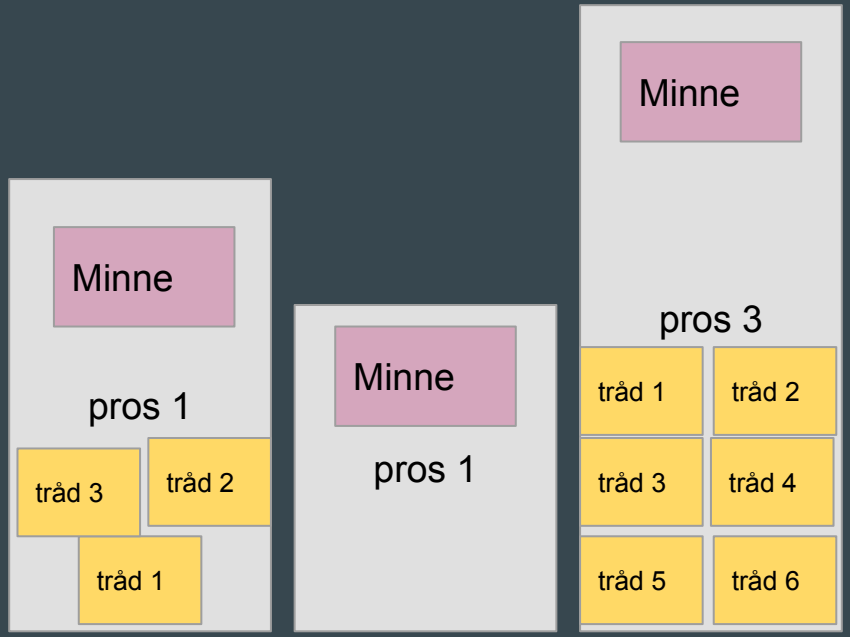
- Steg 1 og 2: Runnable og run()
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html
  - Et interface
  - run()
    - Når du bruker .start() på en tråd vil objektets run metode bli kalt
- Steg 3 og 4: Thread
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#run()
  - En klasse
  - start()
  - Tar inn vår Runnable klasse i konstruktøren

Live-programmering

Lag et program som starter opp 3 tråder som printer en melding ti ganger. Trådene skal ta inn en melding.

Delt data

- Data flere tråder deler
- Prosessens data
- Skriving
- Lesing
- Lesing og skriving
- Er du usikker lås



Locks

- For å bruke en Lock må man importer
  - java.util.concurrent.locks.Lock (Et grensesnitt, bl.a. to metoder lock() og unlock())
  - java.util.concurrent.locks.ReentrantLock (Lager de faktiske låseobjekten)
- Passer på at kun en tråd kan utføre koden om gangen
- Når man bruker Lock med lock() og unlock() er det viktig at man bruker en try-catch blokk. Nå må man også huske finally.
  - Finally er en kodesnutt som vil skje uansett. Når det kommer til Locks må man passe på å alltid unlock() i en finally blokk slik at hvis det skulle skje noe med en tråd kan de andre trådene få fortsette

```
46    public void metode(){
47      lock.lock()
48      try{
49        <kode her>
50      }catch(Exception e){
51        <hvis det er en exception og catche>
52      }finally{
53        lock.unlock() //vil alltid skje
54      }
```

Monitor

- Et objekt som innkapsler den delta dataen
- Fungere som en type beskytter av felles data
- Monitor objektet har metoder som gjør at de andre klassen f.eks. kan endre data og hente ut data

PS: Den metoden blir brukt i mye i 1010 fordi det er en god objektorientert måte å gjøre synkronisering på

Live-koding

Vi skal kode et program som starter opp 3 tråder (disse skal ta inn et tall i konstruktøren). Alle trådene skal prøve å legge til sitt tall i et felles array som ligger inne i en monitor. Det skal signaliseres tydelig til trådene når det ikke lenger er plass til flere tall i arrayet. Videre skal også tråden printe en liten beskjed som inneholder tallet sitt hver gang de får lag til tallet i arrayet.

Condition

- En form for passiv venting
  - Vil bruke mindre ressurser enn aktiv venting
- await(): Får den tråden man er inne i nå til å vente til Condition gir signaliserer at det er greit å fortsette. (Eller at den blir interrupted)
- signal() : Signaliserer til en tråd om at den kan fortsett. (Som står å venter ved await())
- signalAll(): Signaliserer til alle ventene tråder

Jobbe selv

Jobb med hva dere vil og rekk opp hånda hvis dere trenger hjelp med noe/har spørsmål så møtes vi i breakoutroom! 😊