IN1010 - Seminar 11

- Tråder

Laget av: Marlen Jarholt (marleja@ifi.uio.no) og Johanna Haarseth (johannph@ifi.uio.no)

Praktisk

Husk å sjekke emnesiden regelmessig

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!

Oblig 5 er nå publisert (ca. 3 uker på dere). HUSK at denne skal inn rett etter påsken, som betyr at det ikke vil være noen labtimer/gruppetimer og at gruppelærere/faglærere ikke er tilgjengelig den siste uken på mattermost. Kan derfor være lurt å prøve å bli ferdig før ferien stater.

Send Marlen en direktemelding i chatten

Vil du jobbe sammen med noen andre ? (ja /nei)

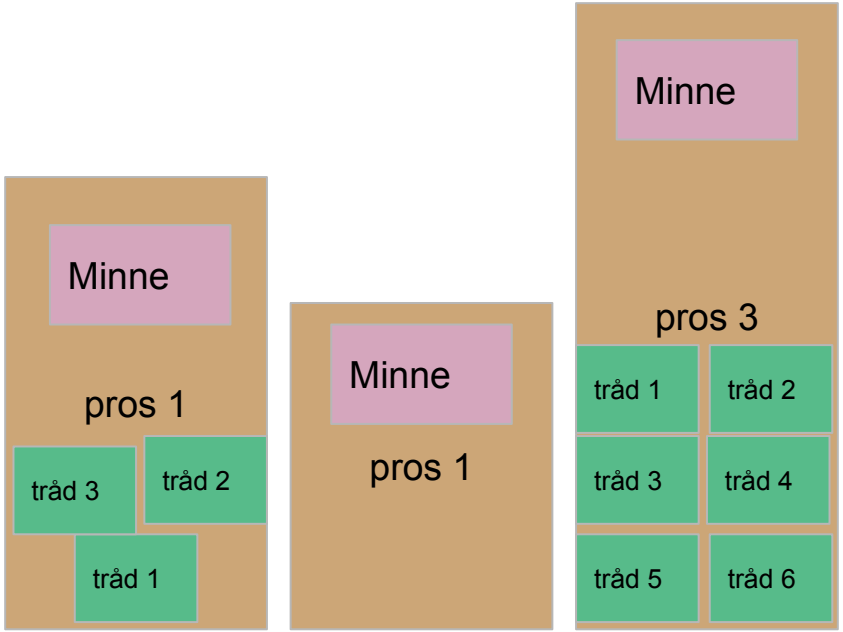
Hvis du har noen ønsker på hvem du vil jobbe med, så send det i samme melding

Svar gjerne også om svaret skulle være nei

Repetisjon

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



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

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

CountDownLatch

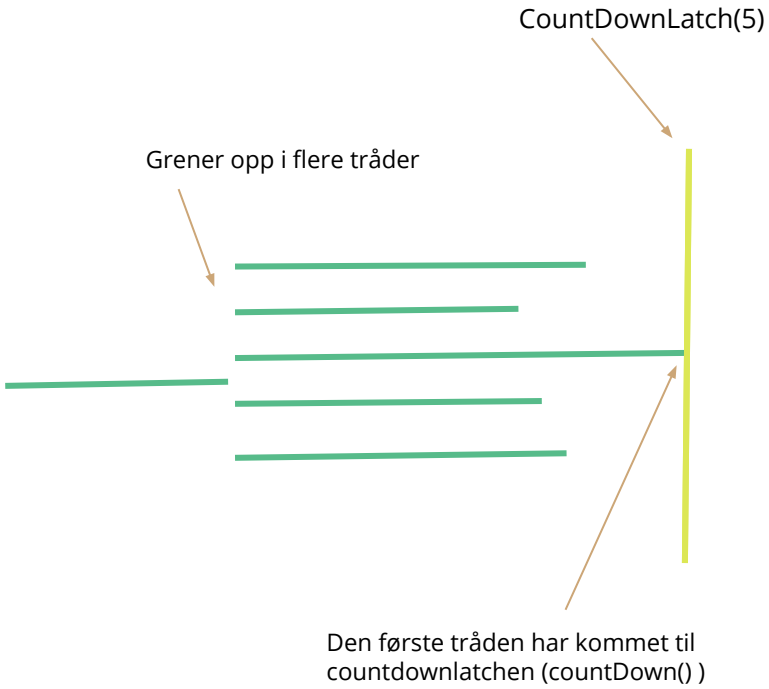
- Konstruktøren tar inn antall tråder den skal vente på
- Nyttige metoder:
  - await(), Tråden blir stoppet her og står å venter til countern har telt seg ned til 0. Denne metoden kan kaste unntak! (Derfor må man bruke try-catch)
  - countDown(): Teller ned counteren en gang.
  - getCount(): returnerer countern (sagt med andre ord hvor langt den har kommet i nedtellingen)

DOKUMMENTASJON: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CountDownLatch.html

```java
class EksempelCountDwon {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CountDownLatch cdl = new CountDownLatch(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CountDownTraad(cdl)).start();
        }
        System.out.println("Hovedtraad venter");
        try {
            cdl.await();
        } catch (InterruptedException e) {
            System.out.println("Ble forstyrret");
        }
        System.out.println("Hovedtraad ferdig");
    }

}

class CountDownTraad implements Runnable{

    CountDownLatch cdl;

    public CountDownTraad(CountDownLatch cdl){
        this.cdl = cdl;
    }

    @Override
    public void run(){
        System.out.println("CountDown");
        cdl.countDown();
    }
}
```

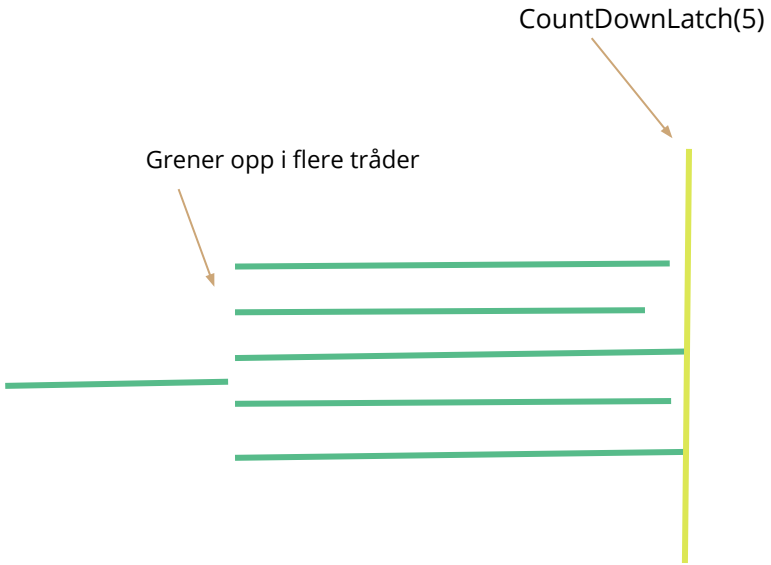

```java
class EksempelCountDwon {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CountDownLatch cdl = new CountDownLatch(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CountDownTraad(cdl)).start();
        }
        System.out.println("Hovedtraad venter");
        try {
            cdl.await();
        } catch (InterruptedException e) {
            System.out.println("Ble forstyrret");
        }
        System.out.println("Hovedtraad ferdig");
    }

}
```

```java
class CountDownTraad implements Runnable{

    CountDownLatch cdl;

    public CountDownTraad(CountDownLatch cdl){
        this.cdl = cdl;
    }

    @Override
    public void run(){
        System.out.println("CountDown");
        cdl.countDown();
    }
}
```



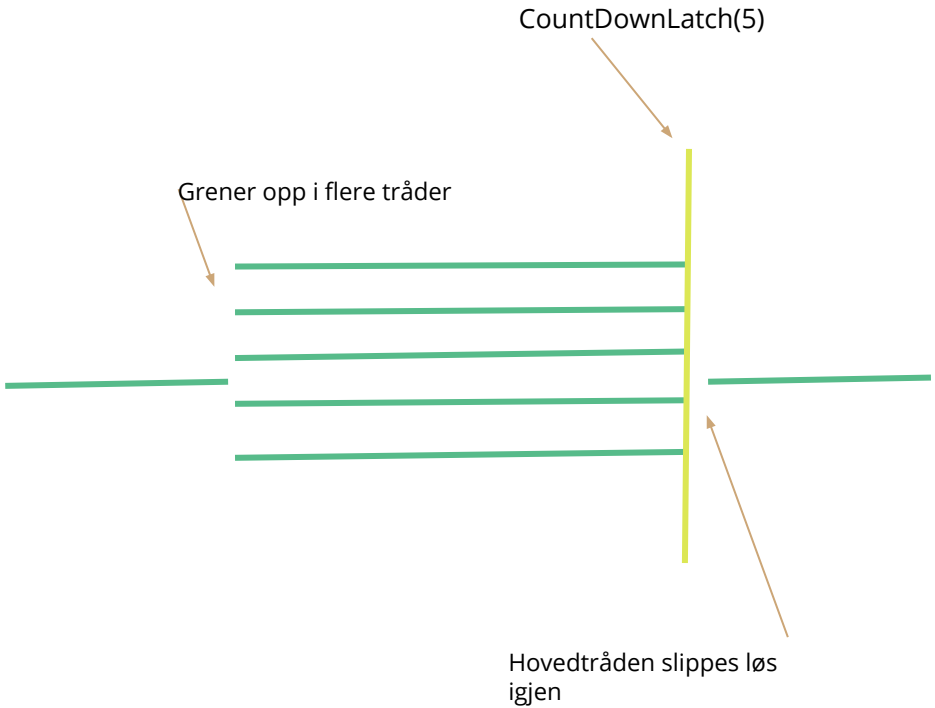

```java
class EksempelCountDwon {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CountDownLatch cdl = new CountDownLatch(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CountDownTraad(cdl)).start();
        }
        System.out.println("Hovedtraad venter");
        try {
            cdl.await();
        } catch (InterruptedException e) {
            System.out.println("Ble forstyrret");
        }
        System.out.println("Hovedtraad ferdig");
    }

}
```

```java
class CountDownTraad implements Runnable{

    CountDownLatch cdl;

    public CountDownTraad(CountDownLatch cdl){
        this.cdl = cdl;
    }

    @Override
    public void run(){
        System.out.println("CountDown");
        cdl.countDown();
    }
}
```




```java
class EksempelCountDwon {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CountDownLatch cdl = new CountDownLatch(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CountDownTraad(cdl)).start();
        }
        System.out.println("Hovedtraad venter");
        try {
            cdl.await();
        } catch (InterruptedException e) {
            System.out.println("Ble forstyrret");
        }
        System.out.println("Hovedtraad ferdig");
    }

}
```

```java
class CountDownTraad implements Runnable{

    CountDownLatch cdl;

    public CountDownTraad(CountDownLatch cdl){
        this.cdl = cdl;
    }

    @Override
    public void run(){
        System.out.println("CountDown");
        cdl.countDown();
    }
}
```




```java
class EksempelCountDwon {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CountDownLatch cdl = new CountDownLatch(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CountDownTraad(cdl)).start();
        }
        System.out.println("Hovedtraad venter");
        try {
            cdl.await();
        } catch (InterruptedException e) {
            System.out.println("Ble forstyrret");
        }
        System.out.println("Hovedtraad ferdig");
    }

}
```

```java
class CountDownTraad implements Runnable{

    CountDownLatch cdl;

    public CountDownTraad(CountDownLatch cdl){
        this.cdl = cdl;
    }

    @Override
    public void run(){
        System.out.println("CountDown");
        cdl.countDown();
    }
}
```

CyclicBarrier

- Konstruktøren tar inn antall tråder som skal synkroniseres på et tidspunkt
- Metoder:
  - await(): Venter til alle trådene kommer til denne barrieren. Kan kaste unntak(Husk try-catch)

DOKUMENTASJON: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CyclicBarrier.html

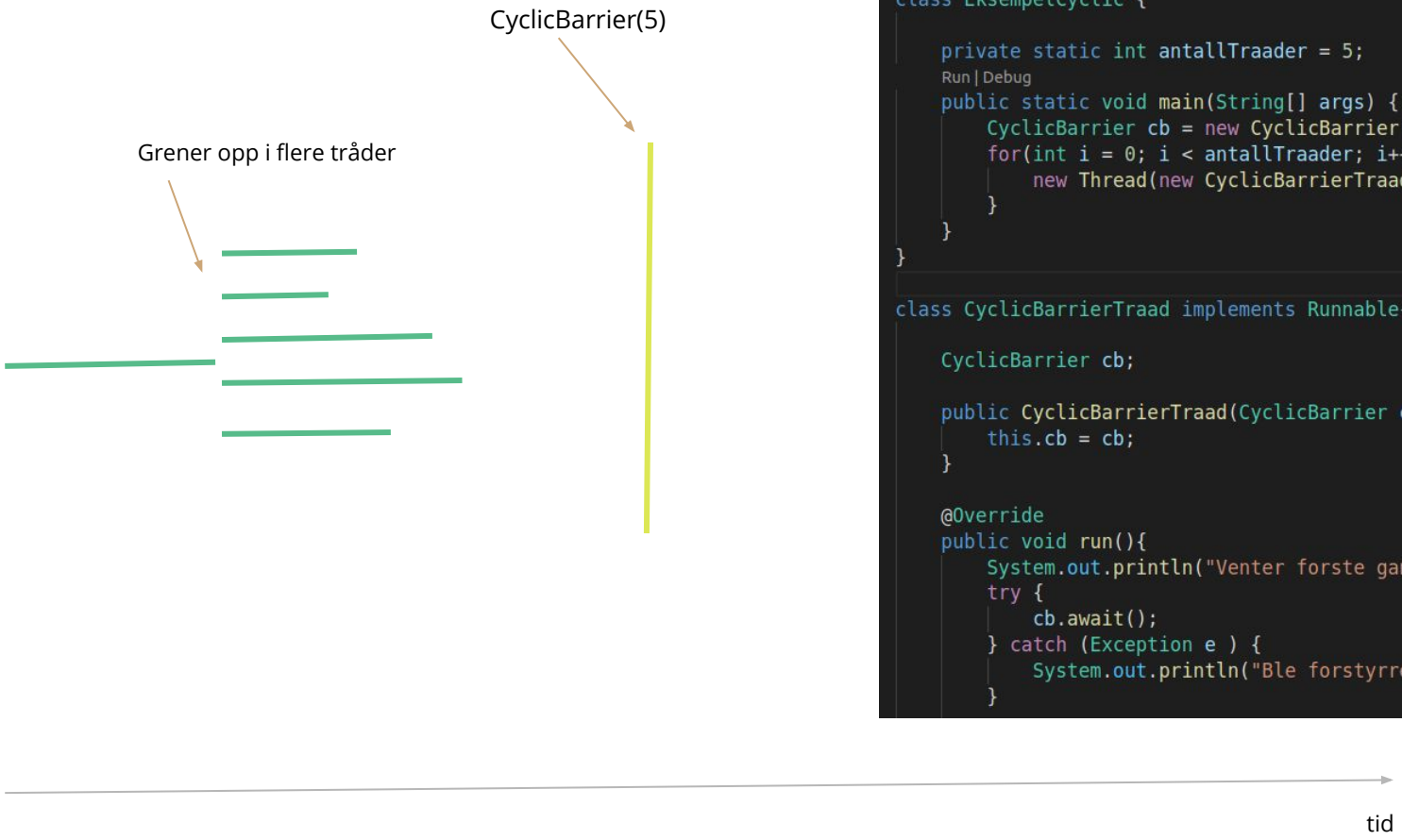
```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}

class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }

        System.out.println("Venter andre gang");
        try {
            cb.await();
        } catch (Exception e) {
            System.out.println("Ble forstyrret");
        }
    }
}
```

Grener opp i flere tråder

```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}
```

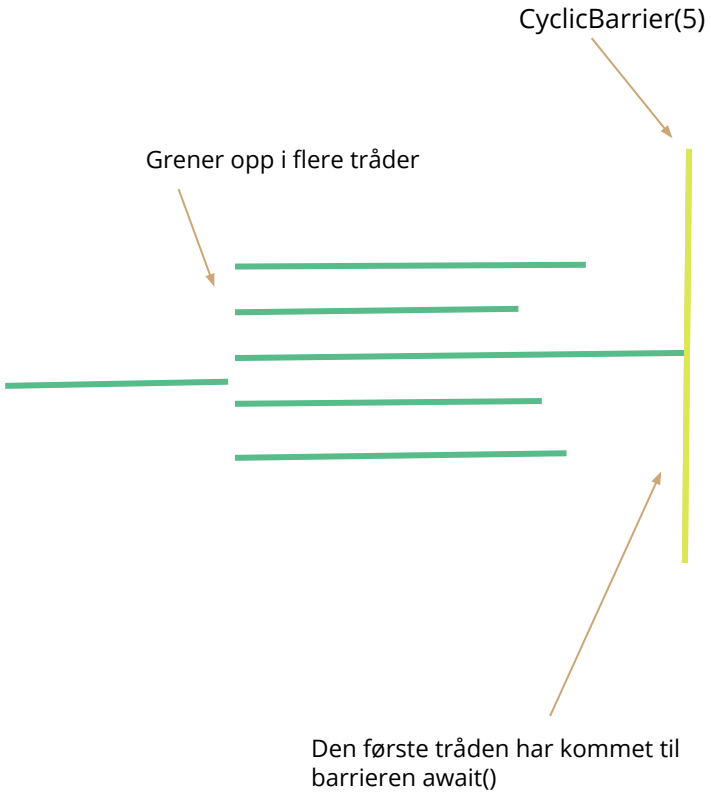
```java
class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }
```

tid



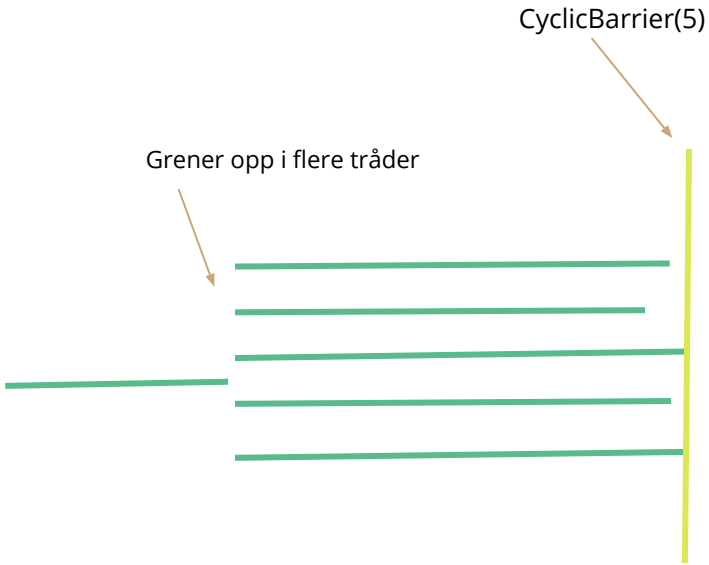

```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}
```

```java
class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }
```



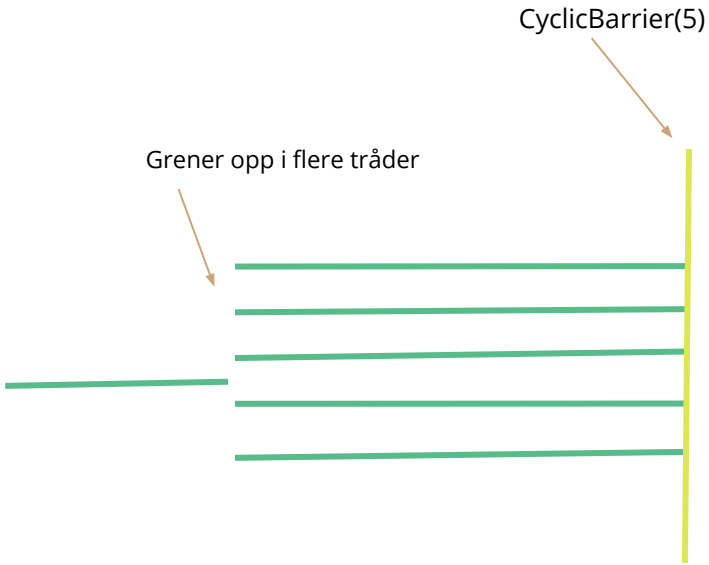

```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}
```

```java
class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }
```




```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}
```

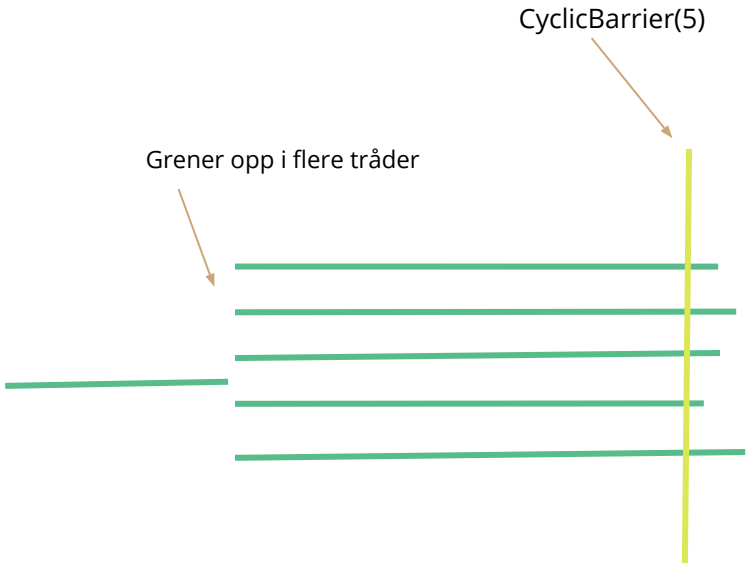
```java
class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }
```




```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}
```

```java
class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }
```

tid




```java
class EksempelCyclic {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);
        for(int i = 0; i < antallTraader; i++){
            new Thread(new CyclicBarrierTraad(cb)).start();
        }
    }
}
```

```java
class CyclicBarrierTraad implements Runnable{

    CyclicBarrier cb;

    public CyclicBarrierTraad(CyclicBarrier cb){
        this.cb = cb;
    }

    @Override
    public void run(){
        System.out.println("Venter forste gang");
        try {
            cb.await();
        } catch (Exception e ) {
            System.out.println("Ble forstyrret");
        }
```

Join

Join er en metode du kan kalle join på en tråd.

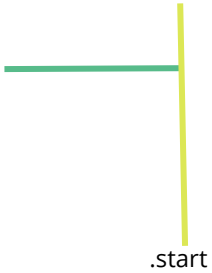
Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)

```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}
class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Join

Join er en metode du kan kalle join på en tråd.

Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)

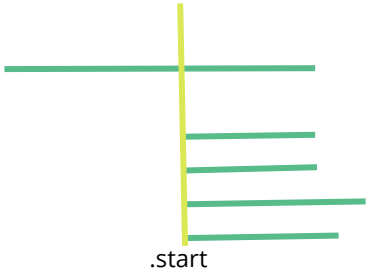



```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}
class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Join

Join er en metode du kan kalle join på en tråd.

Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)

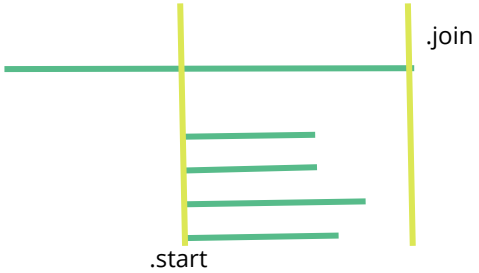



```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}

class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Join

Join er en metode du kan kalle join på en tråd.

Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)

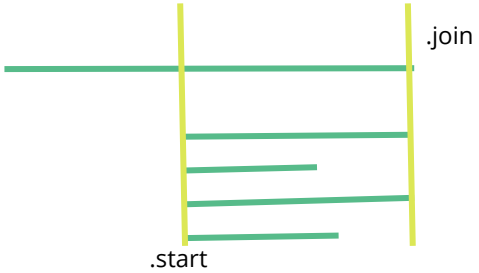



```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}
class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Join

Join er en metode du kan kalle join på en tråd.

Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)

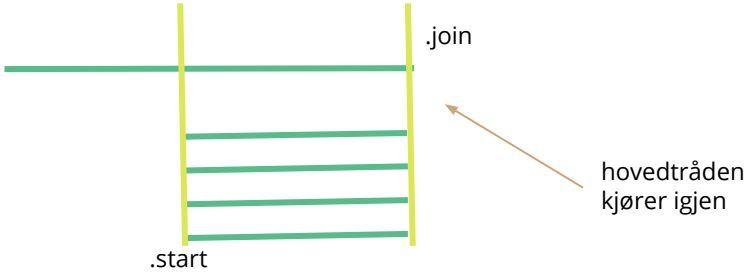



```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}
class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Join

Join er en metode du kan kalle join på en tråd.

Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)

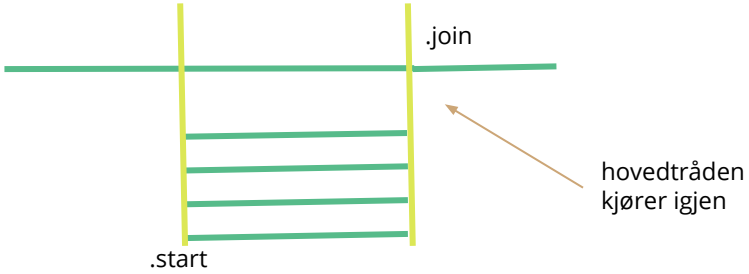



```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}
class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Join

Join er en metode du kan kalle join på en tråd.

Da vil tråden du er i vente til den tråden som kalte på join metoden er terminert (ferdig)




```java
class EksempelJoin {

    private static int antallTraader = 5;
    Run | Debug
    public static void main(String[] args) {

        ArrayList<Thread> traader = new ArrayList<>(antallTraader);
        for(int i = 0; i < antallTraader; i ++){
            Thread traad = new Thread(new JoinTraad());
            traader.add(traad);
            traad.start();
        }

        for(Thread traad : traader){
            try {
                traad.join();
            } catch (InterruptedException e) {
                System.out.println("Interupt feil");
            }
        }
        System.out.println("Hovedtraad ferdig");
    }

}

class JoinTraad implements Runnable{

    @Override
    public void run(){
        System.out.println("Kjorer run metoden");
    }
}
```

Tips oblig 5

Hvilke deler trenger vi?

En beholder til alle hasMapsene:

disse trådene avsluttes.

Delt data!

Du skal programmere en beholderklasse som kan inneholde disse hashmapene. Programmet opprette to objekter av denne beholderklassen. Den ene beholderen skal innholde alle hashmap fra filer fra personer som har hatt viruset (filer merket true), den andre fra personer som ikke har

Slå sammen to og to hashMaps(Tråd):

Du skal programmere enda en trådklasse hvis oppgave er å hente ut to hashmaper fra en av disse beholderene, slå dem sammen og legge resultatet tilbake i beholderen. En sammenslåing utføres ved å legge sammen antall forekomster av like subsekvenser. Dersom en subsekvens går igjen i de to hashmapene som prosesseres, skal altså den resulterende hashmapen ha denne subsekvensen som én av sine nøkler, hvor verdien skal være summen av verdiene knyttet til subsekvensene i de to hashmapene som ble prosessert. Subsekvenser som ikke er like i de to hashmapene overføres uforandret til resultatet. Først når det bare er én hashmap igjen i hver beholder terminerer disse trådene.

Lese fra fil og legge data inn i hashMaps(Tråd):

Oppgavebeskrivelse:

Programmet ditt skal lese N filer, der hver fil representerer immunrepertoaret til en person. Hver fil inneholder en rekke linjer eller sekvenser der hver sekvens representerer en immunreseptor. I tillegg er det en metadatafil der hver linje inneholder et filnavn og informasjonen om filen inneholder data som kommer fra en person som har hatt viruset (true) eller kommer fra en person som vi tror ikke har hatt viruset (false).

Programmet skal lete etter subsekvenser av en gitt lengde i immunreseptorene. Denne lengden skal være en konstant i programmet. Vanligvis vil subsekvenslengden være 3 eller 4. Programmet skal først gå gjennom alle filene og for hver fil produsere en mengde av alle subsekvenser av den gitte lengden. Fordi dette er en mengde, vil det ikke være noen duplikater, se eksemplet på slutten.

Men fordi disse mengdene skal behandles videre, skal programmet lagre dem i hashmaper. Nøklene til disse hashmapene skal være en streng med subsekvensen, verdiene skal være objekter som skal inneholde to instansvariabler: subsekvensen og et antall som er én i utgangspunktet. Det vil være én selv om en fil inneholder flere forekomster av samme subsekvens.

Med N filer vil det derfor bli opprettet N hashmaper. For raskere å kunne finne subsekvensene, skal hver fil behandles av en tråd. For enkelhets skyld kan du, hvis du ønsker det, opprette en tråd for hver fil. Ønsker du en større utfordring kan du anta at denne jobben skal gjøres av K tråder, der K er mindre enn antall filer. Når en tråd har fullført behandling av en fil og opprettet en hashmap, så

Bruk monitor design prinsippet

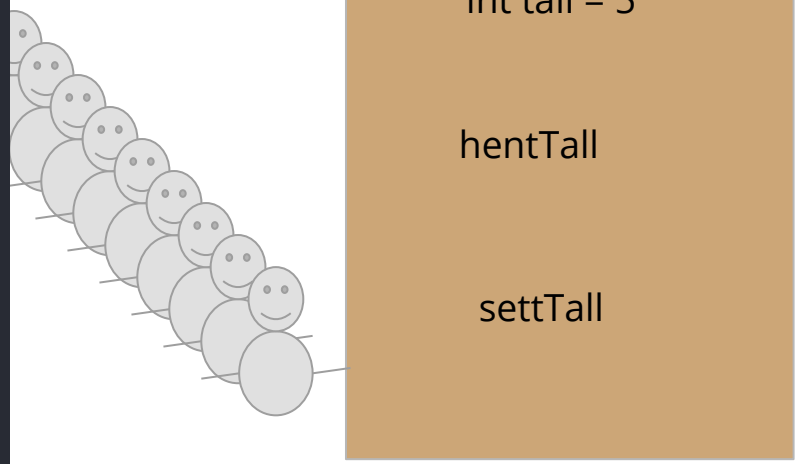
En monitor-klasse er bare en betegnelse på en klasse som tar vare på delt data.

Delt data bør aksesseres av én og én tråd om gangen. Vi bruker lock slik at ingen andre kan komme inn.

Litt som et toalettavlukke eller en voting boot, der er det bare plass til én om gangen.

Bruk monitor design prinsippet

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```

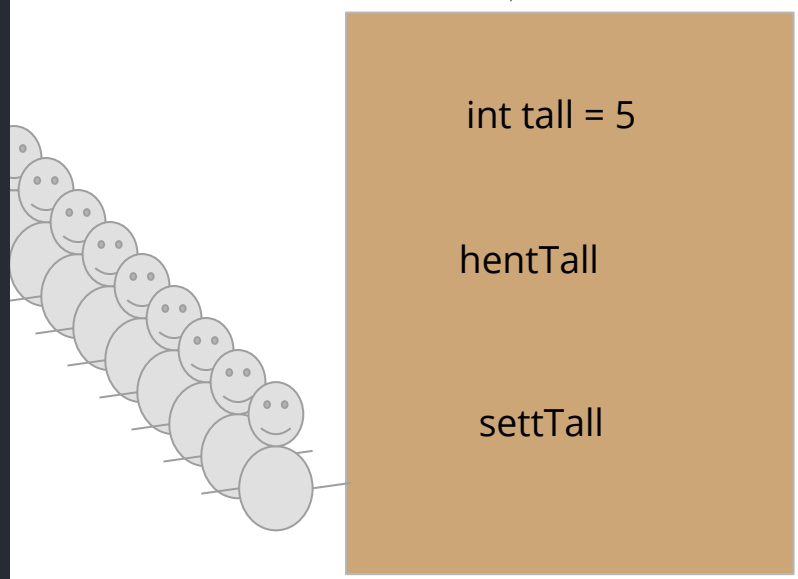


Monitor klasse

Bruk monitor design prinsippet

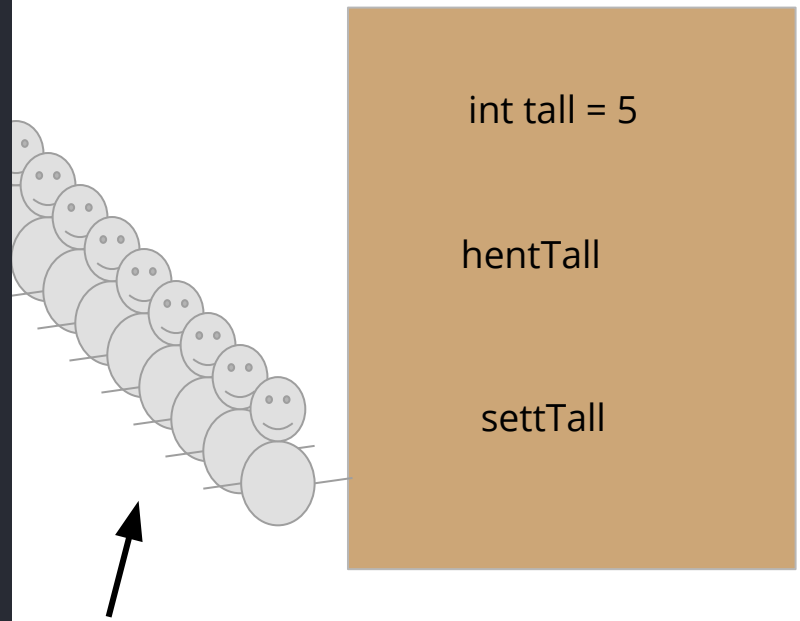
```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
    private Lock laas = new ReentrantLock();
    private int tall;

    public void settTall(int tall){
        laas.lock();
        try {
            this.tall = tall;
        } catch(Exception e) {
            System.out.println("Feil i monitor tallBeholder");
        } finally {
            laas.unlock()
        }
    }
    public int hentTall(){
        laas.lock();
        try {
            return tall;
        } catch(Exception e) {
            System.out.println("Feil i monitor tallBeholder");
        } finally {
            laas.unlock()
        }
```

Forsøk på å illustrere monitor/phone boot/toalett



Bruk monitor design prinsippet

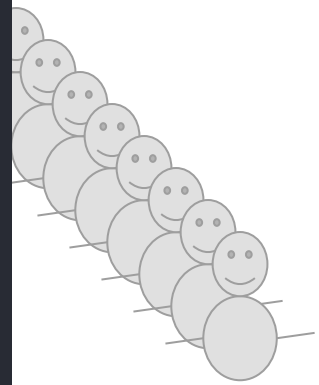
```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```

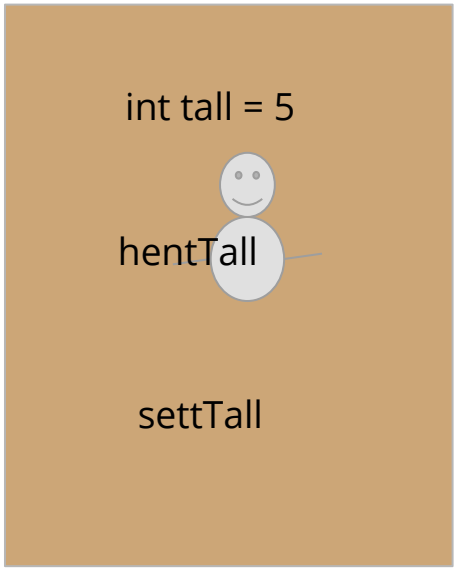


Alle trådene som aksesserer data i monitoren én og én

Bruk monitor design prinsippet

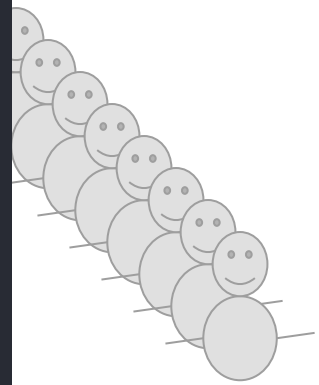
```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```

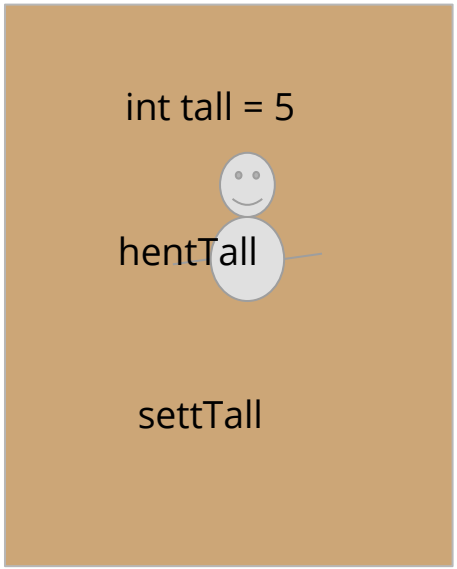




Bruk monitor design prinsippet

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```

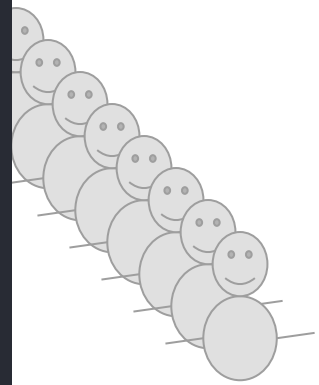


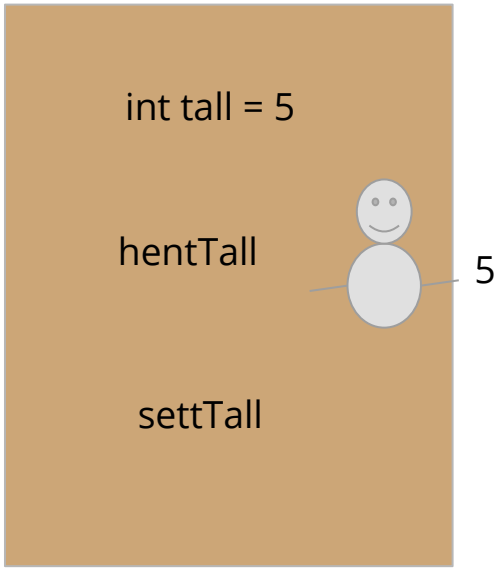


Låst fra noen tar lock() til de tar .unlock()

Bruk monitor design prinsippet

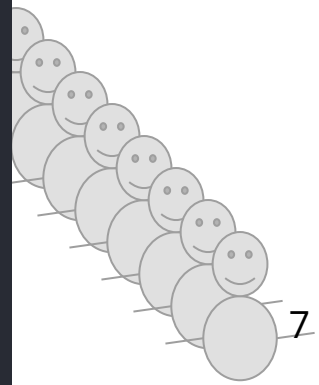
```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```





Bruk monitor design prinsippet

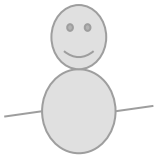
```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```



int tall = 5

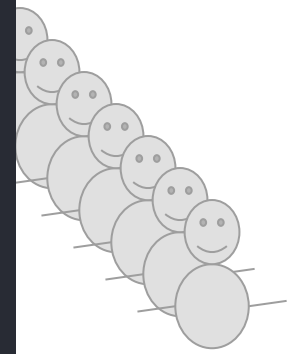
hentTall

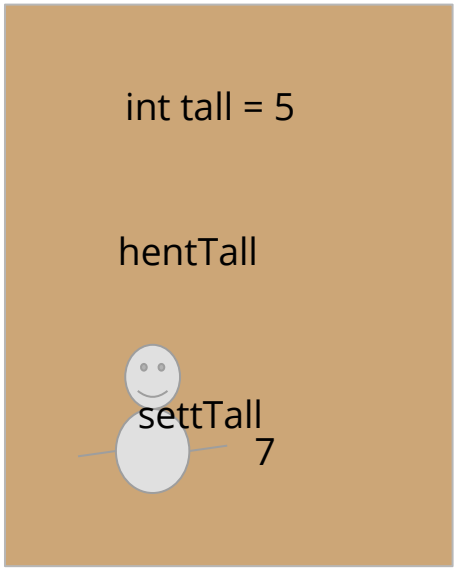
settTall



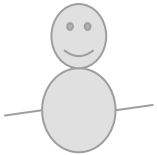
Bruk monitor design prinsippet

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```



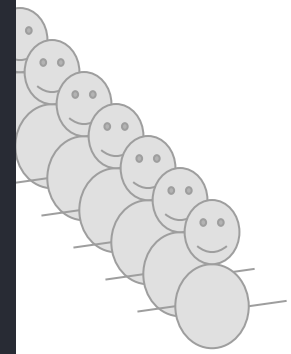


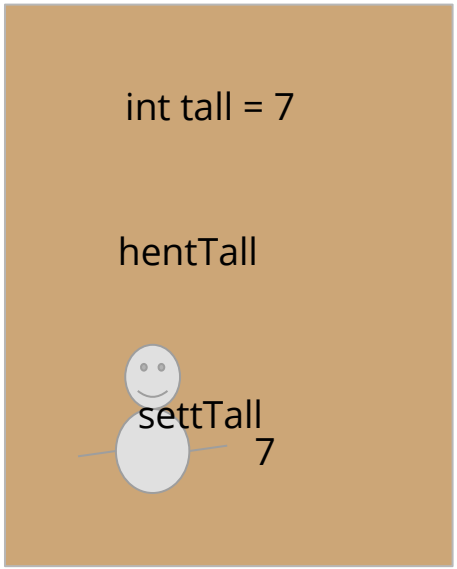
Låst fra noen tar lock() til de tar .unlock()

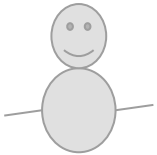


Bruk monitor design prinsippet

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```

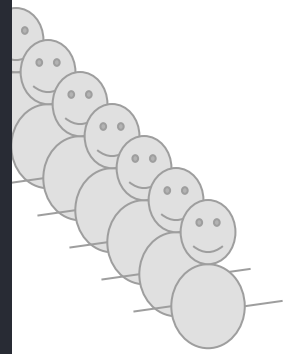






Bruk monitor design prinsippet

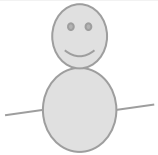
```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class tallBeholder{
  private Lock laas = new ReentrantLock();
  private int tall;

  public void settTall(int tall){
    laas.lock();
    try {
      this.tall = tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
  }
  public int hentTall(){
    laas.lock();
    try {
      return tall;
    } catch(Exception e) {
      System.out.println("Feil i monitor tallBeholder");
    } finally {
      laas.unlock()
    }
```

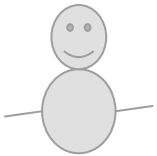


int tall = 7

hentTall

settTall





Vanlige feil

Husk å starte trådene din! traad.start()

Husk å ta unlock() hvis du har tatt lock()

Husk å låse metoder som aksesserer delt data!

Hvis du har await() så husk å putte den i en while(sjekk condition).

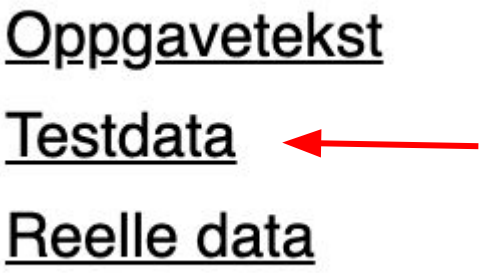
Hvis du har en while(tom){await} så husk å sjekke om vi er ferdige!

Hvis du har while(tom){if(ferdig){return}await]}

Ikke bruk join() rett etter start()!

Bruk testData til å teste!

Det går mye fortere enn å teste med reelle data! Men de er like, testData er bare mindre. Men sjekk med reell data også etterpå!!!!!



Ris, ros, forslag ?

https://nettskjema.no/a/180345

Breakoutrooms

1. Slå på kamera og ha en presentasjonsrunde
2. Jobb sammen med ukesoppgavene, de ligger på emnesiden -> grupper
    a. Enten ved at én deler skjerm eller med codecollab.io/
    b. OBS: codecollab er gratistjeneseter som UiO ikke har avtale med, sannsynligvis vil de samle data om dere. Dere kan fint løse oppgavene uten å bruke den tjenesten!
3. Bruk “ask for help”-knappen for å få hjelp 😊
4. Vi møtes her igjen for å gå gjennom oppgavene til slutt (dere bestemmer hvilke)

Jobbe med oppgaver