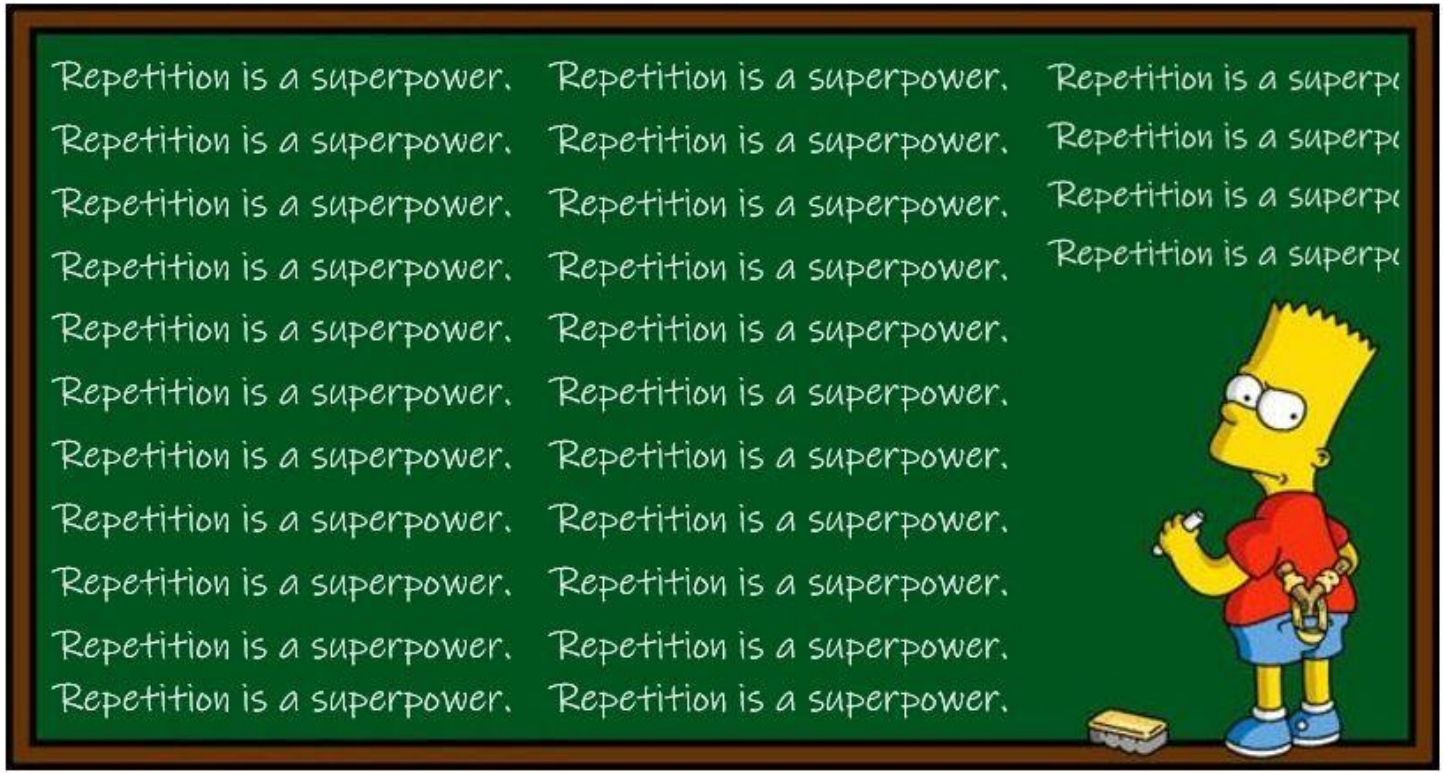
Oppsummering, repetisjon, eksamenstips

IN1010 vår 20205

Stein Gjessing, Dag Langmyhr og Eyvind W. Axelsen



Eksamen: 4. juni 15:00-19:00

Skoleeksamen i Silurveien

Sjekk hvilken sal du skal til i Studentweb

Husk å fylle ut emneevaluering fra Fagutvalget

I dag

- Praktisk info og tips til eksamen
- Gjennomgang/koding av oppgaver fra ordinær eksamen 2024
- Spørsmål og svar

Læringsmål fra emnebeskrivelsen

Hva lærer du? Emnebeskrivelse IN1010 https://www.uio.no/studier/emner/matnat/ifi/IN1010/#learning_outcomes

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 lenkede 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

Pensum

- Følg lenke fra semestersiden til Pensum/litteratur i Leganto

- Alt som er forelest er pensum, se lysark og opptak fra forelesningene
- Alt som har vært i frivillige og obligatoriske oppgaver kan komme på eksamen

- NB! Læringskravet i IN1010 er ikke først og fremst å kunne et pensum, men å kunne skrive gode programmer ved å **bruke programmeringsmetoder som er forelest og innøvd gjennom de forskjellige studieaktivitetene**.

- 💡Har du deltatt på forelesningene, gjort ukesoppgaver og obliger, og forstår dette, så bør du være godt rustet

Praktisk info – bla gjennom disse sidene

- Skoleeksamen i Inspera
  innlogging, brukergrensesnitt, funksjoner, evt håndtegning

- Prøveeksamen er nyttig for å bli kjent (friske opp) Inspera. Test gjerne med et norsk Windows-tastatur
  - Om du bruker Mac til daglig, sjekk at du klarer å skrive { } [ ] etc på et Windows-tastatur
  - Tilgang til prøveeksamen stenger 2 dager før eksamensstart

- Eksamenslokalet i Silurveien
  Oppmøtetid, transport, innganger, saler mm
  NB: T-bane-ombygging!

Bruk ressursene på emnesidene

- Tidligere eksamensoppgaver
  - Ikke alt kommer hvert år, til sammen dekker de det meste av pensum
  - Oppgaver, løsningsforslag og sensorveiledning (hva skal sensor se etter, hva er viktig)
- På eksamenssiden
  - Praktisk info om eksamen
- Programmer fra forelesning
  - Egen mappe på emnesiden

Hva får du på eksamen?

- *Programmeringsoppgaver* som skal besvares i Inspera
  - Inspera er ikke VS Code, ikke regn med autocompletion, kompilering, etc
- *Tegning* på spesialpapir som du får utdelt og som scannes inn (f.eks. datastruktur, klassehierarki)
  - Hvis tegning: Sett av tid til å merke ark riktig
- Det blir ett svarfelt per deloppgave (i motsetning til prøveeksamen)
- Hjelpemidler: Alle trykte og skrevne
  - Husk at du ikke har tid til å lese deg opp i faget på selve eksamen, du må kunne programmere før du møter opp
- ”Trøsterunde” fra faglærer, kommer innom alle rom fra ca en time etter eksamensstart

Hvordan vurderer sensor besvarelsene?

Du må vise forståelse for, og bruk av, sentrale deler av pensum

- Skrivefeil ignoreres vanligvis
  - Men les gjennom det du skriver; feil som kan tyde på manglende forståelse vil gi trekk, og slurv kan gi opphav til misforståelser
- Alle navn på variabler/klasser/metoder vil være på norsk, også i engelsk oversettelse. Bruk disse!
  - Ikke lag egen oversettelse
- Ignoreres (eksempler):
  - Mangler (noen) ';' , '}' eller '{' *om koden ellers gir mening*
  - Småfeil i (gjenkjennbare) navn
- Kan gi trekk (eksempler)
  - Gjennomgående feil/ mangler
  - Bruker beholder på en måte som tyder på manglende forståelse
  - Kode som åpenbart er kopiert fra en annen løsning uten tilpasning
- Det legges ikke vekt på bruk av tilgangsmodifikatorer (public/ private)

Tidsbruk

Pass tiden – alt inkludert **kandidatnummer på eventuelle tegninger** må være klart innen 4 timer

Bruk tid på å

- få oversikt over hele oppgavesettet
- lese teksten nøye for hver oppgave
- forstå datastrukturen før du programmerer (tegn, notér, pseudokode, …)
- vurdere tidsbruk underveis, mot poeng

Lesbar kode kan bidra til at sensor forstår hva du mener selv om det f eks mangler en '{'.

Før eksamen: Hva trenger du å ta med *som du forstår og rekker å finne frem i*

Ikke bruk tid på å

- gjenta oppgaveteksten
- skrive kode du ikke blir bedt om (les oppgaven!)
- unødige detaljer i evt. tegneoppgaver (les oppgaven!)
- kommentere (om du ikke trenger å tolke noe du synes er uklart)
- Med mindre oppgaven ber om det:
  - Test på lovlig input
  - Testprogrammer
  - Import-setninger

Eksamen vår 2024, Oppgave 1 og 2

I denne oppgaven skal du skrive deler av et lite program som behandler fly.

Oppgave 1. Klassehierarki for fly (5 poeng)

Fly representeres av klassen **Fly** som har følgende metoder:

hentId returnerer en String som identifiserer dette flyobjektet, f. eks. «LN-ABC».

hentMotorAntall returnerer et heltall som angir hvor mange motorer flyet har.

hentMTOW returnerer et heltall som angir flyets maksimale startvekt i kilogram som på engelsk heter *Maximum Take Off Weight* forkortet til MTOW.

MTOW, antal motorer (heltall) og deres totale trekkraft (heltall) i kW (kilowatt) og identifikatoren skal være parametere til klassens konstruktør. Flyobjekter skal kunne hektes sammen i en enkeltlenket liste i en senere oppgave, så du skal også ha med instansvariabler i klassen som gjør dette mulig.

I dette eksamenssettet er alt flymateriell enten av klassen **MotorFly** eller klassen **SeilFly**. Alle MotorFly-objekter er enten av klassen **LasteFly** eller klassen **PassasjerFly**.

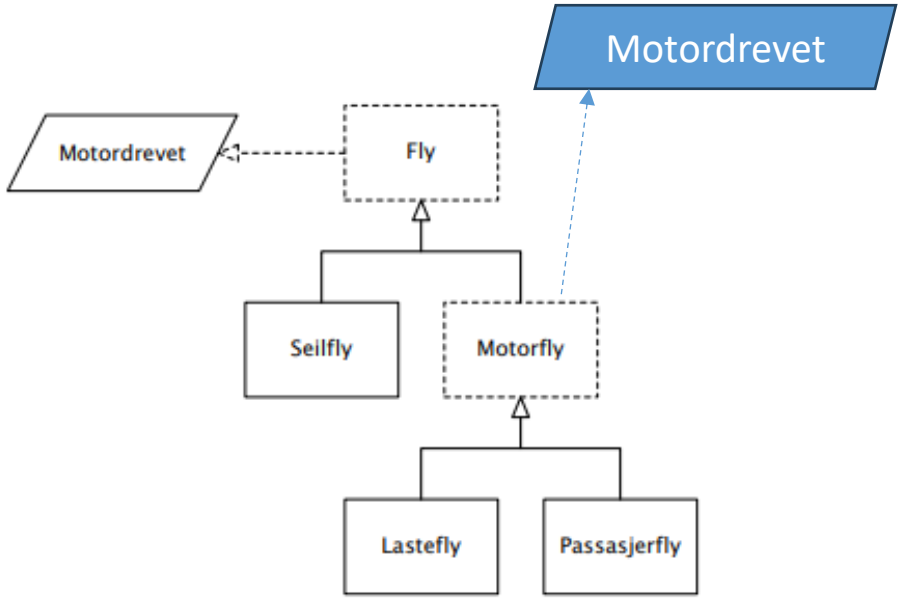
Fly kan implementere et interface **Motordrevet** med følgende grensesnitt:

trekkraft som returnerer maksimal trekkraft av flyets motorer i antall kW som et heltall.

LasteFly har en maks tillatt lastevekt, angitt som et heltall i kilogram, og PassasjerFly har et heltall som representerer maks antall passasjerer det er plass til i flyet. Seilfly har et heltall som representerer minimum synkehastighet målt i cm/s.

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 **Fly, MotorFly, SeilFly, LasteFly, PassasjerFly** og **Motordrevet**.

Det klarer vi lett!



Oppgave 3. Klassen Flyformasjon (28 poeng)

I oppgave 3 skal du skrive en klasse **Flyformasjon** med variabler og metoder. En Flyformasjon er en enkelthektet lenket liste av Flyobjekter og du skal bruke instansvariablene i disse objektene til å lage denne listen. I denne oppgaven kan du aksessere instansvariabler i Fly-klassen og dens subklasser.

3a) 1 poeng

⇨ Skriv konstruktør og instansvariabler for klassen Flyformasjon med peker til første Fly-objekt i listen.

3b) 3 poeng

⇨ Skriv metoden **leggTil** i klassen Flyformasjon som legger til et Flyobjekt først i listen. Parameter til metoden er en referanse til objektet som skal legges til.

3c) 2 poeng

⇨ Skriv metoden **erMed** i klassen Flyformasjon som sjekker om et gitt fly er med i Flyformasjonen. Parameter til metoden er en tekststreng som er flyets identifikasjon. Metoden skal returnere true, hvis flyet er med i Flyformasjonen, og ellers false.

3d) 6 poeng

⇨ Skriv metoden **taUt** i klassen Flyformasjon som tar ut et element i listen. Parameter til metoden er en tekststreng som er identifikasjonen til flyet som skal tas ut. Metoden skal returnere en referanse til det objektet som tas ut. Hvis flyet ikke er med, så skal det returnes NULL.



3e) 8 poeng

⇨ Skriv en iterator over alle Fly-objektene i en Flyformasjon.

3f) 8 poeng

⇨ Skriv metoden **PassasjerFly[] hentPassasjerFly( )** i klassen Flyformasjon som returnerer en array med referanser til alle PassasjerFly-objekter i Flyformasjonen.
Hint: Tell antall passasjerfly før du oppretter arrayen.



Oppgave 4. Maksimum vekt (16 poeng)

4a) 8 poeng

Legg til en metode, **totalVekt,** til Flyformasjon. Metoden skal summere MTOW for alle fly i Flyformasjonen. Metoden skal bruke din iterator fra 3 e).

⇨ Skriv metoden **totalVekt**.

4b) 8 poeng

Legg til en metode, **maksVekt,** til Flyformasjon. Metoden skal returnere den største MTOW blant alle fly i Flyformasjonen. Metoden skal gå gjennom listen av fly rekursivt.

⇨ Skriv metoden **maksVekt** rekursivt.

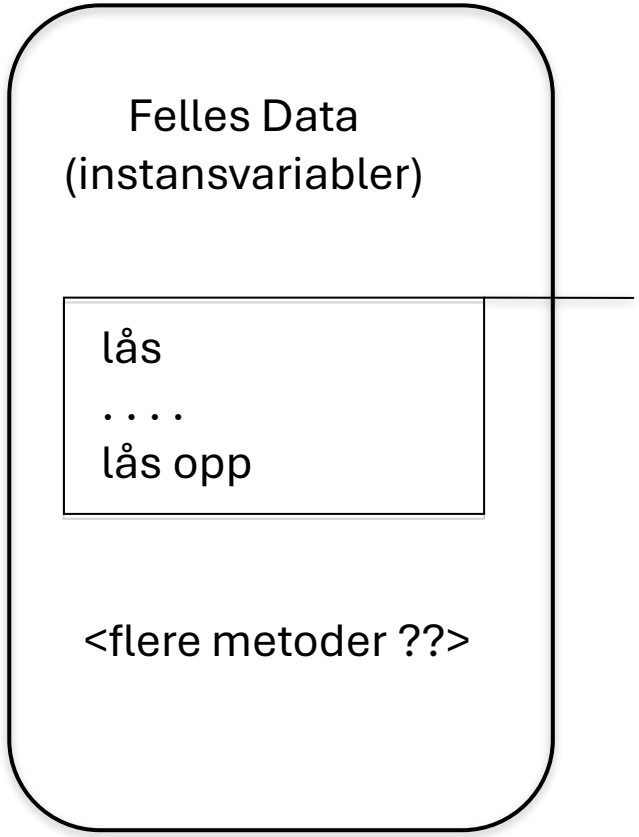


Oppgave 5. Tråder (36 poeng)

I oppgave 5 skal du skrive én trådklasse som heter **Flygeleder** og én som heter **Pilot**. Begge disse skal implementere Runnable. Du skal også skrive en klasse **Rullebane** som skal være en monitor.
(Du skal altså ikke skrive noe fullstendig program)

Alle mors råd: Først: Få en oversikt over oppgaven

I oppgave 5 skal du skrive én trådklasse som heter **Flygeleder** og én som heter **Pilot**. Begge disse skal implementere Runnable. Du skal også skrive en klasse **Rullebane** som skal være en monitor.
(Du skal altså ikke skrive noe fullstendig program



Et Rullebane-objekt
En monitor

Oversikt med tegning:



Et Flygeleder-objekt
En tråd



Et Pilot-objekt
En tråd

I oppgave 5 skal du skrive én trådklasse som heter **Flygeleder** og én som heter **Pilot**. Begge disse skal implementere Runnable. Du skal også skrive en klasse **Rullebane** som skal være en monitor.
(Du skal altså ikke skrive noe fullstendig program

```
class Rullebane {
    . . .
   Lock laas = new ReentrantLock();
   public . . . <metodenavn> () {
        laas.lock();
        try {
        . . .
        } finally { laas.unlock(); }
    }

    <flere metoder ???>
}
```

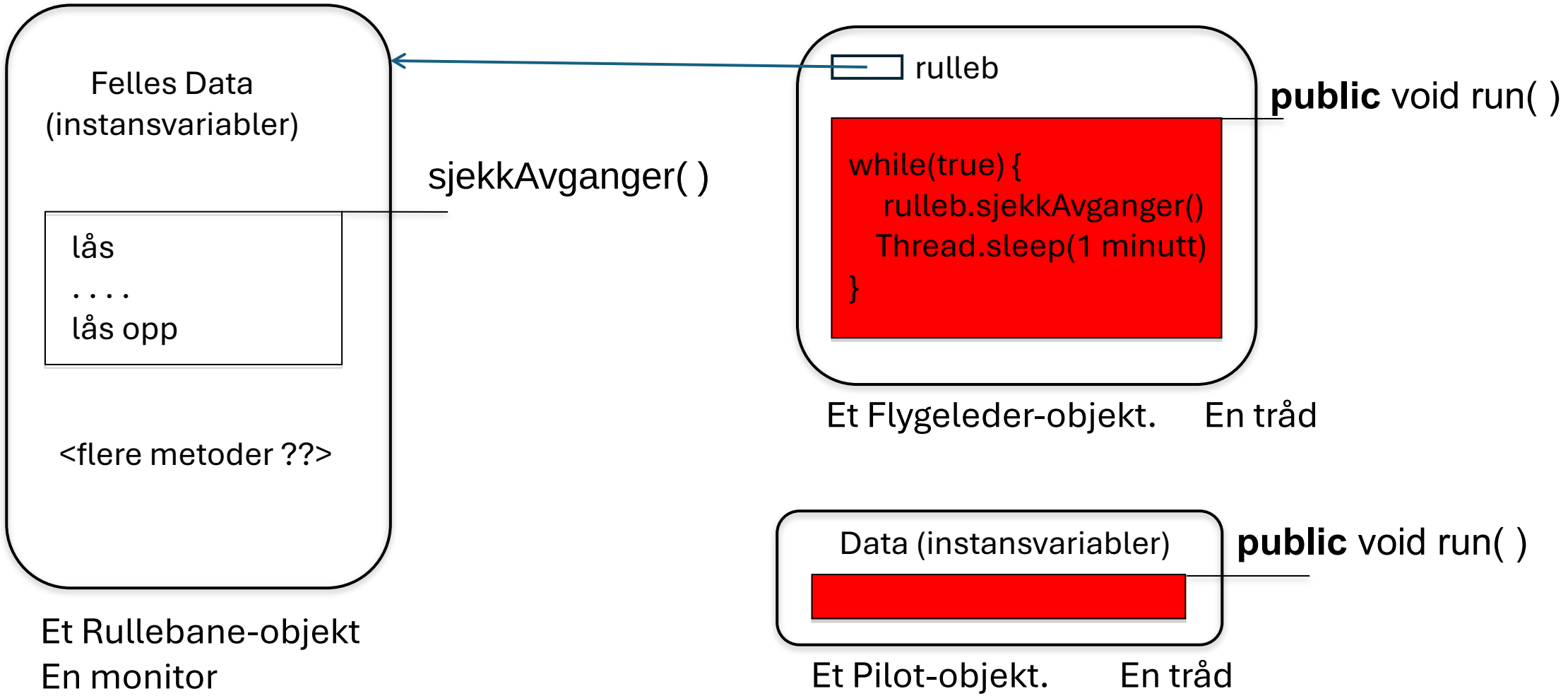
Eller:
Oversikt med kodeskisse:

```
class Flygeleder implements Runnable {
    . . .
    @Override
    public void run () {
        . . .
    }
}
```

```
class Pilot implements Runnable {
    . . .
    @Override
    public void run () {
        . . .
    }
}
```

Oppgave 5 a) 10 poeng

Klassen **Rullebane** skal være en monitor hvor Flygeleder kan se hvilke fly som ønsker å starte og gir tillatelse til fly, så de kan starte. Flygelederen gir starttillatelse til ett fly om gangen ved å kalle **sjekkAvganger**. Etter retur fra **sjekkAvganger** venter Flygelederen 1 minutt (så flyet kan starte og forlate rullebanen) og kaller deretter **sjekkAvganger** igjen for så å gi det neste flyet tillatelse



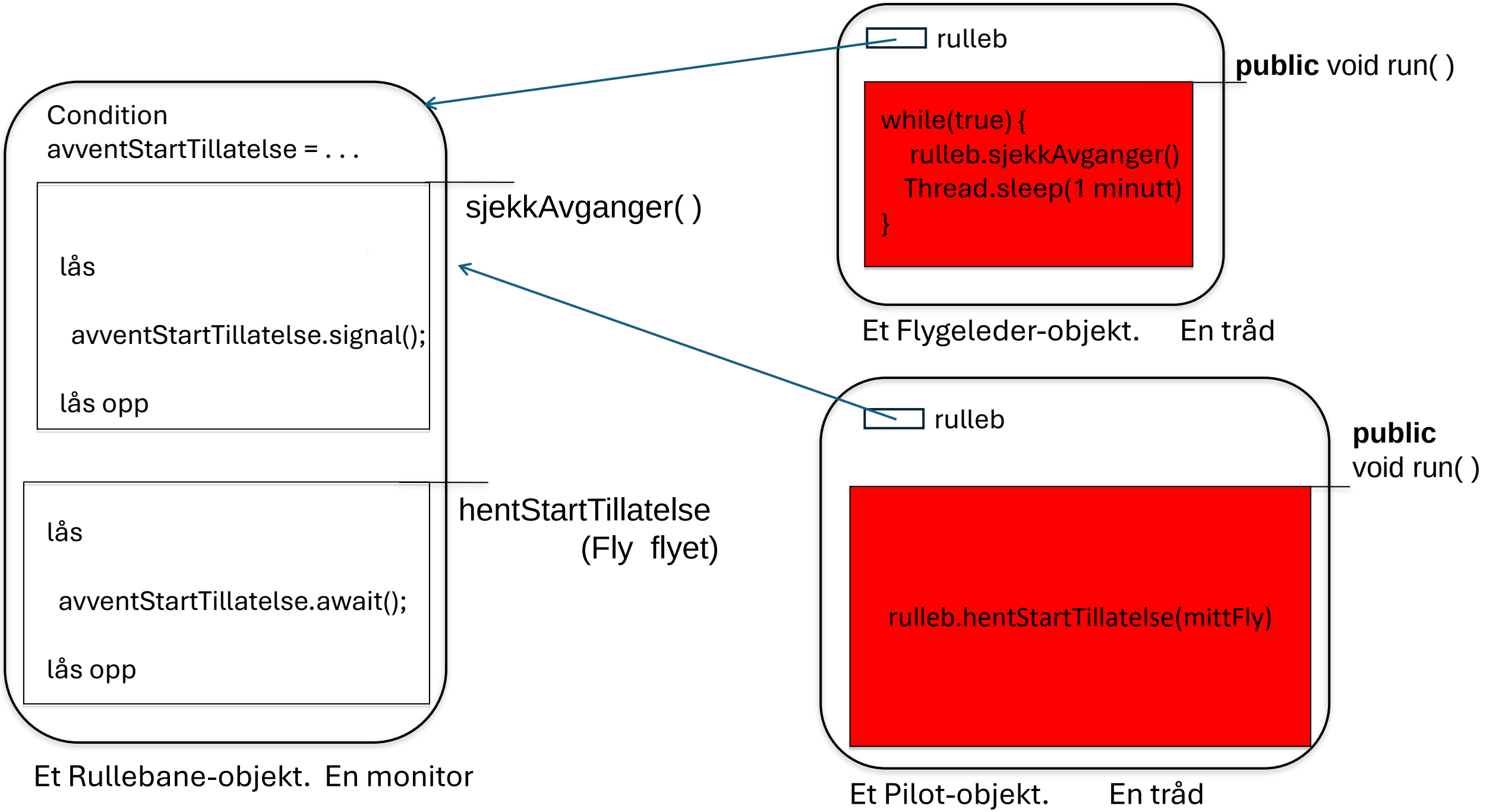
Oppgave 5 a) forts.

Klassen **Rullebane** må ha følgende metoder:

sjekkAvganger som kalles av Flygelederen for å sjekke om der er fly som venter på starttilatelse.
Hvis ikke returner Flygelederen venter 1 minutt og deretter kaller **sjekkAvganger** igjen.
Hvis det er et eller flere fly som venter, så gir flygelederen tillatelse til å starte ved å signalere ett av de ventende flyene via en Condition variabel, **avventStartTillatelse.**
Deretter returner Flygelederen venter 1 minutt og deretter kaller sjekkAvganger igjen.
Metoden har ingen parametere.

hentStartTillatelse som kalles av Pilotene, når de gjerne vil starte. De venter i en Conditionvariabel **avventStartTillatelse**, som Flygelederen signalerer når et fly har fått starttillatelse.
Metoden har én parameter som er et Flyobjekt.

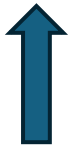
⇨ Skriv monitoren Rullebane, bruk Condition og ReentrantLock; ikke bruk synchronized.


rulleb
public void run( )
while(true) {
rulleb.sjekkAvganger()
Thread.sleep(1 minutt)
}
Et Flygeleder-objekt. En tråd
Condition
avventStartTillatelse = . . .
sjekkAvganger( )
lås
avventStartTillatelse.signal();
lås opp
hentStartTillatelse
(Fly flyet)
lås
avventStartTillatelse.await();
lås opp
Et Rullebane-objekt. En monitor
rulleb
public
void run( )
rulleb.hentStartTillatelse(mittFly)
Et Pilot-objekt. En tråd

```
import java.util.concurrent.locks.*;    // Ved eksamen: Ikke farlig å glemme
class Rullebane {
    private int antVentendeFly = 0;
    private Lock laas = new ReentrantLock();
    årivate Condition avventStartTillatelse = laas.newCondition();
    public void sjekkAvganger () {
        laas.lock();
        try {
            if (antVentendeFly == 0) return;
            avventStartTillatelse.signal(); // skal bare starte ett fly!
            antVentendeFly--;
        } finally { laas.unlock(); }
    }
    public void hentStartTillatelse (Fly flyet) {
        laas.lock();
        try {
            antVentendeFly++;
            avventStartTillatelse.await();
         } catch (InterruptedException e) {
            return;   // Ved eksamen:  Ikke viktig hva - etter avbrudd av tråd
         } finally { laas.unlock(); }
    }
  }
```

5 b) 7 poeng. ⇨ Skriv trådklassen Flygeleder, som beskrevet i 5 a).

```
class Flygeleder implements Runnable {
    private Rullebane rulleb;
    Flygeleder (Rullebane r) {
        rulleb = r;
    }
    @Override
    public void run () {
        while (true) {
            rulleb.sjekkAvganger();
            try {
                Thread.sleep(60 * 1000); /* = 1 minutt */
            } catch (InterruptedException e) { return; }
        }
    }
}
```



I det virkelige liv er det viktig hva du gjør ved unntak.
Ved eksamen må du huske å få med **`InterruptedException`**,
men det er ikke viktig hva du så gjør.

5 c) 7 poeng

⇨ Skriv trådklassen **Pilot**, som beskrevet i 5 a).
Parameter til konstruktøren til Pilot er en referanse til objekt. Antag at en pilot kun har ETT fly.

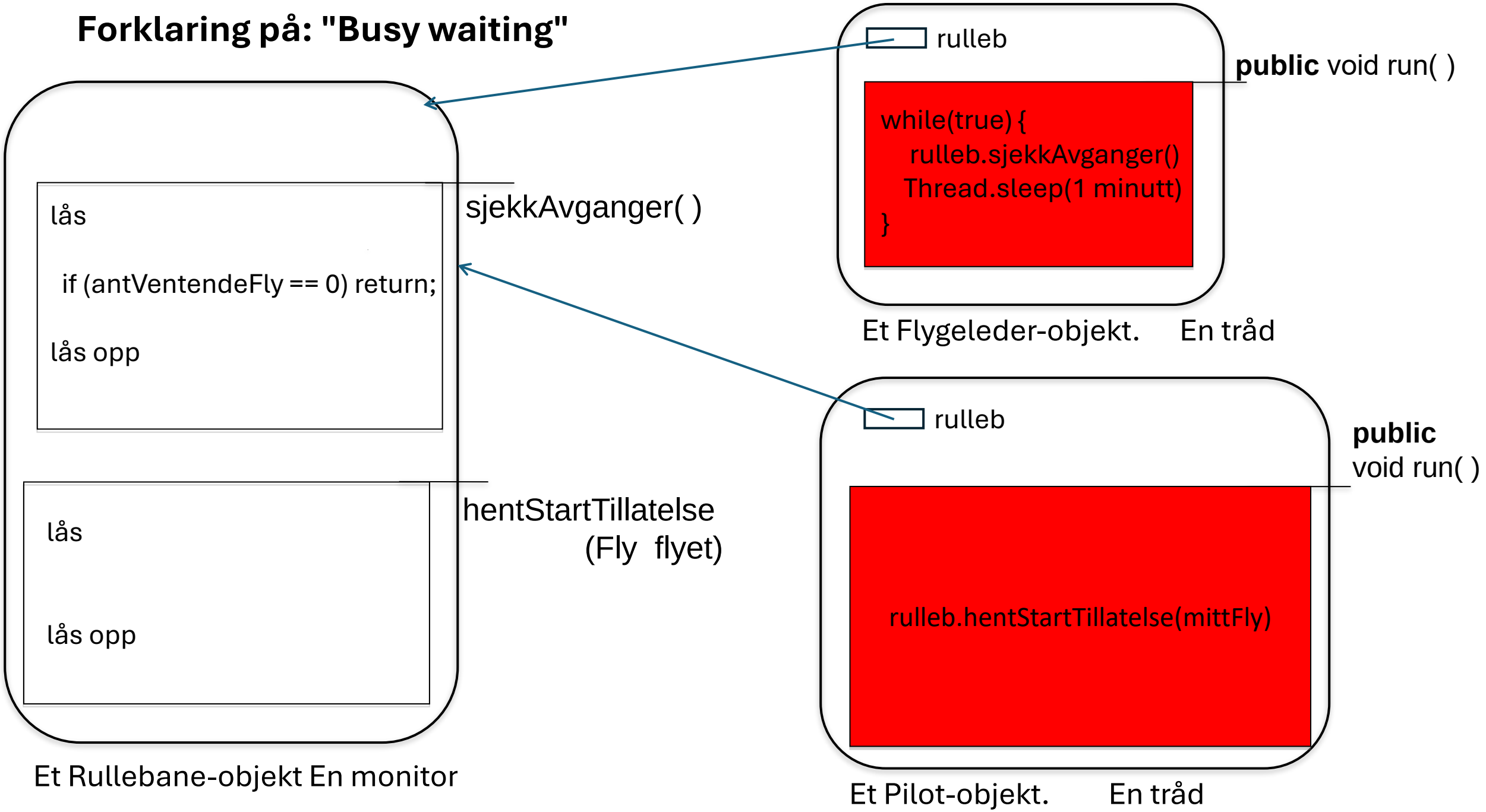
```
class Pilot implements Runnable {
    private Fly mittFly;
    private Rullebane rulleb;
    Pilot (Rullebane r, Fly f) {
        rulleb = r; mittFly = f;
    }
    @Override
    public void run () {
        rulleb.hentStartTillatelse(mittFly);
    }
}
```

5 d) 12 poeng

Flygelederen bruker «busy waiting» og kaller **sjekkAvganger** hvert minutt.
Modifiser **sjekkAvganger** så HVIS det ikke er et fly som venter på starttillatelse, så skal Flygelederen vente på en Condition **avventFly**.
Denne signaleres så av Piloter når de gjerne vil ha starttillatelse.
Modifiser **hentStartTillatelse** så Piloterne signalerer flygelederen, som så gir tilllatelse.

⇨ Omskriv **sjekkAvganger** og **hentStartTillatelse** som beskrevet ovenfor.

Forklaring på: "Busy waiting"



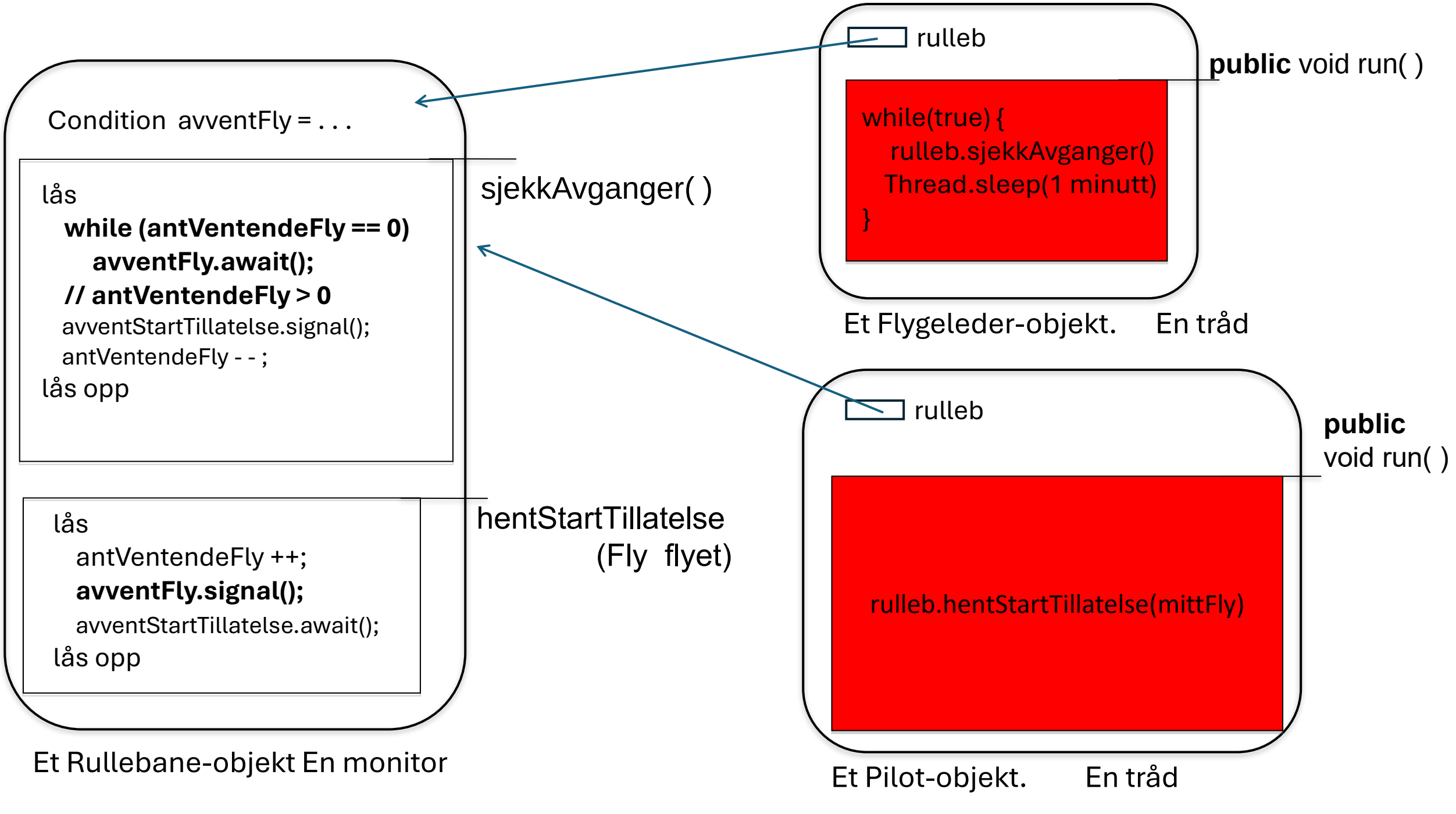
5 d) forts.

Modifiser **sjekkAvganger** så HVIS det ikke er et fly som venter på starttillatelse, så skal Flygelederen vente på en Condition **avventFly**.

Denne signaleres så av Piloter når de gjerne vil ha starttillatelse.

Modifiser **hentStartTillatelse** så Piloterne signalerer flygelederen, som så gir tilllatelse.

⇨ Omskriv **sjekkAvganger** og **hentStartTillatelse** som beskrevet ovenfor.


rulleb
public void run( )
while(true) {
rulleb.sjekkAvganger()
Thread.sleep(1 minutt)
}
Et Flygeleder-objekt. En tråd
Condition avventFly = . . .
lås
while (antVentendeFly == 0)
avventFly.await();
// antVentendeFly > 0
avventStartTillatelse.signal();
antVentendeFly - - ;
lås opp
sjekkAvganger( )
lås
antVentendeFly ++;
avventFly.signal();
avventStartTillatelse.await();
lås opp
hentStartTillatelse
(Fly flyet)
Et Rullebane-objekt En monitor
rulleb
public
void run( )
rulleb.hentStartTillatelse(mittFly)
Et Pilot-objekt. En tråd

Svar på oppgave 5 d)

```
import java.concurrent.locks.*;
class Rullebane5d {
  int antVentendeFly = 0;
  Lock laas = new ReentrantLock();
  Condition avventStartTillatelse =
                    laas.newCondition();
  // én ny deklarasjon i 5d
  Condition avventFly =
                    laas.newCondition();

  public void sjekkAvganger () {
     laas.lock();
     try {
        // én ny linje i 5d
        while (antVentendeFly == 0)
            avVentFly.await();
        // antVentendeFly > 0
        avventStartTillatelse.signal();
        antVentendeFly--;
     } finally { laas.unlock(); }
  }
```

```
public void
hentStartTillatelse (Fly f) {
  laas.lock();
  try {
     antVentendeFly++;
     // én ny linje i 5d:
     avventFly.signalAll();
     avventStartTillatelse.await();
  } catch (InterruptedException e){
     return;
  } finally { laas.unlock(); }
 }
}
```

Alternativ måte å svare på oppgave 5 d)

I klassen Rullebane skal denne deklarasjonen legges til:

```
Condition avventFly = laas.newCondition();
```

I metoden sjekkAvganger skal denne ene linjen legges inn helt først i try-blokken:

```
while (antVentendeFly == 0)  avVentFly.await();
```

I metoden hentStartTillattelse skal denne linjen legges inn som andre linje i try-blokken, rett etter **`antVentendeFly++`**:

```
avventFly.signalAll();    // eller bare avventFly,signal();
```

Og enda en gang: Husk

- Du skal "selge" besvarelsen din til sensor.

- Er svarene dine oversiktlige er det lett for sensor å skjønne programmet ditt og hva du mener

- Er svaret ikke oversiktlig, prøver sensor å forstå hva du har ment, men sensor kan da misforstå / ikke skjønne -> Dårligere karakter

Takk for nå, lykke til!

