

## Ukeoppgaver 17 – PowerShell, Java-threads, synkronisering og deadlock

RØD - Obligoppgaver

GUL – Ikke obligoppgaver

TURKIS – Ukens nøtt og utfordringer

Teorioppgaver besvares ved hjelp av hjelpemdirler.

### 1. (Oblig)

Koden til NosynchThread.java i vim



The screenshot shows a terminal window titled "amnadarastgir" with the command "ssh group25@os25.vlab.cs.oslomet.no" and a size of "119x58". The code displayed is:

```
class SaldoThread extends Thread
{
    static int MAX = 1000000; // En million
    static int count = 0;
    public static int saldo; // Felles variable, gir race condition
    int id;

    SaldoThread()
    {
        count++;
        id = count;
    }

    public void run()
    {
        System.out.println("Tråd nr. " + id + ", med prioritet " + getPriority() + " starter");
        updateSaldo();
    }

    public synchronized void updateSaldo()
    {
        int i;
        if(id == 1)
        {
            for(i = 1;i < MAX;i++)
            {
                saldo++;
            }
        }
        else
        {
            for(i = 1;i < MAX;i++)
            {
                saldo--;
            }
        }
        System.out.println("Tråd nr. " + id + " ferdig. Saldo: " + saldo);
    }
}

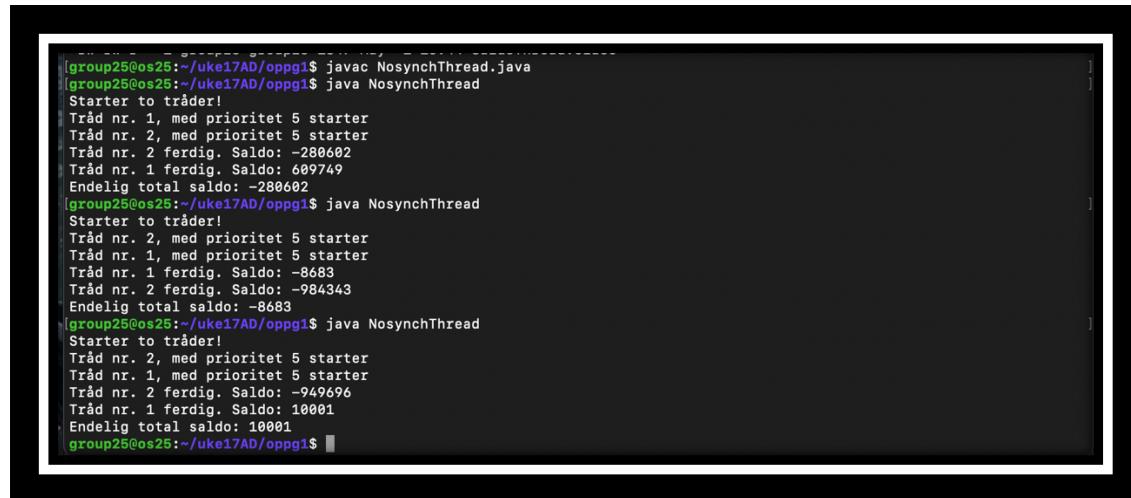
class NosynchThread extends Thread
{
    public static void main (String args[])
    {
        int i;
        System.out.println("Starter to tråder!");

        SaldoThread s1 = new SaldoThread();
        SaldoThread s2 = new SaldoThread();
        s1.start();
        s2.start();

        try{s1.join();} catch (InterruptedException e){}
        try{s2.join();} catch (InterruptedException e){}

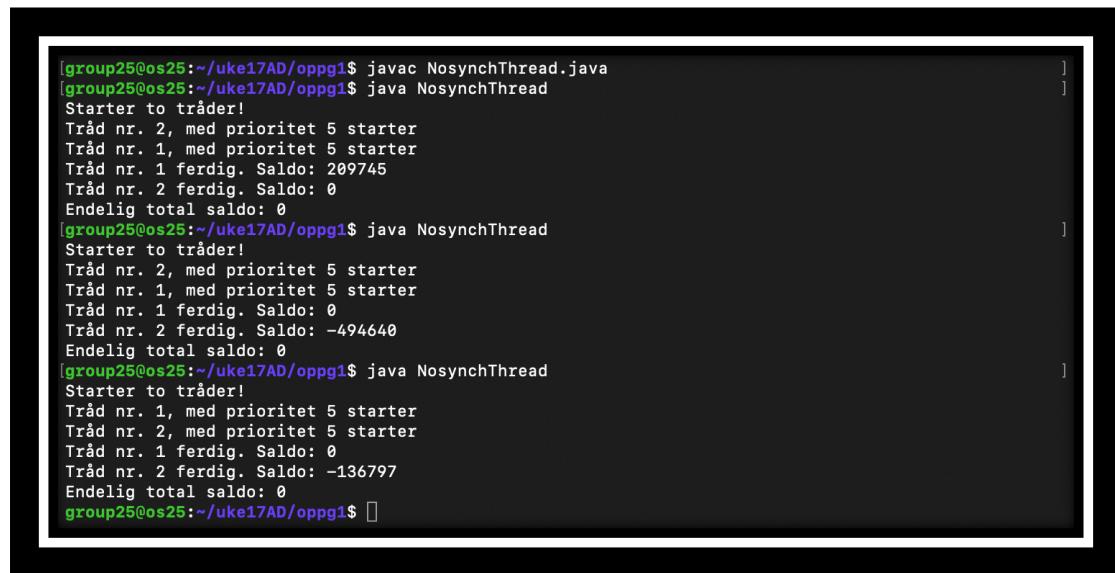
        System.out.println("Endelig total saldo: " +saldoThread.saldo);
    }
}
```

Kompilerer og kjører. For hver kjøring kan vi se at vi får et ulikt resultat. Dette er fordi trådene ikke er synkroniserte.



```
[group25@os25:~/uke17AD/oppg1$ javac NosynchThread.java
[group25@os25:~/uke17AD/oppg1$ java NosynchThread
Starter to tråder!
Tråd nr. 1, med prioritet 5 starter
Tråd nr. 2, med prioritet 5 starter
Tråd nr. 2 ferdig. Saldo: -288602
Tråd nr. 1 ferdig. Saldo: 609749
Endelig total saldo: -288602
[group25@os25:~/uke17AD/oppg1$ java NosynchThread
Starter to tråder!
Tråd nr. 2, med prioritet 5 starter
Tråd nr. 1, med prioritet 5 starter
Tråd nr. 1 ferdig. Saldo: -8683
Tråd nr. 2 ferdig. Saldo: -984343
Endelig total saldo: -8683
[group25@os25:~/uke17AD/oppg1$ java NosynchThread
Starter to tråder!
Tråd nr. 2, med prioritet 5 starter
Tråd nr. 1, med prioritet 5 starter
Tråd nr. 2 ferdig. Saldo: -949696
Tråd nr. 1 ferdig. Saldo: 10001
Endelig total saldo: 10001
group25@os25:~/uke17AD/oppg1$ ]
```

Nå ser vi at saldoen alltid blir 0 så det vil si at trådene er synkroniserte.



```
[group25@os25:~/uke17AD/oppg1$ javac NosynchThread.java
[group25@os25:~/uke17AD/oppg1$ java NosynchThread
Starter to tråder!
Tråd nr. 2, med prioritet 5 starter
Tråd nr. 1, med prioritet 5 starter
Tråd nr. 1 ferdig. Saldo: 209745
Tråd nr. 2 ferdig. Saldo: 0
Endelig total saldo: 0
[group25@os25:~/uke17AD/oppg1$ java NosynchThread
Starter to tråder!
Tråd nr. 2, med prioritet 5 starter
Tråd nr. 1, med prioritet 5 starter
Tråd nr. 1 ferdig. Saldo: 0
Tråd nr. 2 ferdig. Saldo: -494640
Endelig total saldo: 0
[group25@os25:~/uke17AD/oppg1$ java NosynchThread
Starter to tråder!
Tråd nr. 1, med prioritet 5 starter
Tråd nr. 2, med prioritet 5 starter
Tråd nr. 1 ferdig. Saldo: 0
Tråd nr. 2 ferdig. Saldo: -136797
Endelig total saldo: 0
group25@os25:~/uke17AD/oppg1$ ]
```

Hvis det tar lengre tid er det fordi når en tråd jobber med et kritisk avsnitt, altså en felles variabel, så må alle andre tråder vente til den som bruker tråden er ferdig.

```
import java.lang.Thread;
class SaldoThread extends Thread
{
    static int MAX = 1000000; // En million
    static int count = 0;
    public static int saldo; // Felles variable, gir race condition
    int id;

    SaldoThread()
    {
        count++;
        id = count;
    }

    public void run()
    {
        System.out.println("Tråd nr. " + id + ", med prioritet " + getPriority() + " starter");
        updateSaldo();
    }

    public synchronized void updateSaldo()
    {
        int i;
        if(id == 1)
        {
            for(i = 1;i < MAX;i++)
            {
                synchronized (NosynchThread.lock){
                    saldo++;
                }
            }
        }
        else
        {
            for(i = 1;i < MAX;i++)
            {
                synchronized (NosynchThread.lock){
                    saldo--;
                }
            }
        }
        System.out.println("Tråd nr. " + id + " ferdig. Saldo: " + saldo);
    }
}

class NosynchThread extends Thread
{
    public static Object lock = new Object(); //LOCK OBJEKT

    public static void main (String args[])
    {
        int i;
        System.out.println("Starter to tråder!");

        SaldoThread s1 = new SaldoThread();
        SaldoThread s2 = new SaldoThread();
        s1.start();
        s2.start();

        try{s1.join();} catch (InterruptedException e){}
        try{s2.join();} catch (InterruptedException e){}
        System.out.println("Endelig total saldo: " +SaldoThread.saldo);
    }
}
```

## 2. (UKENS NØTT NR 1!)

Koden viser synkronisering av tråder gjennom bruk av synchronized-metoder.

Metodene upSaldo() og downSaldo() er merket som synchronized, og de oppdaterer variablen saldo som er felles for begge trådene.

Hvis det viser seg at synkroniseringen ikke fungerer, kan det være fordi det er en mulighet for at begge trådene prøver å oppdatere saldo-variablen samtidig, og at synkroniseringen ikke virker som forventet. En annen mulighet er at koden kjører for kort tid til at feilen kan vises, og at det derfor trengs flere runder i løkken eller flere gjentatte kjøringer av programmet for å vise feilen.

En måte å sikre synkronisering på, er å bruke en Lock-objekt i stedet for synchronized-metoder. Lock-objekter gir mer fleksibilitet og kontroll over synkroniseringen, og kan være mer effektive i noen situasjoner.

### 3. (UKENS NØTT NR 2!)

Problemet med denne algoritmen er at den ikke tar hensyn til at flere prosesser kan forsøke å ta mutex samtidig. Dersom begge prosessene forsøker å ta mutex samtidig, vil begge havne i en uendelig løkke i GetMutex-metoden, siden de begge venter på at turn skal bli lik deres egen verdi. Dette kalles en deadlock-situasjon, og det vil hindre begge prosessene i å komme inn i det kritiske avsnittet, selv om bare én av dem skal være der.

En mulig løsning på dette problemet er å legge til et ekstra signal slik at en prosess kan varsle en annen prosess om at den skal vente før den tar mutex. Dette kan gjøres ved å bruke en variabel som angir om en prosess ønsker å ta mutex, og hvis begge prosessene ønsker å ta mutex, kan man bruke en annen variabel for å avgjøre hvilken av dem som får lov til å ta mutex først. Dette vil sikre at bare én prosess tar mutex om gangen og hindre deadlock.

## DINING PHILOSOPHERS PROBLEM?

Oppgaven foreslår en algoritme for å implementere en mutex (mutual exclusion), som er en teknikk for å sikre at kun én prosess (eller tråd) kan aksessere en ressurs eller et kritisk avsnitt av koden om gangen. Algoritmen bruker en delt variabel **turn**, som er enten 0 eller 1, og to funksjoner: **GetMutex** og **ReleaseMutex**.

**GetMutex(t)** funksjonen sjekker den delte variabelen **turn** og venter i en løkke til **turn** er lik **t**, der **t** er verdien som tilhører prosessen som prøver å få tilgang til den beskyttede ressursen.

**ReleaseMutex(t)** funksjonen bytter verdien av **turn** til den andre prosessen sin verdi (**1-t**), slik at den andre prosessen nå kan få tilgang til den beskyttede ressursen.

Algoritmen implementerer en form for vekselvis aksess til ressursen, ved at hver prosess venter på at den andre prosessen skal fullføre sitt kritiske avsnitt før de prøver å aksessere ressursen selv.

Men selv om denne algoritmen kan virke korrekt ved første øyekast, kan det oppstå en situasjon kjent som "bryter-lås" (deadlock) når begge prosessene prøver å få tak i ressursen samtidig. Det skjer hvis **GetMutex** funksjonen kalles samtidig av begge prosessene, men først den ene og så den andre funksjonen. Da vil begge prosessene vente uendelig lenge på at **turn** skal få verdien sin, og det oppstår en evigvarende venteloop som hindrer videre fremgang i programmet.

### 4. (UKENS NØTT NR 3!)

```

static boolean[] flag = new boolean[2]; // Begge false i utgangspunktet
GetMutex(int t)
{
    int other;
    other = 1 - t;
    flag[t] = true; // Ønsker å gå inn i kritisk avsnitt
    while (flag[other] == true){}
}

ReleaseMutex(int t)
{
    flag[t] = false;
}

```

## DINING PHILOSOPHERS PROBLEM?

denne algoritmen sikrer gjensidig utelukkelse. Når en prosess ønsker å gå inn i det kritiske avsnittet, setter den flagget for sin egen prosess til true og sjekker om flagget for den andre prosessen er true i en loop. Hvis det er tilfelle, vil prosessen fortsette å vente i løkken til den andre prosessen har fullført sitt kritiske avsnitt og satt flagget sitt til false.

Det som likevel gjør denne algoritmen ubrukelig, er at den kan føre til såkalt "starvation". Hvis en av prosessene fortsetter å prøve å få tak i mutexen, men alltid blir blokkert av den andre prosessen, kan den ende opp med å aldri få tilgang til det kritiske avsnittet. Dette kan skje fordi det ikke er noen garanti for at flaggene skifter tilstand mellom hver prosess sin tur til å utføre sitt kritiske avsnitt. Dette betyr at den andre prosessen kan fortsette å ta tak i mutexen først og blokkere den første prosessen, selv om begge prosessene prøver å få tak i mutexen.

### 5. (UKENS NØTT NR 4!)

Til denne oppgaven brukte jeg hjelpebidrager med nettverk

5. Ukens nøtt nr. 4: Peterson-algoritmen (se Tanenbaum) er en meget elegant softwareløsning av mutex-problemet. Det fantes løsninger før den ble funnet i 1981, men de var meget kompliserte. Den er en blanding av forsøk 2 og 3 og ser slik ut:

```

static boolean[] flag = new boolean[2]; // Begge false i utgangspunktet
static int turn = 0;

GetMutex(int t)
{
    int other;
    other = 1 - t;
    flag[t] = true; // Ønsker å gå inn i kritisk avsnitt
    turn = other; // Viktig
    while (flag[other] == true && turn == other){}
}

ReleaseMutex(int t)
{
    flag[t] = false;
}

```

Overbevis deg selv om hvordan og hvorfor den alltid virker, uansett når en Context Switch intreffer.

Petersons algoritme er en software-løsning på mutex-problemet som ble funnet av Gary L. Peterson i 1981. Den er en av de mest brukte algoritmene for å sikre gjensidig utelukkelse mellom prosesser i et parallelt system.

Algoritmen bruker to boolske flagg og en heltallsvariabel som kalles turn. Flaggene indikerer om en prosess ønsker å gå inn i det kritiske avsnittet, og turn indikerer hvem som skal få tilgang til det kritiske avsnittet.

Når en prosess ønsker å gå inn i det kritiske avsnittet, setter den sitt eget flagg til true og bytter turn-variabelen med den andre prosessen. Deretter sjekker den om den andre prosessen ønsker å gå inn i det kritiske avsnittet, og om turn-variabelen er satt til den andre prosessen. Hvis dette er tilfelle, må prosessen vente til den andre prosessen har gått ut av det kritiske avsnittet.

Når en prosess går ut av det kritiske avsnittet, setter den sitt eget flagg til false.

Algoritmen fungerer ved å sikre at bare en prosess kan være i det kritiske avsnittet om gangen. Selv om et kontekstbytte skjer mellom to prosesser som ønsker å gå inn i det kritiske avsnittet, vil algoritmen fortsatt sikre gjensidig utelukkelse.

Det er flere måter å bevise at Petersons algoritme fungerer, men en vanlig metode er å analysere alle mulige kombinasjoner av trådsekvenser og vise at bare en prosess kan være i det kritiske avsnittet om gangen.

#### 6. (Ikke oblig)

Ja, Petersons mutex-løsning vil fortsatt fungere når prosess-schedulingen er preemptive. Selv om schedulingen kan bytte mellom prosessene på vilkårlige tidspunkter, vil algoritmen sikre at bare en prosess går inn i det kritiske avsnittet av gangen.

Når det gjelder ikke-preemptive scheduling, vil Petersons algoritme også fungere. I en ikke-preemptive scheduleringsmodell vil en prosess alltid kjøre til den enten blokkeres på en I/O-operasjon eller frivillig gir opp CPU-tiden sin ved å kalle en "yield"-funksjon. I dette scenariet vil Petersons algoritme sørge for at bare en prosess går inn i det kritiske avsnittet, selv om schedulingen ikke bytter mellom prosessene så ofte.

#### 7. (Ikke oblig)

## KAN IKKE GJØRE DEN:

Opprett to nye brukere med brukernavn klara(fullt navn: Klara Klok) og bjarne(fullt navn: Bjarne Berntsen) på Windows Virtualbox VM'en (eventuelt en annen Windows-PC) og gi dem passord. Brukerne skal ha begrensede rettigheter(vanlige brukere). Opprett i tillegg en gruppe med navn kbgruppen som kun de to er medlemmer av.

## 8. (Ikke oblig)

## KAN IKKE GJØRE DEN

Logg ut og inn igjen som brukeren klara. Start notepad og lag en fil klara.txt. Skriv inn en linje tekst i filen. Hvor i filsystemet ligger denne filen? Logg så ut og inn som brukeren bjarne. Kan han lese innholdet av filen klara.txt? Kan han endre på innholdet i denne tekstfilen? Er det mulig å sette rettigheter slik at bare disse to brukerne får skrive og leserettigheter til denne filen? Forklar hvordan eller eventuelt hvorfor det ikke går. Hvor kan man legge filer som skal deles av alle brukere?

## 9. (Ikke oblig)

## 10. (Ikke oblig)

## 11. (Ikke oblig)

## 12. (Ikke oblig)

## 13. (Oblig)

Under lager jeg en fil som heter hello.ps1. Jeg kunne også bare ha skrevet 'touch hello.ps1'

```
[PS /Users/amnadastgir> New-Item -ItemType File -name hello.ps1
Directory: /Users/amnadastgir

UnixMode      User        Group          LastWriteTime          Size
-----      ----        ----          -----          --
-rw-r--r--  amnadastgir    staff  09.05.2023 11:21           0
```

Under åpner jeg tekstredigeringsverktøyet 'code' som tok meg med til visual studio code. Jeg måtte først inn i vs-code og under view laste ned cmdlet-en med

Shell Command: Install 'code' command in PATH

```
[PS /Users/amnadastgir> code hello.ps1
[PS /Users/amnadastgir> ./hello.ps1
Hello World!
```

Også kunne jeg kjøre programmet slikt vist over. Kan også hente innholdet med Get-Content

```
Hello World!
[PS /Users/amnadastgir> Get-Content hello.ps1
Write-Host "Hello World!"
```

#### 14. UKENS NØTT!

#### 15. (Oblig)

Under har jeg laget og kjørt i en Linux-terminal:

```
[amnadastgir@Amnas-MacBook-Air Ukesoppgaver % vim oppg15uke17
[amnadastgir@Amnas-MacBook-Air Ukesoppgaver % chmod 700 oppg15uke17
[amnadastgir@Amnas-MacBook-Air Ukesoppgaver % ./oppg15uke17
kopi.txt
```

Forsøker det samme i pwsh kommandolinja i min Mac terminal. Får ikke til kjøringen i terminalen.

```
[PS /Users/amnadastgir/Desktop/uke17> ./oppg15
```

Bare vindu dukker opp:

```
mkdir dir
cd dir
echo hei > fil.txt
cp fil.txt lik.txt
mv fil.txt ..
cp lik.txt kopi.txt
rm lik.txt
ls
```

#### 16. (Oblig)

Denne kommandoen får ikke jeg kjørt ettersom jeg bare har en PowerShell kommandolinje på min MAC.

```
[PS /Users/amnadastgir> PSVersionTable
PSVersionTable: The term 'PSVersionTable' is not recognized as a name of a cmdlet, function, script file, or executable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
[PS /Users/amnadastgir> PSVersionTable.VersionTable
PSVersionTable.VersionTable: The term 'PSVersionTable.VersionTable' is not recognized as a name of a cmdlet, function, script file, or executable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
```

**17. (Ikke oblig)****18. (Oblig)**

Under lagde jeg filen selv fordi skriptet ikke fungerte på meg.

```
[PS /Users/amnadastgir> touch fil.txt  
[PS /Users/amnadastgir> (Get-ChildItem .\fil.txt).CreationTime  
tirsdag 9. mai 2023 12:16:40
```

**19. (Oblig)**

Dette kan ikke jeg gjøre.

```
[Get-Item: Cannot find drive. A drive with the name 'C' does not exist.  
[PS /Users/amnadastgir> (Get-Item C:\Windows\system.ini).CreationTime.Year  
Get-Item: Cannot find drive. A drive with the name 'C' does not exist.  
PS /Users/amnadastgir>
```

**20. (Ikke oblig)****21. (Oblig)**

Jeg har ikke Windows maskin eller VM, så det ser slikt ut:

```
[Get-Member: You must specify an object for the Get-Member cmdlet.  
PS /Users/amnadastgir> Get-Process idle | Get-Member  
Get-Process: Cannot find a process with the name "idle". Verify the process name and call the cmdlet again.  
Get-Member: You must specify an object for the Get-Member cmdlet.  
PS /Users/amnadastgir>
```

Også henter man verdien til id slik, altså ved å bruke den som en property

```
[PS /Users/amnadastgir> (Get-Process idle).Id  
Get-Process: Cannot find a process with the name "idle". Verify the process name and call the cmdlet again.  
PS /Users/amnadastgir>
```

**22. (Oblig)**

```
Start-Process notepad.exe  
(Get-Process notepad).WaitForExit()  
(Get-Process notepad).Stop()  
(Get-Process notepad).Kill()
```

### 23. (Ikke oblig)

### 24. (Oblig)

 Både `(ls test.txt).Length` og `(ls test.txt).get\_Length()` gir samme output, nemlig størrelsen på filen "test.txt" i antall bytes. Forskjellen ligger i hvordan informasjonen hentes ut.

`.Length` er en property som kan brukes for å hente ut lengden av en fil eller en streng i PowerShell. Dette er en innebygd funksjon i PowerShell og er derfor veldig enkel å bruke.

`.get\_Length()` er derimot en metode som tilhører .NET-klassen `System.IO.FileInfo`, som er en klasse som håndterer informasjon om filer. `get\_Length()`-metoden kan brukes for å hente ut lengden av en fil, akkurat som `.Length`-propertyen, men metoden krever at man først oppretter et `System.IO.FileInfo`-objekt og bruker dette til å hente ut lengden.

Det er derfor ingen forskjell i resultatet, men det er en liten forskjell i hvordan informasjonen hentes ut.

Angående bash, kan man bruke `ls -l` for å få informasjon om en fil, inkludert filstørrelsen i bytes. Størrelsen vil være angitt under kolonnen "Size". Så det gir mening å kjøre tilsvarende kommandoer som i PowerShell i bash, men man ville brukt et annet syntaks for å få tilsvarende informasjon.

### 25. (Oblig)

Har ikke admin terminal så jeg kan ikke kjøre eller gjøre denne oppgaven.

### 26. (Oblig)

 Når vi endrer koden til å bruke to forskjellige lock-objekter, kan programmet nå komme inn i en situasjon som kalles "deadlock". Dette skjer når begge trådene prøver å få tak i begge lock-objektene på samme tid. For eksempel kan tråd 1 få tak i lock1, og deretter vente på lock2 mens tråd 2 allerede har fått tak i lock2 og venter på lock1. Dermed blir begge trådene stående fast og ventende på hverandre, og programmet går i stå.

Java har mekanismer for å hindre deadlocks, for eksempel ved å sørge for at trådene alltid prøver å få tak i lock-objektene i samme rekkefølge. Dette kan for eksempel oppnås ved å sortere lock-objektene etter en fast rekkefølge og alltid prøve å få tak i dem i denne rekkefølgen. Det er også mulig å bruke ulike verktøy og teknikker for å analysere og unngå deadlocks i programvare.

### 27. (Oblig)

EFFICODE, har allerede gjort.

**28. (Oblig)**

GIDDER IKKE



