

Løsningsforslag uke 10 seminaroppgaver

Oppgave 1:

- a. Når er det fordelaktig å benytte `CountDownLatch`? Og når er det fordelaktig å benytte `CyclicBarrier`? Diskuter forskjellen mellom de 2.
- Når du ønsker å vite at x antall oppgaver/ gjerne av x antall tråder er utført er det en fordel å benytte `CountDownLatch`. Denne teller ned til 0 og åpner da barrieren. `CountDownLatch` kan ikke resettes.
 - `CyclicBarrier` er nyttig når du vil at det skal skje noe for hver n-te gang noe annet har blitt utført, uavhengig av hvilke tråder som gjorde oppgaven. `CyclicBarrier` teller ned til 0, og begynner så på nytt igjen.
- b. Du ønsker å skrive ut noe på skjermen hver 5. gang en oppgave er utført, uavhengig av hvilke tråder som har gjort oppgaven. Hva slags barriere ville du benyttet her?
- `CyclicBarrier` ref det over.

Oppgave 2:

1. Lag et program som starter opp 8 tråder. Alle trådene skal si hei 2 ganger, men ingen av trådene har lov til å si hei den andre gangen før alle trådene er ferdig med å si hei den første gangen.
2. Utvid deloppgave 1, nå skal trådene skrive ut hei 3 ganger. En tråd kan ikke skrive ut hei den tredje gangen, før alle trådene er ferdig med å skrive hei ut den andre gangen

Fasit se oppgave 3 og 4 her:

<https://www.uio.no/studier/emner/matnat/ifi/IN1010/v20/repetisjonskurs/trader/>

Oppgave 3:

Endre oppgave 2 fra forrige uke, slik at personen som fyller opp brusautomaten blir ferdig når alle kundene er ferdig.

Oppgave 4:

Skriv et program som starter noen tråder (akkurat hvor mange bestemmer du selv). Trådene er en Deltager, og de konkurrerer i å sende det største tallet til monitoren (Tallet de sender skal genereres tilfeldig, og skal være et positive heltall). Den tråden som har det største tallet skal slutt skal skrive ut at den har vunnet konkurransen, men for at en tråd skal vite om den har skrevet ut det høyeste tallet må den vente til alle trådene er ferdig. Monitoren trenger to metoder, en for å ta imot et nytt tall, og en for å returnere det. Husk at kun en tråd kan endre på data om gangen.

Løsning:

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

import java.util.Random;

/**
 * A given number of threads, contenders, generate a random
 * number. The contender who submits the largest value wins.
```

```

*
* In order for a contender to know if they won, they must be
* able to wait until all contenders are done.
*/
public class CountdownExample {
    private static final int NUM_THREADS = 10;

    public static void main(String[] args) {
        // Make a barrier, set to wait for NUM_THREADS calls to countdown().
        CountDownLatch allDoneBarrier = new CountDownLatch(NUM_THREADS);

        // Make and start threads. No need to store threads for waiting,
        // we can use the barrier for this.
        KeepLargestMonitor monitor = new KeepLargestMonitor();
        for (int i = 0; i < NUM_THREADS; i++) {
            new Thread(new Contender(monitor, allDoneBarrier)).start();
        }

        // Wait for all contenders to submit a number.
        try {
            allDoneBarrier.await();
        } catch (InterruptedException e) {}
        System.out.println("Largest: " + monitor.getLargest());
    }
}

/* Monitor to save the largest recieved value. */
class KeepLargestMonitor {
    private final Lock lock = new ReentrantLock();
    private int largest;

    public int getLargest() { return largest; }

    public void giveNumber(int number) {
        lock.lock();
        try { largest = Math.max(largest, number); }
        finally { lock.unlock(); }
    }
}

/* Conteder class may skip constructor if made as anonymous. */
class Contender implements Runnable {
    private final KeepLargestMonitor monitor;
    private final CountDownLatch allDoneBarrier;
    private final int id;
    private static int numberOfWorkersDoingThisJob = 0;

    public Contender(KeepLargestMonitor monitor, CountDownLatch allDoneBarrier) {
        this.id = numberOfWorkersDoingThisJob++;
        this.monitor = monitor;
        this.allDoneBarrier = allDoneBarrier;
    }

    public void run() {
        // Generate, and submitt a random number.
        Random random = new Random();
        int number = random.nextInt(100); // Max of 100.
        System.out.printf("Thread #%d generated number: %d\n", id, number);
        monitor.giveNumber(number);

        // Report that we are done, then wait for the rest.
        allDoneBarrier.countDown();
        try {
            allDoneBarrier.await();
        } catch (InterruptedException e) {}
    }
}

```

```
    // If we submitted the largest value, we won!  
    if (number == monitor.getLargest())  
        System.out.printf("Thread #%d won!.\n", id);  
    }  
}
```