Seminaropplegg uke 14

Barrierer

En barriere tenkes på som et samlingspunkt i koden, der alle stopper og venter på at alle andre også har kommet frem. Deretter kan vi la alle trådene fortsette, *samtidig*.

* CountDownLatch(int count) - Alle slipper gjennom barrierer når .countDown() har blitt kalt count ganger. Vi kan *velge* å vente ved barrieren med .await().
* CyclicBarrier(int count) - Når count tråder har kalt på .await(), slipper alle gjennom. Barrieren kan så *brukes på nytt* på samme måte.

**Eksempel:**

|  |
| --- |
| class Arbeider implements Runnable {    private final CountDownLatch ferdigSignal;    Arbeider(CountDownLatch ferdig) {       ferdigSignal = ferdig;    }    public void run() {       try {         System.out.println("Gjor arbeid!");         gjorArbeid();         ferdigSignal.countDown();       } catch (InterruptedException ex) {} // return;    }     void gjorArbeid() { ... }  }  class TestCountdown { // ...    void main() throws InterruptedException {      int antTraader = 5;      CountDownLatch ferdigSignal = new CountDownLatch(antTraader);       for (int i = 0; i < antTraader; ++i) {        new Thread(new Arbeider(ferdigSignall)).start();      }        ferdigSignal.await();           // venter paa alle traadene        System.out.println("Alle er ferdige!);    }  } |

**Oppgave 1:**

1. 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. CountDownLatchen 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.

1. 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:**

Fyll inn manglende kode.

**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 submitts the largest value wins.  
 \*  
 \* In order for a conteder to know if they won, they must be  
 \* able to wait untill 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 submitt 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 contructor 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);  
    }  
}