Seminaroppgaver uke 14

Oppgave 1. Diskuter sammen:

a) Når er det fordelaktig å benytte CountDownLatch? Og når er det fordelaktig å benytte CyclicaBarrier?

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?

Oppgave 2. Fyll inn manglende kode:

Programmet som lages er dokumentert under. Kort fortalt er det et program hvor hver tråd er en deltager i ett spill, deltageren trekker ett tilfeldig generert tall. Den tråden som får det høyeste tallet blir vinneren av konkurransen, men man kan ikke kåre vinneren før alle trådene har trukket tall.

```
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(____________);

        // 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++) {


              ____________________________________

        }

        // Wait for all contenders to submitt a number.
        try {



              ________________________

        } 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) {
       //Laas laasen:


        ___________________
        try { largest = Math.max(largest, number); }
        finally { lock.unlock(); }
    }
}

/* Conteder class may skip contructor if made as anonymous. */


class Contender implements ___________ {
    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 = ___________;
        this.monitor = monitor;
        this.allDoneBarrier = allDoneBarrier;
    }
```

```
    public void ___________() {
        // 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.
//signaliser at traaden har gitt sitt tall til monitoren:

        allDoneBarrier._________();
        try {
//be traaden om aa vente til de andre er ferdige:

            allDoneBarrier._______();
        } catch (InterruptedException e) {}

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