Løsningsforslag - gruppetime 12

Oppgave 1 - Løsningsforslag

CountDownLatch:

- En CountDownLatch lar én eller flere tråder vente til et sett med operasjoner som kjøres på andre tråder er fullførte.
- Den initialiseres med en gitt tellerverdi, og await()-metoden brukes til å blokkere en eller flere ventertråder til telleren når null.
- Den er nyttig når du skal synkronisere én eller flere tråder som skal vente på at en eller flere hendelser skal skje.
- Eksempel: Vent til alle ressurser er lastet før du starter en applikasjon, eller vent på at flere uavhengige beregninger er utført før du fortsetter med neste trinn.

.join():

- join() brukes til å vente på at en eller flere tråder skal avslutte før tråden som venter fortsetter å kjøre.
- join() kan brukes av andre tråder enn hovedtråden.
- Eksempel: Main-tråden venter på andre tråder før den fortsetter sin kjøring.

I scenarier hvor det er behov for å synkronisere flere operasjoner eller vente på flere tråder, kan CountDownLatch være mer passende. På den annen side, når en tråd må vente på at én eller flere tråder skal avslutte, uten behov for å synkronisere tilstanden mellom flere av disse trådene underveis, kan .join() være å foretrekke.

Hver femte oppgave:
Når du ønsker å skrive ut noe på skjermen hver femte gang en oppgave er utført, uavhengig av hvilken tråd som har gjort oppgaven, høres CountDownLatch bedre ut fordi det ikke er avhengig av en bestemt tråd, men av antall utførte oppgaver.

Du kan initialisere en CountDownLatch med en teller satt til fem, og hver gang en oppgave er fullført, kan en tråd kalle countDown() på latchen. Så snart telleren når null (etter fem countDown()-kall), kan relevant kode for å skrive ut meldingen bli utført. Husk at du kan “nullstille” latchen ved å opprette en ny instans som du f.eks. lar samme peker peke på.

Oppgave 2 - Løsningsforslag

```java
import java.util.concurrent.CountDownLatch;

public class HeiTraad implements Runnable {
    private static final int ANT_TRAADER = 8;

    // Latches for hver runde
    private static CountDownLatch forsteRunde = new CountDownLatch(ANT_TRAADER);
    private static CountDownLatch andreRunde = new CountDownLatch(ANT_TRAADER);
    private static CountDownLatch tredjeRunde = new CountDownLatch(ANT_TRAADER);

    private String navn;
    public HeiTraad(String navn) {
        this.navn = navn;
    }

    @Override
    public void run() {
        try {
            siHei(1);        // Si "Hei" for forste gang
            forsteRunde.countDown();
            forsteRunde.await();

            siHei(2);        // Si "Hei" for andre gang
            andreRunde.countDown();
            andreRunde.await();

            siHei(3);        // Si "Hei" for tredje gang
            tredjeRunde.countDown();
            tredjeRunde.await();
        } catch (InterruptedException e) {
            System.err.println("En traad ble avbrutt i paavente av de andre.
            Avslutter programmet.");
            System.exit(1);
        }
    }

    private void siHei(int runde) {
        System.out.println("Hei " + runde + " fra traad " + navn);
    }

    public static void main(String[] args) {
        for (int i = 1; i <= ANT_TRAADER; i++) {
            new Thread(new HeiTraad("Traad-" + i)).start();
        }
    }
}
```

Oppgave 3 - Løsningsforslag

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

class Monitor {
    private int maxTall = Integer.MIN_VALUE;
    private final ReentrantLock lock = new ReentrantLock();

    public void mottaTall(int tall) {
        lock.lock();
        try {
            if (tall > maxTall) {
                maxTall = tall;
            }
        } finally {
            lock.unlock();
        }
    }

    public int henteStorsteTall() {
        lock.lock();
        try {
            return maxTall;
        } finally {
            lock.unlock();
        }
    }
}

class Deltaker implements Runnable {
    private final String navn;
    private final Monitor monitor;
    private final CountDownLatch ferdigSignal;
    private final Random random = new Random();

    public Deltaker(String navn, Monitor monitor, CountDownLatch ferdigSignal) {
        this.navn = navn;
        this.monitor = monitor;
        this.ferdigSignal = ferdigSignal;
    }

    @Override
    public void run() {
        int tilfeldigTall = random.nextInt(Integer.MAX_VALUE);
        monitor.mottaTall(tilfeldigTall);
        ferdigSignal.countDown();
```

```
        try {
            ferdigSignal.await(); // Vente til alle traadene er ferdige
            if (tilfeldigTall == monitor.henteStorsteTall()) {
                System.out.printf("\nJeg, %s, har vunnet konkurransen med tallet %d!\n",
                navn, tilfeldigTall);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.err.printf("%s ble avbrutt og er ikke lenger med i konkurransen.",
            navn);
        }
    }
}

public class Konkurranse {
    public static void main(String[] args) throws InterruptedException {
        final int ANT_TRAADER = 10;
        CountDownLatch ferdigSignal = new CountDownLatch(ANT_TRAADER);
        Monitor monitor = new Monitor();

        Thread[] traader = new Thread[ANT_TRAADER];

        for (int i = 0; i < ANT_TRAADER; i++) {
            traader[i] = new Thread(new Deltaker("Deltaker " + (i+1), monitor,
            ferdigSignal));
            traader[i].start();
        }

        for (int i = 0; i < ANT_TRAADER; i++) {
            traader[i].join();
        }

        System.out.println("Konkurransen er over. Vel hjem!");
    }
}
```

Oppgave 4 - Løsningsforslag (veldig forenklet)

```
import java.util.concurrent.CountDownLatch;

class Rontgenavdelingen implements Runnable {
    private CountDownLatch latch;

    public Rontgenavdelingen(CountDownLatch latch) {
        this.latch = latch;
    }

    @Override
    public void run() {
        try {
            System.out.println("Rontgenavdelingen genererer og tolker bilder.");
            Thread.sleep(3000);
            latch.countDown();
            System.out.println("Rontgenbildene er klare og tolket.");
        } catch (InterruptedException e) {
            System.err.println("Rontgenavdelingen feilet. Sykehuset maa stenge!");
            System.exit(1);
        }
    }
}

class Operasjonsavdelingen implements Runnable {
    private CountDownLatch latch;

    public Operasjonsavdelingen(CountDownLatch latch) {
        this.latch = latch;
    }

    @Override
    public void run() {
        try {
            System.out.println("Operasjonsavdelingen venter på rontgenbilder.");
            latch.await();
            System.out.println("Operasjonsavdelingen har startet operasjonen.");
            Thread.sleep(5000);
            System.out.println("Operasjonsavdelingen har fullfort operasjonen.");
        } catch (InterruptedException e) {
            System.err.println("Operasjonsavdelingen feilet. Sykehuset maa stenge!");
            System.exit(1);
        }
    }
}
```

```java
public class Sykehus {
    public static void main(String[] args) {
        CountDownLatch latch = new CountDownLatch(1);

        Thread rontgenThread = new Thread(new Rontgenavdelingen(latch));
        Thread operasjonThread = new Thread(new Operasjonsavdelingen(latch));
        operasjonThread.start();
        rontgenThread.start();

        try {
            rontgenThread.join();
            operasjonThread.join();
        } catch (InterruptedException e) {
            System.err.println("Sykehuset feilet i paavente av rontgen og operasjon, og
            maa derfor stenge!");
            System.exit(1);
        }

        System.out.println("Sykehuset har fullfort dagens prosedyrer!");
    }
}
```