Uke 12

Seminar

Litt om design

Design

Hva er et godt design? (fra IN1030):

- En god utforming **gjør den jobben den er ment å gjøre**.
- En god utforming er enkel og elegant.
    - Eleganse innebærer å finne akkurat riktig abstraksjonsnivå.
- En god utforming er **gjenbrukbar, utvidbar og enkel å forstå**.
- Et godt objekt har et **lite og veldefinert ansvarsområde**.
- Et godt objekt **skjuler implementasjon detaljer fra andre objekter**

Testing

Ha som mål å knekke programmet, ikke å få alle testene til å kjøre. Det kan være lurt å skrive testene før selve programmet, da dette gjør at du vil måtte tenke på eventuelle utfordringer før man begynner med kodingen - lite gjennomtenkt kode fører fort til spagetti kode.

Eksempel: Skrive enkle tester for å automatisere oblig testingen

```
ArrayList<Integer> liste = new ArrayList<Integer>();
liste.add(5);
liste.add(11);

test("ArrayList, index 2", liste.get(2), 12);


boolean test(String beskrivelse, int inn, int forventet){
    System.out.print("TEST: " + beskrivelse + ": " + inn + " == " + forventet + ": ");
    if(inn == forventet){
        System.out.println("RIKTIG");
        return true;
    }else{
        System.out.println("FEIL");
        return false;
```

```
    }
}

boolean test(String beskrivelse, double inn, double forventet){
    System.out.print("TEST: " + beskrivelse + ": " + inn + " == " + forventet + ": ");
    if(inn == forventet){
        System.out.println("RIKTIG");
        return true;
    }else{
        System.out.println("FEIL");
        return false;
    }
}

boolean test(String beskrivelse, String inn, String forventet){
    System.out.print("TEST: " + beskrivelse + ": " + inn + " == " + forventet + ": ");
    if(inn.equals(forventet)){
        System.out.println("RIKTIG");
        return true;
    }else{
        System.out.println("FEIL");
        return false;
    }
}
```

Tråder: fortsettelse

Kniv/saks oppgave: https://trix.ifi.uio.no/assignments/1005

Løsning

```
import java.util.ArrayList;
import java.util.concurrent.locks.ReentrantLock;

// Oppgave 1
class Kniv {}

// Oppgave 2
class Saks {
    Kniv[] kniver = new Kniv[2];

    Saks(Kniv k1, Kniv k2) {
        kniver[0] = k1;
        kniver[1] = k2;
    }
}
```

```
// Oppgave 3
class KnivMonitor {

    private ReentrantLock lock = new ReentrantLock();
    private ArrayList<Kniv> knivHaug = new ArrayList<>();

    public int antallKniver(){
        return knivHaug.size();
    }

    public void settInn(Kniv kniv) {
        lock.lock();
        try {
            knivHaug.add(kniv);
        }

        finally {
            lock.unlock();
        }
    }

    public Kniv[] taUtTo() {
        lock.lock();
        try {
            if (knivHaug.size() < 2) {
                return null;
            }
            Kniv[] kniver = new Kniv[2];
            kniver[0] = knivHaug.remove(0);
            kniver[1] = knivHaug.remove(0);
            return kniver;
        }
        finally {
            lock.unlock();
        }
    }
}

// Oppgave 4
class KnivProdusent implements Runnable{

    private KnivMonitor knivMonitor;
    private int antallKniver;

    KnivProdusent(KnivMonitor knivMonitor, int antallKniver) {
        this.knivMonitor = knivMonitor;
        this.antallKniver = antallKniver;
    }

    @Override
    public void run() {
        for (int i = 0; i < antallKniver; i++) {
            Kniv nyKniv = new Kniv();
            knivMonitor.settInn(nyKniv);
        }
    }
}
```

```
// Oppgave 5
class SaksMonitor {

    private ReentrantLock lock = new ReentrantLock();
    private ArrayList<Saks> ferdigeSakser = new ArrayList<>();

    public void settInn(Saks saks) {
        lock.lock();
        try {
            ferdigeSakser.add(saks);
        }
        finally {
            lock.unlock();
        }
    }

    public int antallSakser() {
        return ferdigeSakser.size();
    }
}

// Oppgave 6
class SaksProdusent implements Runnable{
    private KnivMonitor knivMonitor;
    private SaksMonitor saksMonitor;

    SaksProdusent(SaksMonitor saksMonitor, KnivMonitor knivMonitor) {
        this.saksMonitor = saksMonitor;
        this.knivMonitor = knivMonitor;
    }

    @Override
    public void run() {
        while (knivMonitor.antallKniver() > 1) {
            Kniv[] kniver = knivMonitor.taUtTo();
            if (kniver != null) {
                Saks nySaks = new Saks(kniver[0], kniver[1]);
                saksMonitor.settInn(nySaks);
            }
        }
    }
}

// Oppgave 7

class Hovedprogram {

    public static void main(String[] args) {

        int antTraader = 10;
        int antallKniverPerTraad = 1000;

        KnivMonitor knivMonitor = new KnivMonitor();
        SaksMonitor saksMonitor = new SaksMonitor();

        Thread[] knivProdusenter = new Thread[antTraader];
```

```
        Thread[] saksProdusenter = new Thread[antTraader];

        for (int i = 0; i < antTraader; i++) {
            knivProdusenter[i] = new Thread(new KnivProdusent(knivMonitor, antallKniverPerTraad));
            saksProdusenter[i] = new Thread(new SaksProdusent(saksMonitor, knivMonitor));
            knivProdusenter[i].start();
        }

        System.out.println("Startet alle kniv produsenter.");
        System.out.println("Venter paa at kniv produsenter skal bli ferdige. \n");

        for (Thread knivProdusent : knivProdusenter) {
            try {
                knivProdusent.join();
            } catch (InterruptedException e) {
                System.err.println(e);
            }
        }
        System.out.println("Kniv produsenter ferdige.");
        System.out.println("Starter saks produsenter.\n");

        for (Thread saksProdusent : saksProdusenter) {
            saksProdusent.start();
        }

        for (Thread saksProdusent : saksProdusenter) {
            try {
                saksProdusent.join();
            } catch (InterruptedException e) {
                System.err.println(e);
            }
        }

        System.out.println("Ferdig med aa lage sakser fra kniver.");
        System.out.println("Antall sakser produsert = " + saksMonitor.antallSakser());
    }
}
```