Ukeoppgaver, gr.time 15 (Design og testbarhet)

Oppgave 1: Brudd på S-prinsippet i SOLID

Ta utgangspunkt i koden nedenfor, og utbedre den slik at den *ikke lenger bryter med S-prinsippet* i SOLID.

```
interface Monitor {
    void lesInnData();
    void slaaSammenProduktHashMaps();
    void identifiserStavefeilIHashMaps();
}

class Aktivitetstraad implements Runnable {
    private Monitor monitor;
    public Aktivitetstraad(Monitor monitor) {
        this.monitor = monitor;
    }

    @Override
    public void run() {
        monitor.lesInnData();
        monitor.slaaSammenProduktHashMaps();
        monitor.identifiserStavefeilIHashMaps();
    }
}
```

Oppgave 2: Brudd på D-prinsippet i SOLID

Ta utgangspunkt i koden nedenfor, og utbedre den slik at den *ikke lenger bryter med D-prinsippet* i SOLID. Hint: biler kan ha ulike motortyper; for eksempel en bensin- eller dieselmotor, eller en elektrisk motor.

```
class Bensinmotor {
    public void start() {
        System.out.println("Starter bensinmotoren...");
    }
}

class Bil {
    private Bensinmotor motor;

    public Bil() { this.motor = new Bensinmotor(); }

    public void startMotor() { motor.start(); }
}
```

Oppgave 3: Brudd på I-prinsippet i SOLID

Ta utgangspunkt i koden nedenfor, og utbedre den slik at den *ikke lenger bryter med I-prinsippet* i SOLID.

```
interface Kafeansatt {
    void lagMat(); void lagKaffe();
}

class Kokk implements Kafeansatt {
    @Override
    public void lagMat() { System.out.println("Lager et deilig maaltid..."); }
    @Override
    public void lagKaffe() { throw new UnsupportedOperationException(); }
}

class Barista implements Kafeansatt {
    @Override
    public void lagMat() { throw new UnsupportedOperationException(); }
    @Override
    public void lagKaffe() { System.out.println("Lager en deilig kaffe..."); }
}
```

Oppgave 4: Enhetstesting og integrasjonstesting

Hvordan vil du gå frem for å enhetsteste og integrasjonsteste koden nedenfor? Planlegg testene (steg 1), gjennomfør dem (steg 2) og kontroller resultatene (steg 3). Forklar hvorfor det er hensiktsmessig å bruke en egen klasse for kursdeltakelse.

```
import java.util.List; import java.util.ArrayList;

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

    public String getName() {
        return this.navn;
    }
}

class Kurs {
    private String tittel;
    public Kurs(String tittel) {
        this.tittel = tittel;
    }
```

```
    public String hentTittel() {
        return this.tittel;
    }
}

class Kursdeltakelse {
    private Kurs kurs;
    private List<Deltaker> deltakere;

    public Kursdeltakelse(Kurs kurs) {
        this.kurs = kurs;
        this.deltakere = new ArrayList<>();
    }

    public void leggTilDeltaker(Deltaker student) {
        this.deltakere.add(student);
    }

    public List<Deltaker> hentDeltakere() {
        return this.deltakere;
    }

    public Kurs hentKurs() {
        return this.kurs;
    }
}
```

Oppgave 5: Designmønster for innlevering 5

Basert på den andre forelesningen om GUI og forelesningen om design og testbarhet, hvilket designmønster kan være anbefalt å bruke når du utarbeider innlevering 5 om GUI? Godt tips: Planlegg og implementer programmet ditt med utgangspunkt i det anbefalte designmønsteret du kommer frem til.