Løsningsforslag, gr.time 15 (Design og testbarhet)

Oppgave 1 - Løsningsforslag: Brudd på S-prinsippet i SOLID

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

class LeseTraad implements Runnable {
    private Monitor monitor;

    public LeseTraad(Monitor monitor) {
        this.monitor = monitor;
    }

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

public class SlaaSammenHashMapsTraad implements Runnable {
    private Monitor monitor;

    public SlaaSammenHashMapsTraad(Monitor monitor) {
        this.monitor = monitor;
    }

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

class FinneStavefeilTraad implements Runnable {
    private Monitor monitor;

    public FinneStavefeilTraad(Monitor monitor) {
        this.monitor = monitor;
    }

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

Oppgave 2 - Løsningsforslag: Brudd på D-prinsippet i SOLID

```
interface Motor {
    void start();
}

//Eksempel: en bensindrevet motor
class Bensinmotor implements Motor {
    @Override
    public void start() {
        System.out.println("Starter bensinmotoren...");
    }
}

public class Bil {
    private Motor motor;

    public Bil(Motor motor) {
        this.motor = motor;
    }

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

Oppgave 3 - Løsningsforslag: Brudd på I-prinsippet i SOLID

```
interface Matlagingsansatt {
    void lagMat();
}

interface Kaffelagingsansatt {
    void lagKaffe();
}

class Kokk implements Matlagingsansatt {
    @Override
    public void lagMat() { System.out.println("Lager et deilig maaltid..."); }
}

class Barista implements Kaffelagingsansatt {
    @Override
    public void lagKaffe() { System.out.println("Lager en deilig kaffe..."); }
}
```

Oppgave 4 - Løsningsforslag: Enhetstesting og integrasjonstesting

Under testingen er det viktig at du tester hver klasse med metoder for seg (enhetstesting). Du skal også teste at Deltaker, Kurs og Kursdeltakelse fungerer sammen (integrasjons-testing). For både enhetstester og integrasjonstest, sjekk om forventet informasjon samsvarer med informasjonen fra programmet. Hvis det ikke er samsvar, må du endre programmet slik at det blir korrekt/feilfritt. Når du tester kan du - men må ikke - bruke *assert*-nøkkelordet Lars Tveito gjennomgikk i forelesningen om invarianter.

Bruken av en egen klasse for Kursdeltakelse overholder prinsippet om Separation of Concerns og Single Responsibility Principle (SRP) i programvareutvikling. Du kan relatere måten å dele opp et objektorientert program på til UML-modelleringen i IN1030.

I denne oppgaven har vi tre klasser: Deltaker, Kurs, og Kursdeltakelse.

- Deltaker-klassen representerer en student eller deltaker i et kurs. Den har ansvar for å lagre og hente deltakerens navn.
- Kurs-klassen representerer et bestemt kurs. Den har ansvar for å lagre og hente kursets tittel.
- Kursdeltakelse-klassen representerer deltakelse i et kurs. Den har ansvar for å lagre hvilket kurs det gjelder og hvilke deltakere som er påmeldt kurset. Klassen har også metoder for å legge til deltakere og hente listen over deltakere.

Ved å ha en egen klasse for Kursdeltakelse, holder vi logikken for kursdeltakelse (som å legge til deltakere og hente listen over deltakere) adskilt fra logikken for Deltaker og Kurs. Dette gjør koden mer organisert, lettere å forstå og vedlikeholde, og mer fleksibel for fremtidige endringer eller utvidelser. For eksempel, hvis vi senere ønsker å legge til funksjonalitet for å fjerne en deltaker fra et kurs, kan vi gjøre det i Kursdeltakelse-klassen uten å påvirke Deltaker- eller Kurs-klassene.

Oppgave 5 - Løsningsforslag: Designmønster for innlevering 5

Model-View-Controller (MVC). Modell for data og datahåndtering, View for GUI og Controller for delegering mellom View og Model. Vi sier gjerne at Controller håndterer brukerinteraksjonen som finner sted i det grafiske brukergrensesnittet. Model-View-Controller (MVC) tar utgangspunkt i SOLID-prinsippene, og sørger dermed for høy grad av “cohesion” og “loose coupling” i programmet (begreper du har møtt på tidligere i IN1010).