IN1010 uke 15

Gruppe 2

Agenda

- Design og testbarhet
  - Hva man skal tenke på
  - SOLID-prinsippene & MVC
  - Bruke interfaces
- Å skrive tester
  - Arrange, act, assert
  - Kodeeksempel: Enhetstest
- Kodeeksempel: Bruke MVC-mønsteret

- Jobbe med oppgaver/trix/oblig!

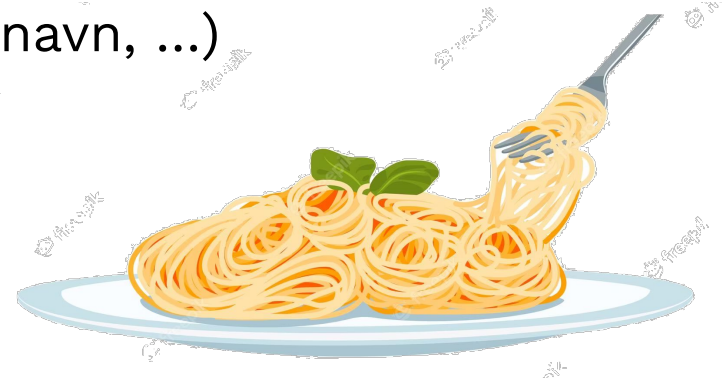
Design og testbarhet

Tenke på når man skriver kode

- Oftest skal kode leses/endres masse av deg/andre, ikke bare noe man skriver 1 gang og aldri ser på igjen

- For å lett kunne lese og endre:
  - Leselig (kommentarer, variabelnavn, ...)
  - God design/struktur

- Hvis ikke: Spaghettikode :(
  - Vanskelig å følge,
    for mange deler er viklet inn i hverandre



SOLID-prinsippene

S
O
L
I
D

- Single responsibility principle
  En klasse bør bare ha ett ansvarsområde, og dette bør være veldefinert
- Open/closed principle
  Komponenter bør være åpne for utvidelser, men lukket for modifikasjoner. Eksempel: klasser kontrollerer hva subklasser kan endre (private, protected, final, etc).
- Liskov substitution principle
  Objekter i et program bør kunne byttes ut med instanser av subtyper uten at dette endrer programmets korrekthet.
- Interface segregation principle
  Mange små grensesnitt er bedre enn store, monolittiske "super-grensesnitt"
- Dependency inversion principle
  Komponenter bør avhenge av abstraksjoner, og ikke konkrete implementasjoner

Bilde fra Eyvind W. Axelsen sin forelesning uke 13 v23

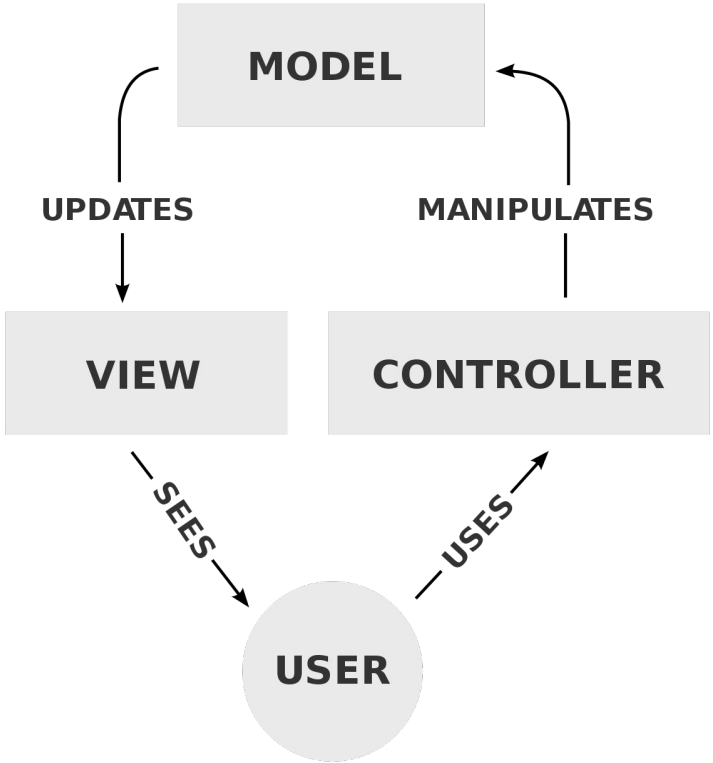
SOLID-prinsippene

Viktigst i IN1010 / hovedpoenget:

- Hver klasse skal ha **1** ansvarsområde
  - Skal være tydelig hva ansvaret er

- Ting bør avhenge av abstraksjoner (interface) og ikke konkrete implementasjoner (eksempel snart)

SOLID-prinsippene og MVC (Model-View-Controller)



SOLID-prinsippene og MVC (Model-View-Controller)

MVC-mønsteret følger SOLID-prinsippene.
Noen eksempler:

- **S**ingle Responsibility Principle
  I MVC: hvert komponent (Model, View, Controller) ansvarlig for en bestemt oppgave:
  - Model: data/logikk
  - View: det vi ser
  - Controller: mellomledd mellom Model & View.

SOLID-prinsippene og MVC (Model-View-Controller)

- **O**pen/Closed Principle
  “Komponenter bør være åpne for utvidelser, men lukket for modifikasjoner”

  I MVC: kan lett utvide (f.eks. legge til flere visninger/modeller uten å endre i gammel kode)

- **L**iskov Substitution Principle
  I MVC: kan erstatte/endre visningen uten å påvirke resten av programmet.

Hvorfor bruke interfaces

```
class SpillKontroller {
    SpillTerminalView view;
    SpillModell modell;

    public SpillKontroller(SpillTerminalView view) {
        this.view = view;
        this.modell = new SpillModell();
    }
}
```

Hvis vi vil at Spillet skal kunne vises i GUI istedenfor....?

```
class SpillKontroller {
    SpillView view;
    SpillModell modell;

    public SpillKontroller(SpillView view) {
        this.view = view;
        this.modell = new SpillModell();
    }
}


interface SpillView {
    /* metoder som alle SpillViews skal tilby */
}
```

Hvorfor bruke interfaces

- Bruker SpillView interface istedenfor den direkte klassen (implementasjonen)

  → Enkelt bytte ut view til annen klasse

  → Gjenbruk av koden, spill for både GUI og terminal

  → Veldig tydelig hva en SpillView-klasse skal gjøre

Å skrive tester

Enhetstesting

- Tester bare én del/klasse
- Skal være uavhengig av resten av programmet

→ Sikre at delen/klassen fungerer som den skal

→ Bruke testen senere for å se at delen fortsatt fungerer etter endringer i programmet

Kodeeksempel

Enhetstest for Katt-klasse

Kodeeksempel

GUI sjekkliste skrevet etter MVC-mønsteret

Jobb med oppgaver!

IN1010 Emnesiden → Grupper →
Gruppe 2 → Uke15

Husk å ta pause ☕