

# REPETISJONSKURS IN1010 - GUI

Sivert Fjeldstad Madsen

# I DAG

- GUI
- Standard oppsett
- MVC
- Livekode



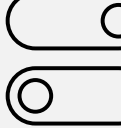
*graphic design is my passion*

# GUI – GRAPHICAL USER INTERFACE

- De fleste programmene vi har skrevet i INI010 blir brukt gjennom terminalen
- Programmer som er tenkt å bli brukt mye har sjelden terminalen som eneste interaksjonspunkt
- Man programmerer gjerne et eget *interaksjonslag* som lar brukeren påvirke programmet
  - Det er dette vi kaller et brukergrensesnitt
- Her vil en bruker kunne få se og interagere med programmet på ulike måter



# GUI - KODESKIKK

- En lur ting å tenke på når man programmerer større programmer som skal ha et GUI, er å holde det adskilt fra resten av koden 
- Tenk at du skal kunne enkelt bytte mellom flere forskjellige GUIer ved bare å endre hvilken klasse som blir brukt. F.eks:
  - En klasse som gir et grensesnitt basert på **terminalen** (som i oblig 4)
  - En klasse som gir et grensesnitt basert på Swing og AWT
- På denne måten er det enkelt å gjøre endringer på grensesnittet uten å måtte endre masse underliggende logikk, og vice versa
- Dette er en annen form for innkapsling

# DEKLARATIV PROGRAMMERING

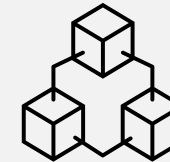
- De fleste moderne rammeverk for GUIer bruker det som kalles **deklarativ programmering**
- I IN1010 har vi i all hovedsak drevet med **imperativ** programmering, med noen få smakebiter av **funksjonell** programmering
- Deklarativ programmering går ut på at programmereren (deg) forteller *hva* de vil ha, men ikke i like stor grad *hvordan*
- Dette gjør at vi slipper å forholde oss til mange ting, sånn som hvordan man oppretter tråder for de ulike GUI-delene og hvordan man faktisk tenger et vindu på en skjerm
- Det gjør også at vi må godta å ikke ha like god kontroll over alt som vi har blitt vant til



# GENERELT OPPSETT

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;
```

Importerer bibliotekene vi trenger



```
...  
try {  
    UIManager.setLookAndFeel(  
        UIManager.getCrossPlatformLookAndFeelClassName()  
    );  
} catch (Exception e) {  
    System.exit(1);  
}
```

Setter utseendet til å matche operativsystemet (hvis mulig)

```
JFrame vindu = new JFrame("Dette er navnet på vinduet!");  
vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
...  
vindu.pack();  
vindu.setLocationRelativeTo(null);  
vindu.setVisible(true);
```

Oppretter vinduet

Gjør så programmet stopper når vinduet lukkes

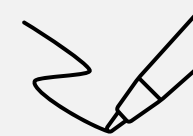
Legger alt innhold i vinduet

Setter vinduet midt på skjermen

Gjør vinduet synlig

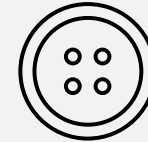
# LEGG TIL ELEMENTER I VINDUET

- Et helt tomt vindu er ikke spesielt interessant
- Vi bruker innebygde klasser for å representere ulike ting:
  - **JPanel** – for å lage tegneflater
  - **JLabel** – for tekst
  - **TextField** og **TextPane** – for tekstfelder
  - **Button** – for knapper
- Elementer legges til en tegneflate med **.add()**
- Tegneflater legges til vinduet (JFrame) med **.add()**



# JBUTTON

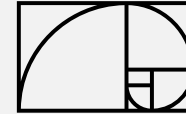
- Knapper er en av de vanligste måtene for en bruker å interagere med et program
- Vi må definere hva som skal skje når en knapp blir trykket på, og dette gjør vi med indre klasser
  - Ganske likt som arbeiderklassene når vi programmerer med tråder
- «Hendelses»-klassen vår må implementere interfacet **ActionListener**
  - Som krever at vi skriver metoden **public void actionPerformed (ActionEvent e)**
  - Denne metoden definerer hva som skjer når knappen blir trykket på





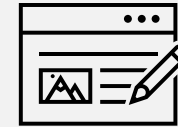
# INTERFACE

- Det kan være en god idé å lage et **interface** for view-klassen
  - La alle views tilknyttet programmet implementere dette
- På denne måten blir det enklere å erstatte viewet med et annet dersom man skulle ønske det
- I kontrolleren (og eventuelt modellen) kan man da alltid vite at metodene man bruker finnes uansett hvilket view man bruker
- Dette gjør det enklere for andre å bygge videre på programmet ditt
- Det vil også kunne gjøre testing enklere



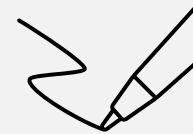
# LAYOUT

- Når vi lager GUI vil vi ofte presentere de ulike elementene på bestemte måter
- Vi kan da bruke ulike former for **layout**
  - Vi kan bestemme dette for alle tegneflater (JPanel)
  - Merk også at vi kan ha tegneflater inne i tegneflater
- **BorderLayout** lar oss velge noen forhåndsdefinerte posisjoner
- **GridLayout** lar oss lage et rutenett
- Slide **39** og **40** fra [forelesningen](#) gir gode eksempler på dette



# FONTER OG FARGER

- Man kan endre fonten og stilen på teksten i alle elementer som inneholder tekst
  - Dette gjøres med `element.setFont()`
- I tillegg kan man endre størrelse, farge, rammer, og bakgrunner på de fleste elementer
- Igjen gir [forelesnings-pdfen](#) en god gjennomgang av dette, på slide **41-44**



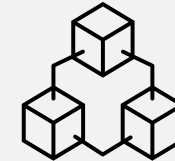
# MODEL-VIEW-CONTROLLER

- Når vi skriver større programmer er det lurt å strukturere dem på en fornuftig måte
- Vi trenger ikke finne opp hjulet på nytt hver gang
- Derfor benytter vi oss gjerne av et allerede eksisterende **programmeringsmønster**
- I IN1010 lærer vi om **MVC**
  - I IN2000 lærer man **MVVM**: Model-View-ViewModel
  - Det finnes også andre

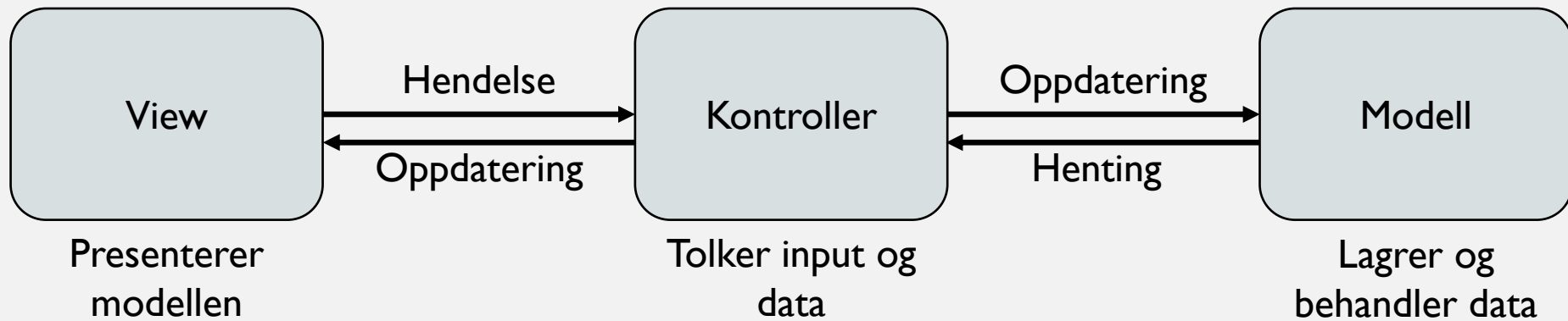


# MVC - FORTSETTELSE

- Tanken bak MVC er å dele opp programmet vårt i tre ulike deler som hver har ansvaret for forskjellige ting
- **Modellen** er der all dataen og mesteparten av logikken bak programmet ligger
  - Dette kan f.eks. være en eller flere lister med data
- **Viewet** er det som presenterer dataen fra modellen til brukeren
  - Typisk **GUI**en til programmet
- **Kontrolleren** er det som ligger mellom og kobler til to sammen
  - All kommunikasjon mellom GUIen og modellen bør skje gjennom kontrolleren



# PROGRAMFLYT I MVC



# OPPSUMMERING

- GUI i Java gir flere muligheter for brukerinteraksjon!
- Lag et ark med de vanligste konstruksjonene i Swing og AWT, siden disse brukes hele tiden
- Ikke tenk på grafisk design på eksamen!
  - Bare pass på å ha med alle delene dere blir bedt om
- MVC handler om en tredeling av ansvar
  - Viewet og modellen kjenner kun til kontrolleren, og kontrolleren kjenner til de to andre

