UNIVERSITETET I OSLO

Institutt for informatikk

Grafiske brukergrensesnitt i Java - del 2
GUI 2

18. april 2023

Stein Gjessing

Agenda

- Å programmere GUI
- Tilpassing av utseende på vinduet (form, farge, . . .)
- Eksempel: Bruk av Telleren vi laget i januar
  - Spesifikasjon av grensesnitt ved hjelp av interface
  - Alternative brukergrensesnitt
- Programmeringsmønster: Modell – View – Controll
- Litt om koblinger mellom - og samhold inne i – objekter
- Eksempel: Tripp – trapp – tresko

- Se også:
  - Horstmann kap 10 og 11
  - Programkode på IN1010-siden (ukene 11 og 12)

Hvordan lage programmer med GUI

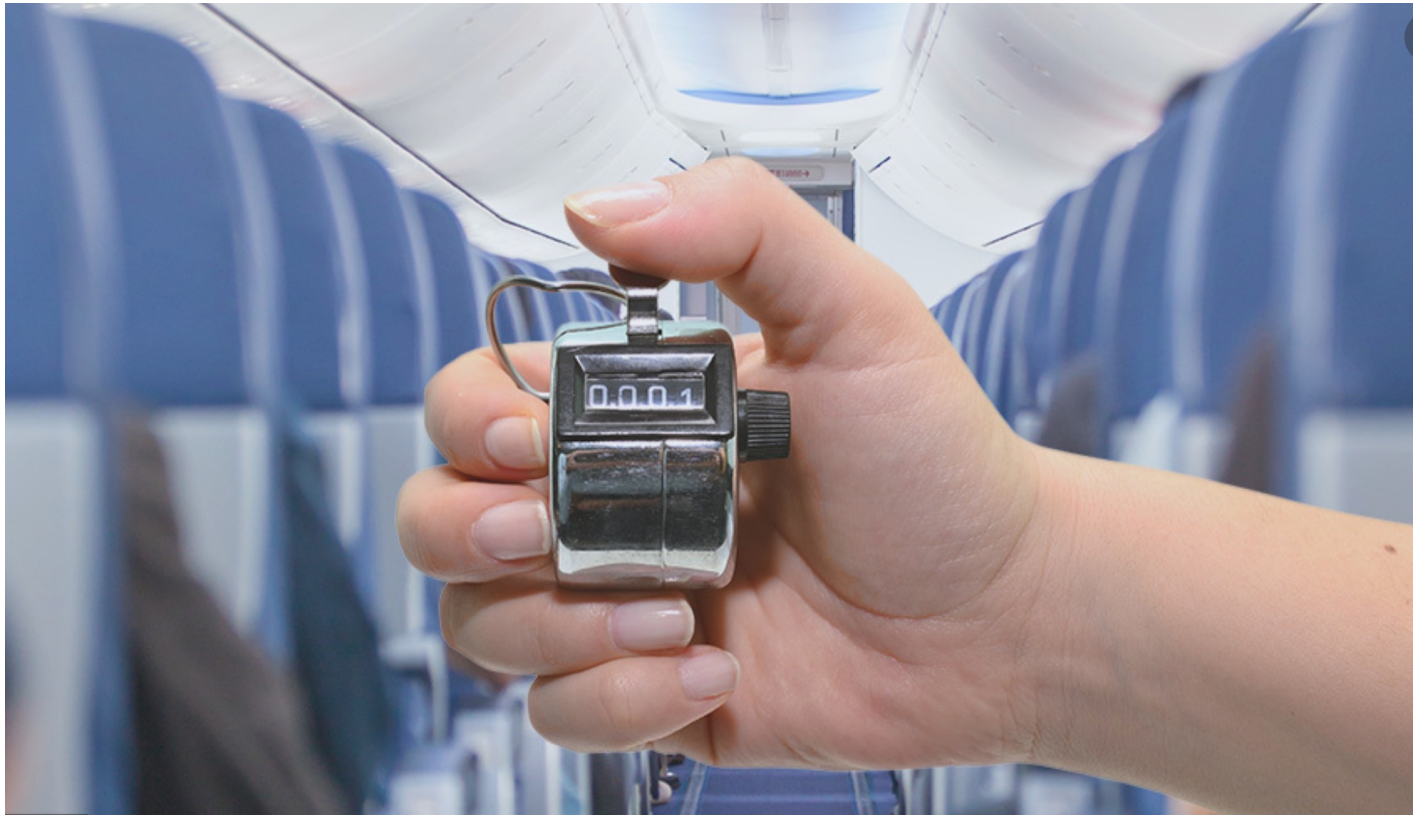
- Design av skjermbildet
  - “Look and feel”
  - Komponenter – hvor ligger de i skjermbildet
  - Rammer – farger – fonter
- Hendelsesdrevne programmer
  - Java: Lag objekter/klasser med metoder som tar seg av hendelsene
  - Bestemme hvilke metoder som tar seg av hva
  - Avgjøre kontrollflyten
    - Hva skjer som følge av en hendelse
  - Pass på at vinduet alltid reagerer raskt

Fra uke 1: En teller (engelsk: counter)

- Vi skal nå sette denne teller inn i et litt større program

Vi skal lage et program der en bruker kommuniserer med en slik teller

Hvordan skal vi planlegge dette programmet?

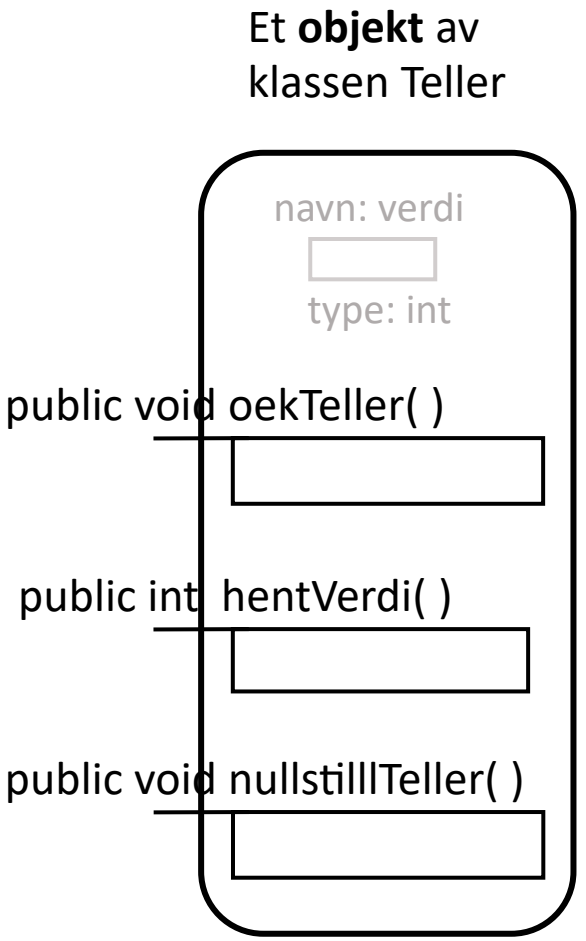


Vi laget et teller-objekt

Når vi skal lage et program der en person bruker en slik teller må vi tenke på

Programvaredesign-
Programvarearkitektur*

*Mer neste uke og 9. mai



Arkitekturmønstre

Snøhettas forslag til regjeringskvartal

Typisk hus i by i USA

Kjente mønstre for programdesign (generelt – ikke bare for GUI)

- Det er mange kjente og velprøvde måter å dele opp programkode. Slike kjente måter kalles ”**design patterns**”.
  - Tatt fra boka “The Timeless Way of Building”, Christoffer Alexander, Oxford University Press, 1979, ISBN 0195024028
    - *“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such away that you can use this solution a million times over, without doing it the same way twice”*

Et slikt **programmeringsmønster** har mange lært i IN1000 med kommandoløkka som skriver ut en meny og tar kommandoer fra brukeren.

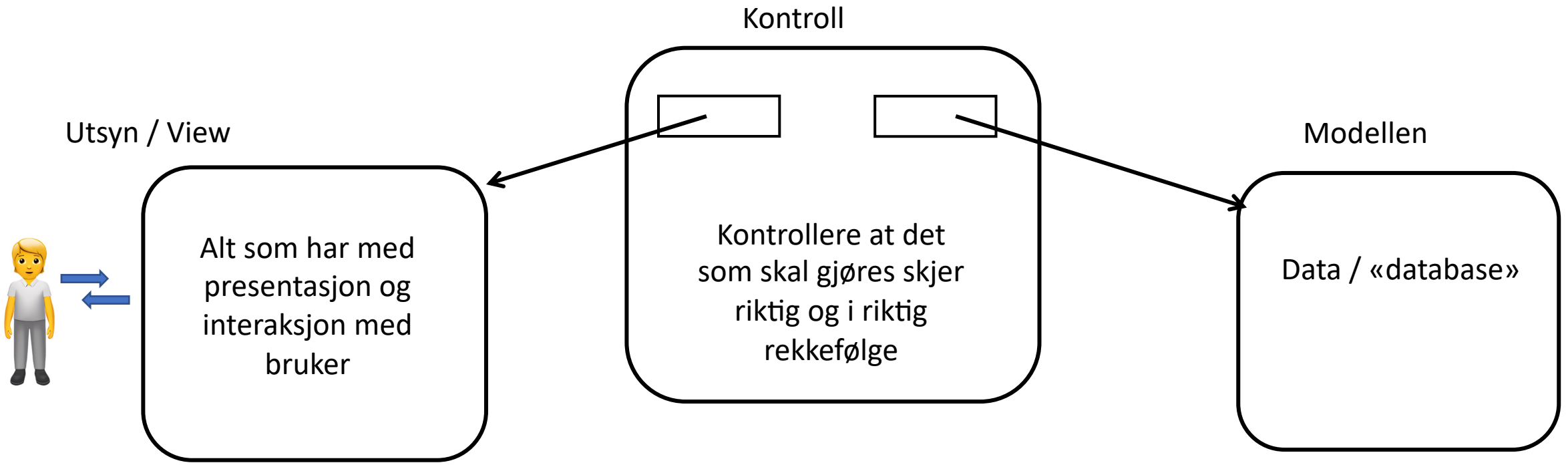
Model – View – Controller (MVC)

Del programmet opp i tre moduler / objekter:

- **Model**
  - lagrer det som trengs for å representere programmets tilstand - data
    I vårt program: Telleren
- **View (norsk: utsyn)**
  - Viser fram data og tar seg av brukerinteraksjonen (f.eks. knappetrykk).
- **Controller**
  - styrer det hele

Funnet opp av IFI-professoren Trygve Reenskaug

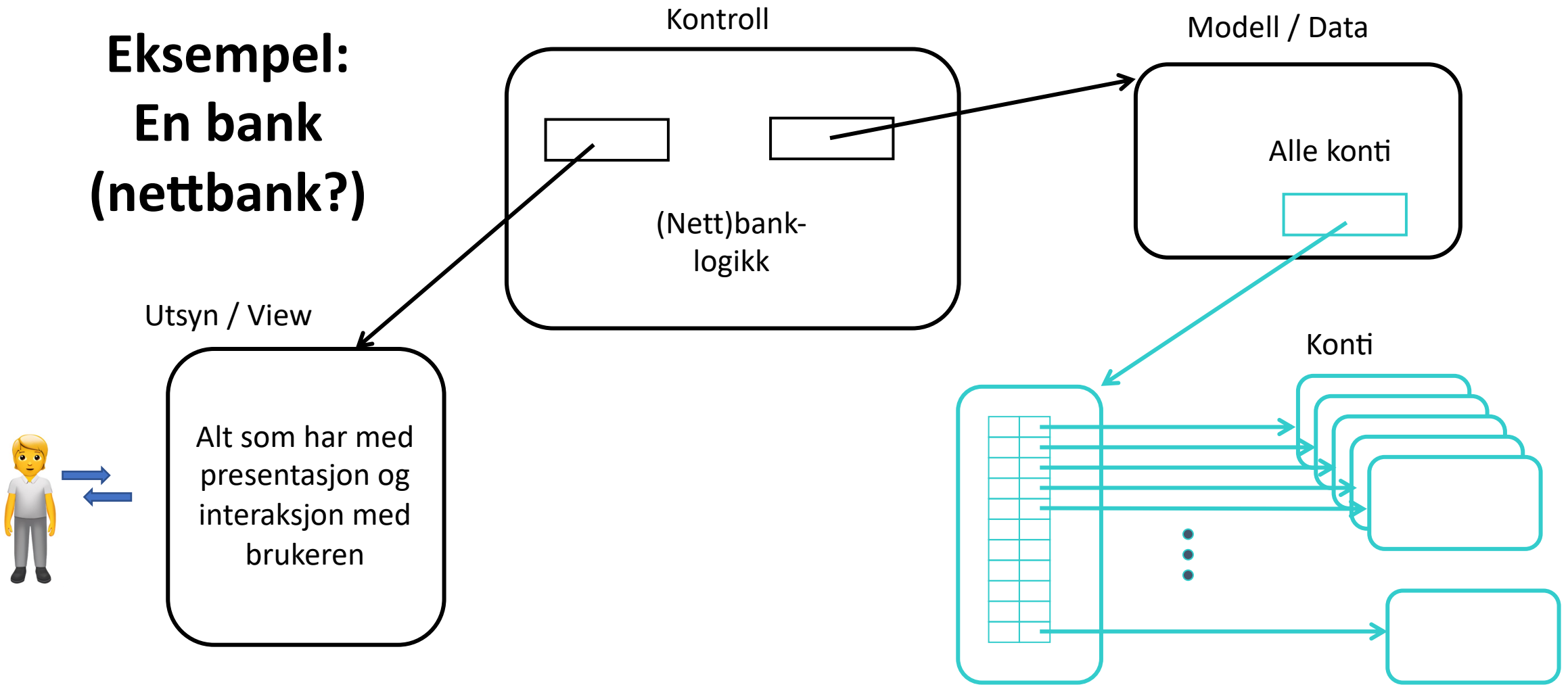
Programmeringsmønsteret Model – View – Controller (MVC)



Mange programmer kan deles opp på denne måten

Programmeringsmønsteret Model – View – Controller (MVC)

Eksempel: En bank (nettbank?)



Programmeringsmønsteret
Model – View – Controller (MVC)

Vårt tellerprogram

Kontroll

Kontrollere at det som skal gjøres skjer i riktig rekkefølge

Modell

public void oekTeller( )

public int hentVerdi( )

public void nullstilllTeller( )

Et **objekt** av klassen Teller

Utsyn / View

Alt som har med presentasjon og interaksjon med brukeren

MVC – struktur (med to forskjellige ”Views”)

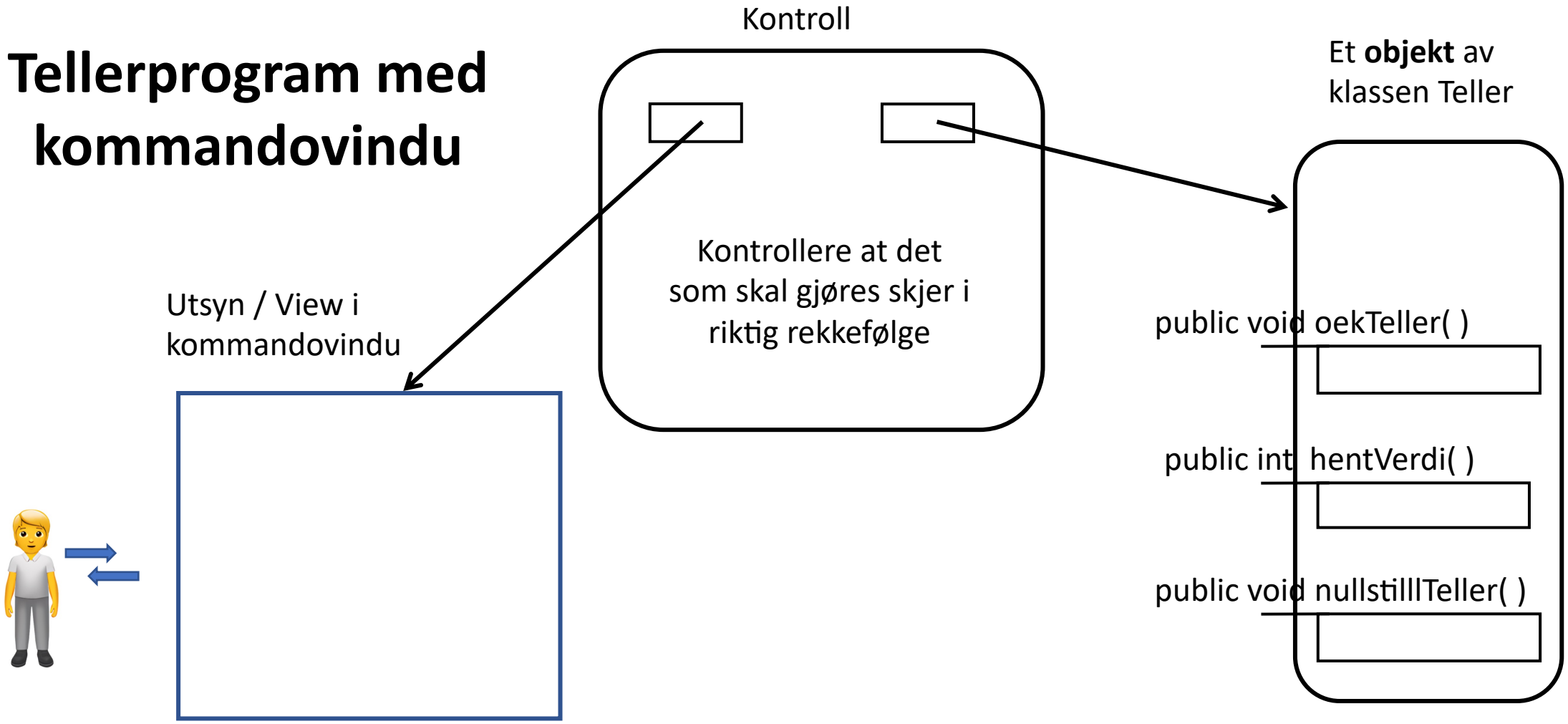
- Modellen lagrer data
  - Her “telleren”

1. View oppretter en dialog med brukeren i kommandovinduet eller
2. View oppretter et GUI-vindu og tar seg av interaksjon med brukeren

- Controller styrer det hele
  - Sørger for at ting skjer i riktig rekkeføle
  - Sørger for at knappetrykk får riktig effekt
  - Sørger for at modellen blir riktig avlest og oppdatert

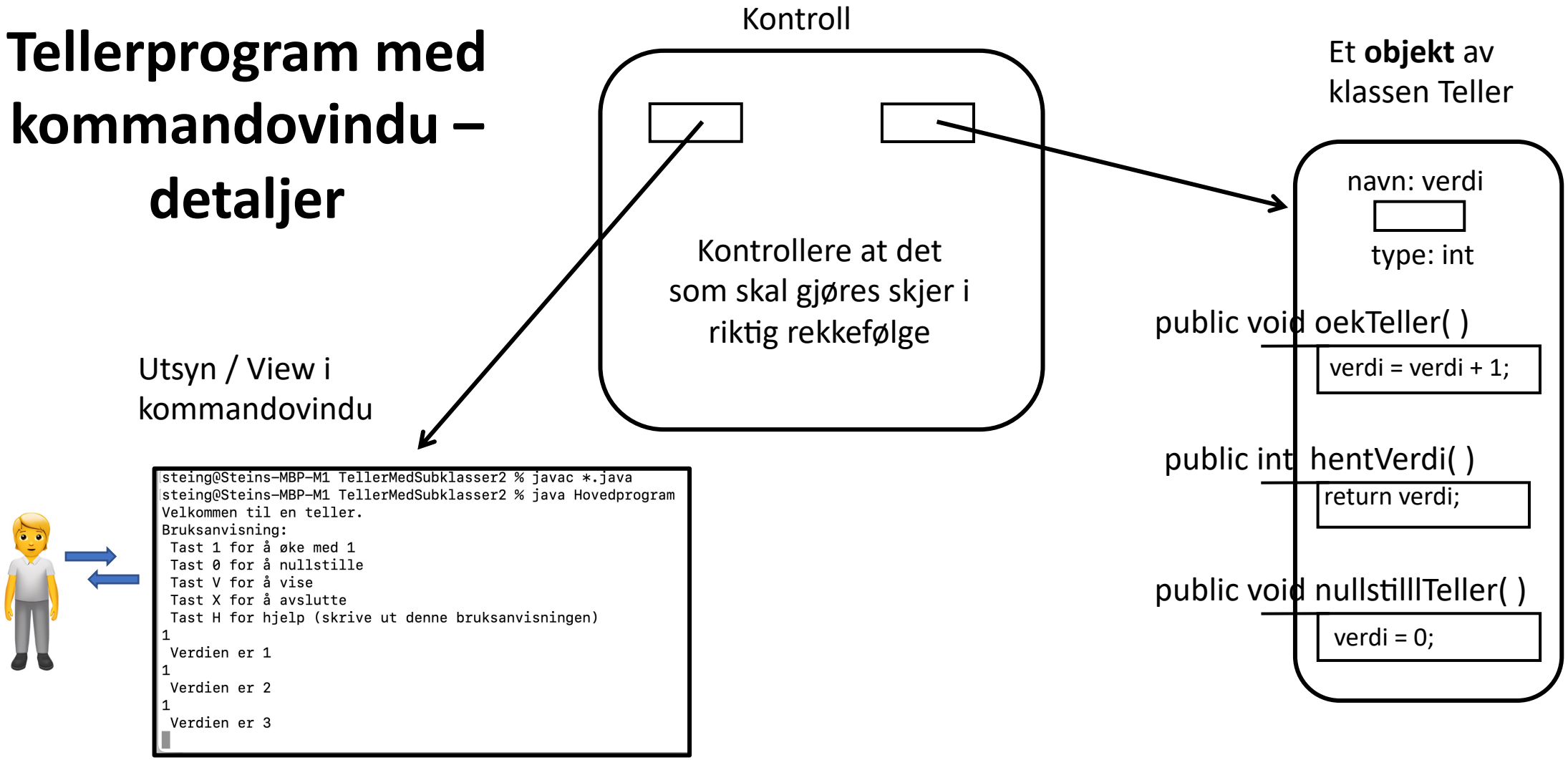
Programmeringsmønsteret
Model – View – Controller (MVC)

Tellerprogram med kommandovindu



Design av Teller-program med MVC

Tellerprogram med kommandovindu – detaljer

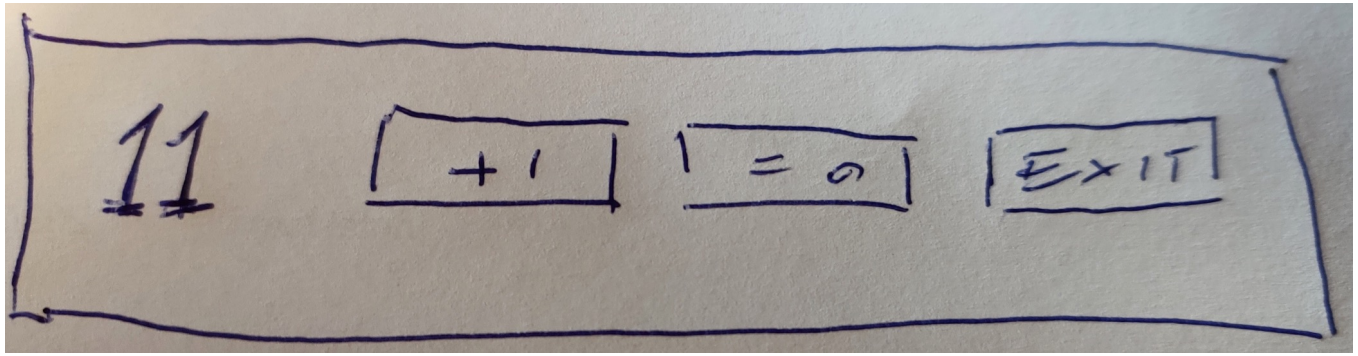


GUI- Design

Du bør legge mye arbeid i å designe skjermbildet fordi det er arbeidsomt å endre det senere:

- Hvilke GUI-elementer skal vi ha med?
- Hvordan skal de stå i forhold til hverandre?
- Det beste er å tegne for hånd:

Når vi skal lage et grafisk brukergrensesnitt

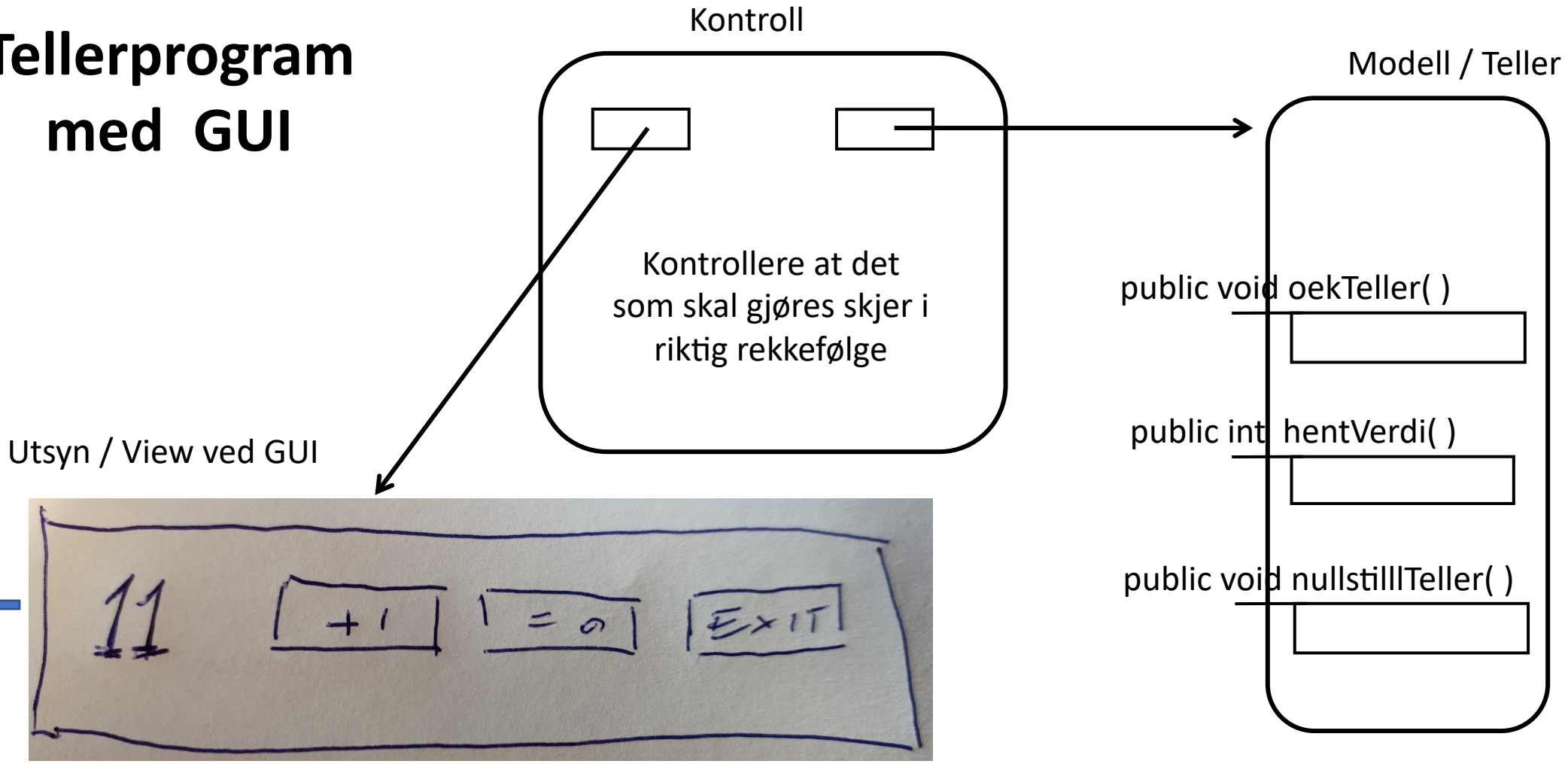


Tegn på papir først og vær sikker på at du lager vinduet slik du ønsker det. Litt tungvindt å forandre senere.

- Hvilke GUI-elementer skal være med
- Hvordan skal disse stå i forhold til hverandre

Design av Teller-program med MVC

Tellerprogram med GUI



Programmeringsmønsteret
Modell – View – Controller (MVC)

Tellerprogram med GUI – detaljer

Kontroll

Kontrollere at det som skal gjøres skjer i riktig rekkefølge

Et **objekt** av klassen Teller

navn: verdi

type: int

public void oekTeller( )

verdi = verdi + 1;

public int hentVerdi( )

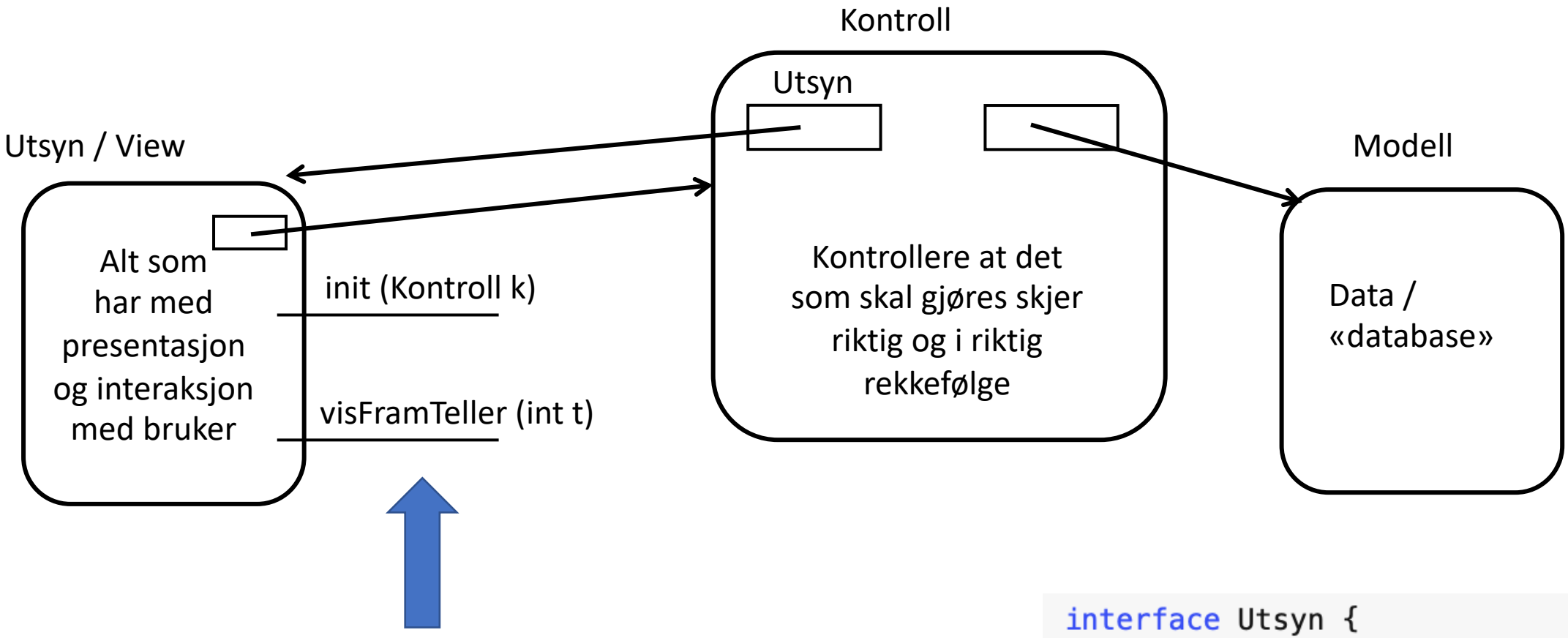
return verdi;

public void nullstilllTeller( )

verdi = 0;

Utsyn / View med GUI

Hva må kontrollen vite om View-et

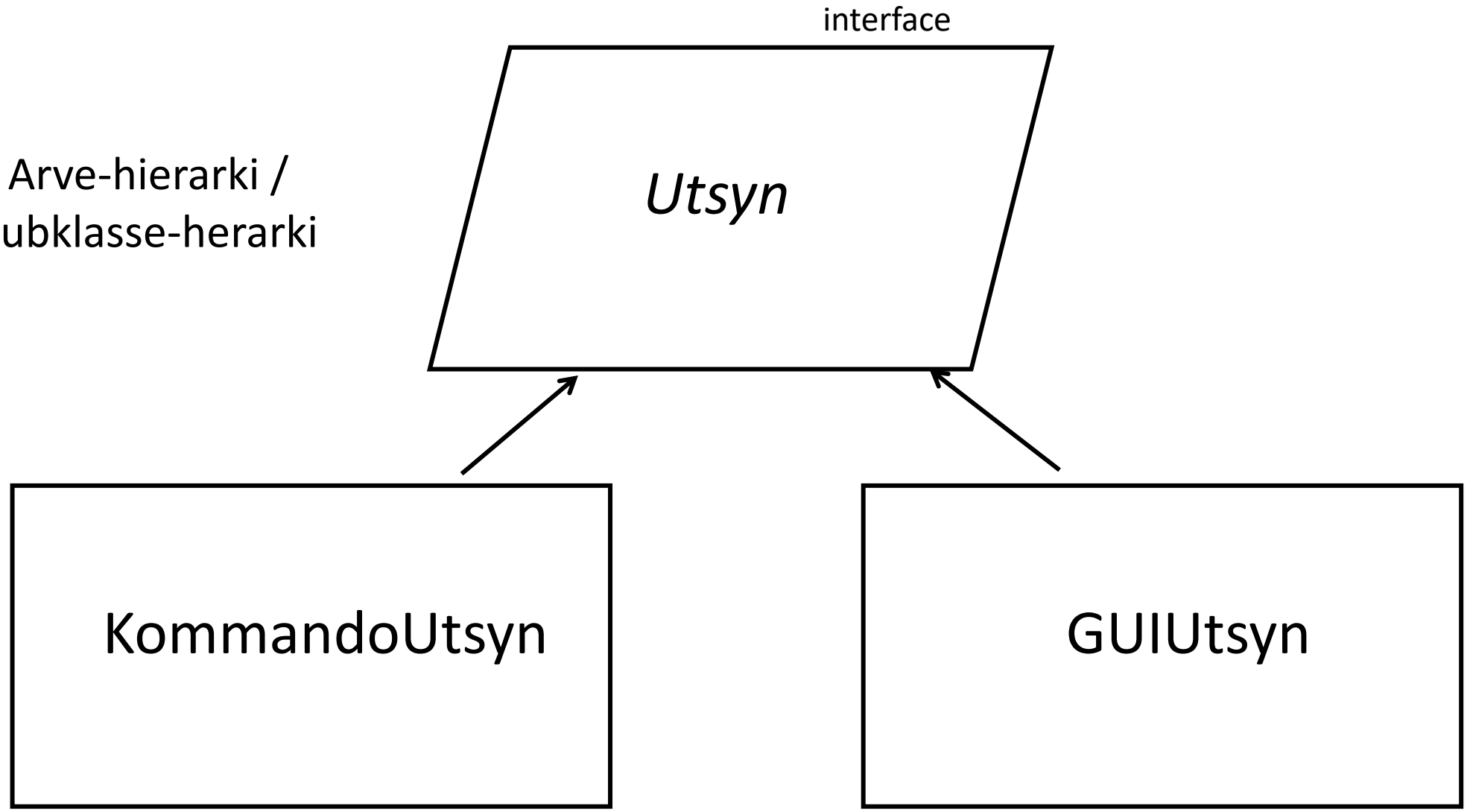


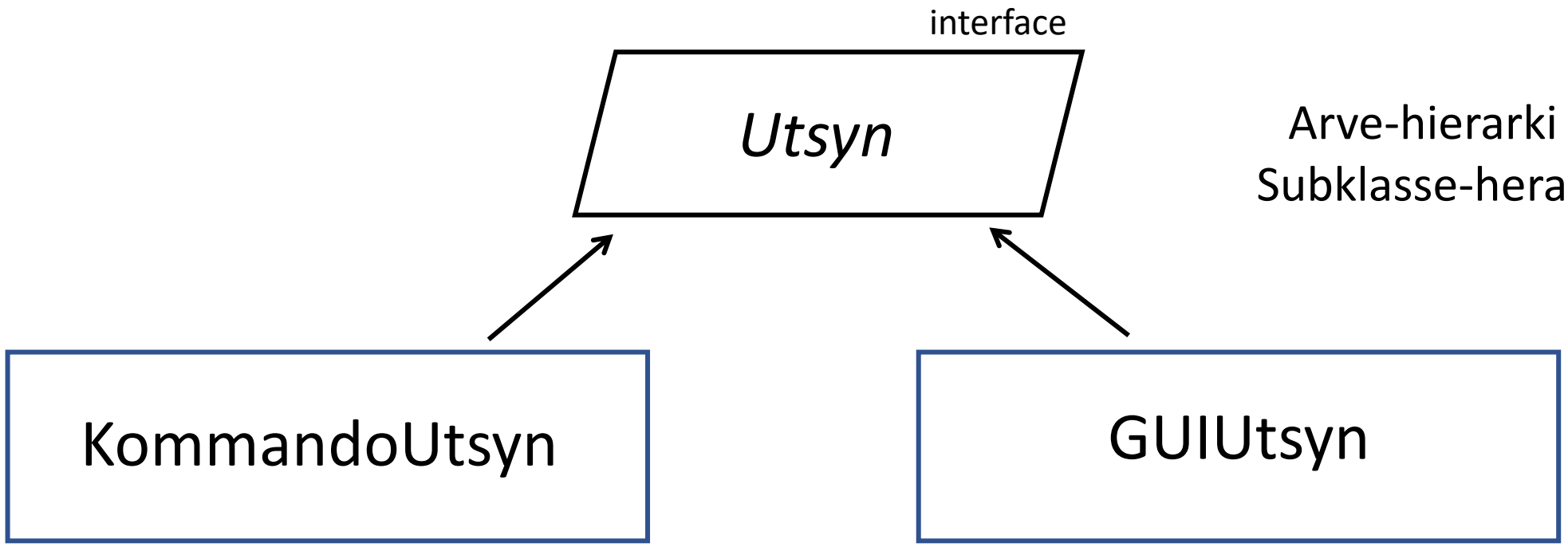
Dette kan beskrives av et interface:

```
interface Utsyn {
     void init(Kontroll k);
     void visFramTeller (int t);
}
```

To alternative implementasjoner

Arve-hierarki /
Subklasse-herarki





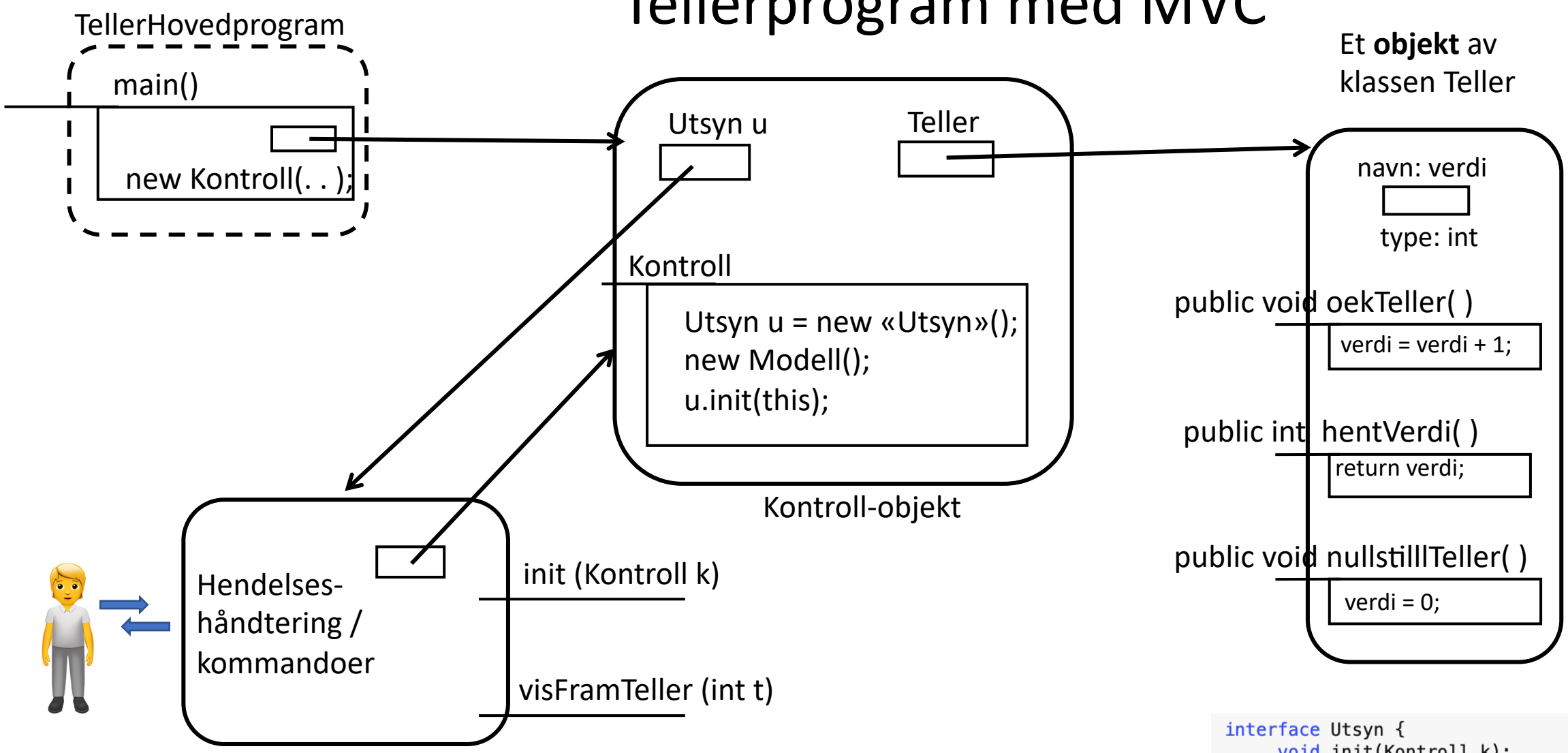
Arve-hierarki /
Subklasse-herarki

"Objekt av klassen KommandoUtsyn":

```
steing@Steins-MBP-M1 TellerMedSubklasser2 % javac *.java
steing@Steins-MBP-M1 TellerMedSubklasser2 % java Hovedprogram
Velkommen til en teller.
Bruksanvisning:
 Tast 1 for å øke med 1
 Tast 0 for å nullstille
 Tast V for å vise
 Tast X for å avslutte
 Tast H for hjelp (skrive ut denne bruksanvisningen)
1
 Verdien er 1
1
 Verdien er 2
1
 Verdien er 3
```

"Objekt av klassen GUIUtsyn":

Mer detaljert datastruktur i vårt Tellerprogram med MVC



Objekt som implementerer interfacet Utsyn (enten KommandoUtsyn eller GUIUtsyn):

```
interface Utsyn {
     void init(Kontroll k);
     void visFramTeller (int t);
}
```

Hele Modellen

```
class Modell {
    private int tellerverdi = 0;

    public int hentVerdi() {
        return tellerverdi;
    }

    public void oekTeller () {
        tellerverdi++;
    }

    public void nullstillTeller () {
        tellerverdi = 0;
    }
}
```

Legg merke til:

Ingen avhengigheter

(er seg selv nok)

Hele Kontrollen

Må kjenne til modellen og interfacet Utsyn

(og kunne lage objekter)

```
class Kontroll {
    private Utsyn ut;
    private Teller modell;
    Kontroll (char type) {
      if( type == 'K' || type == 'k')
           ut = new KommandoUtsyn();
           else  ut = new GUIUtsyn();
       modell = new Teller();
    }
    public void init() {
        ut.init(this);
    }
    void oekTeller () {
        modell.oekTeller();
        int x = modell.hentVerdi();
        ut.visFramTeller(x);
    }
    void nullstillTeller () {
        modell.nullstillTeller();
        int x = modell.hentVerdi();
        ut.visFramTeller(x);
    }
    public int hentVerdi() {
        return modell.hentVerdi();
    }
    void avslutt () {
        System.out.println(" Takk for at du brukte telleren");
        System.exit(0);
    }
}
```

Utsyn/View

interface

```
interface Utsyn {
      void init(Kontroll k);
      void visFramTeller (int t);
}
```

Dette er alt kontrollen trenger å vite !!

class KommandoUtsyn implements Utsyn

```
import java.util.Scanner;

class KommandoUtsyn implements Utsyn {
    private Kontroll kontroll;
    private Scanner innData;
    private void skrivBruksanvisning() {
        System.out.println("Bruksanvisning:");
        System.out.println(" Tast 1 for å øke med 1");
        System.out.println(" Tast 0 for å nullstille");
        System.out.println(" Tast V for å vise");
        System.out.println(" Tast X for å avslutte");
        System.out.println(" Tast H for hjelp (skrive ut denne bruksanvisningen)");
    }
```

fortsettes neste side

Resten av class KommandoUtsyn extends Utsyn

Kommandoløkka:

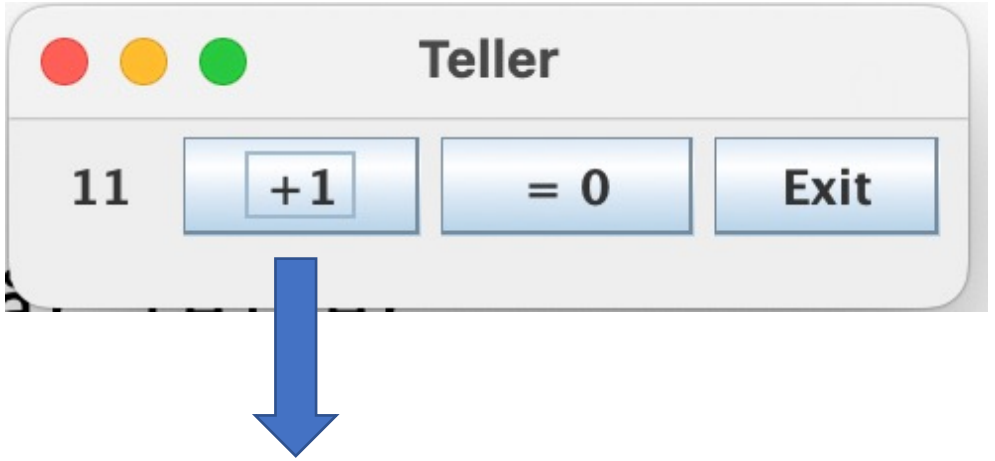
```
    @Override
    public void init(Kontroll k) {
        kontroll = k;
        boolean fortsett = true;
        innData = new Scanner(System.in);
        System.out.println("Velkommen til en teller.");
        skrivBruksanvisning();

        while (fortsett)  {
            String kommando = innData.nextLine();
            if (kommando.charAt(0) == '1') {
                kontroll.oekTeller();
            }
            else  if (kommando.substring(0,1).equals("0")) {
                kontroll.nullstillTeller();
            }
            else if (kommando.charAt(0) == 'V' || kommando.charAt(0) == 'v') {
                  int verdi = kontroll.hentVerdi();
                  visFramTeller(verdi);
            }
            else if (kommando.charAt(0) == 'X'|| kommando.charAt(0) == 'x') {
                  fortsett = false;
            }
             else if (kommando.charAt(0) == 'H' || kommando.charAt(0) == 'h') {
                  skrivBruksanvisning();
            }
        }
        kontroll.avslutt();
    }
    @Override
    public void visFramTeller (int t) {
          System.out.println(" Verdien er "+ t);
    }
}
```

class GUIUtsyn implements Utsyn

Start: Deklarasjoner

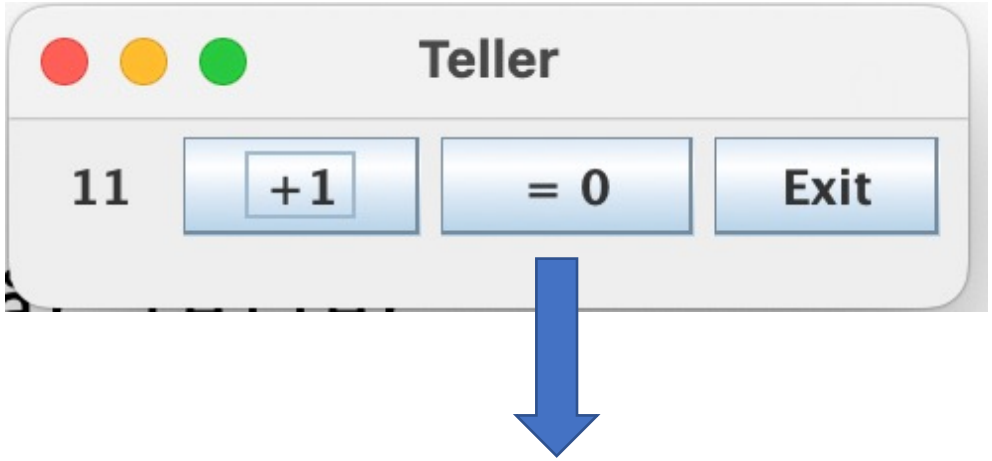
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class GUIUtsyn implements Utsyn {
    private Kontroll kontroll;
    private JButton tell, resett, slutt;
    private JFrame vindu;
    private JLabel antall;
    private JPanel panel;
```

Hendelseshåndterer 1:




```
class OekTeller implements ActionListener {
    @Override
     public void actionPerformed (ActionEvent e) {
            kontroll.oekTeller();
    }
}
```

Hendelseshåndterer 2:




```
class Nuller implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.nullstillTeller();
    }
}
```

Hendelseshåndterer 3:


```
}
class Stopper implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.avslutt();
    }
}
```

Inne i metoden

```
@Override
public void init(Kontroll k) {
```


Hele vinduet:

```
vindu = new JFrame("Teller");
vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Tegneflaten inne i vinduet:

```
panel = new JPanel();
vindu.add(panel);
```

Fortsatt inne i metoden init():

Lag de tre knappene og knyt dem opp mot sine hendelseshåndterere

```
tell = new JButton(" +1 ");
tell.addActionListener(new OekTeller());
```

```
resett = new JButton(" = 0 ");
resett.addActionListener(new Nuller());
```

```
slutt = new JButton("Exit");
slutt.addActionListener(new Stopper());
```

Fortsatt inne i metoden init():

Lag et felt for tellerinformasjon:

```
antall = new JLabel("  0  ");
```

Legg alle komponentene på tegneflaten og vis dem fram:

```
        panel.add(antall);  panel.add(tell);
        panel.add(resett);  panel.add(slutt);

        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);

    }
```

Slutt på metoden init()
Slutt på hele GUIUtsyn-klassen

Hovedprogram: Oppretter et Kontroll-objekt og kaller init() på det

```
class TellerHovedprogram {
    public static void main (String[] arg) {
        if(arg.length == 0) {
            System.out.println("          Dette programmet må startes   enten slik:");
            System.out.println("        > java TellerHovedprogram K     eller slik:");
            System.out.println("        > java TellerHovedprogram G         ");
            System.out.println();
        } else {
             Kontroll kontroll = new Kontroll(arg[0].charAt(0));
             kontroll.init();
        }
    }
}
```

Alle programmene ligger som ressurser uke 12

Lærdom (i tillegg til MVC og GUI):

- Mange objekter trenger bare kjenne til interfacet / grensesnittet til andre objekter

- Ofte må vi lage objektene først og deretter kalle init(. . .) for å koble dem sammen.

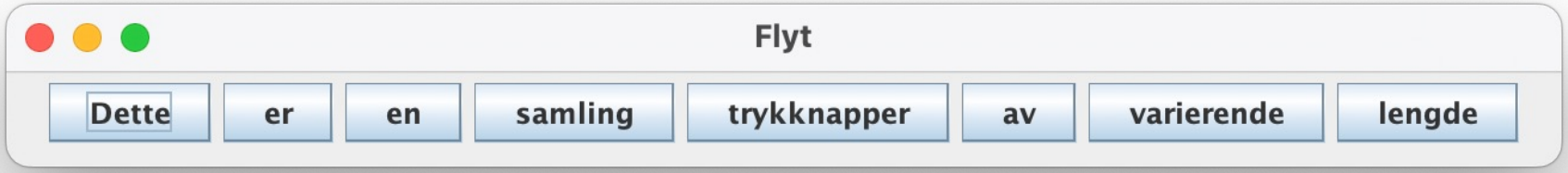
Mer GUI – form og farge

Siden alle klassene vi bruker er subklasser av JComponent, har de samme metoder for å endre:

- plassering i tegnepanelet
- størrelse
- font
- farge (både tekst og bakgrunn)
- ramme

Plassering på tegneflaten

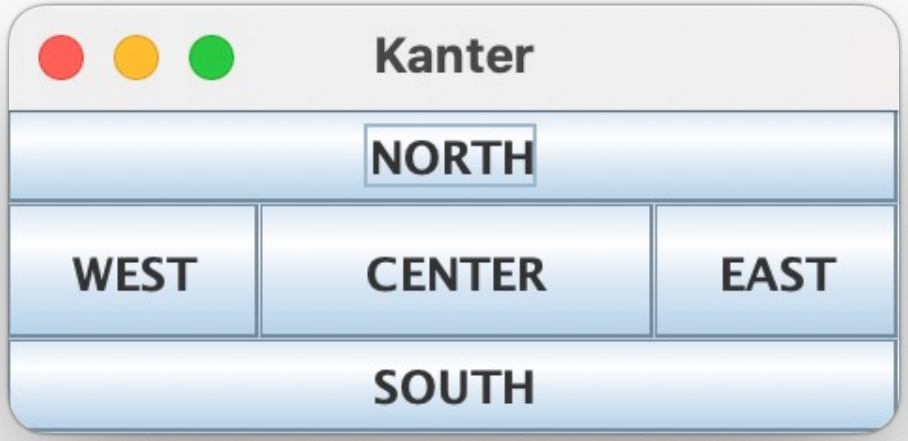
- For alle tegneflater (dvs JPanel) kan vi angi hvordan innholdet skal plasseres.
- Standard er **FlowLayout**
- Standard er at elementene kommer etter hverandre som når vi skriver ordene i en tekst. Ved endring av vinduet, vil linjedelingen kunne bli anderledes:



BorderLayout

- Ofte ønsker man å plassere elementene i faste posisjoner:

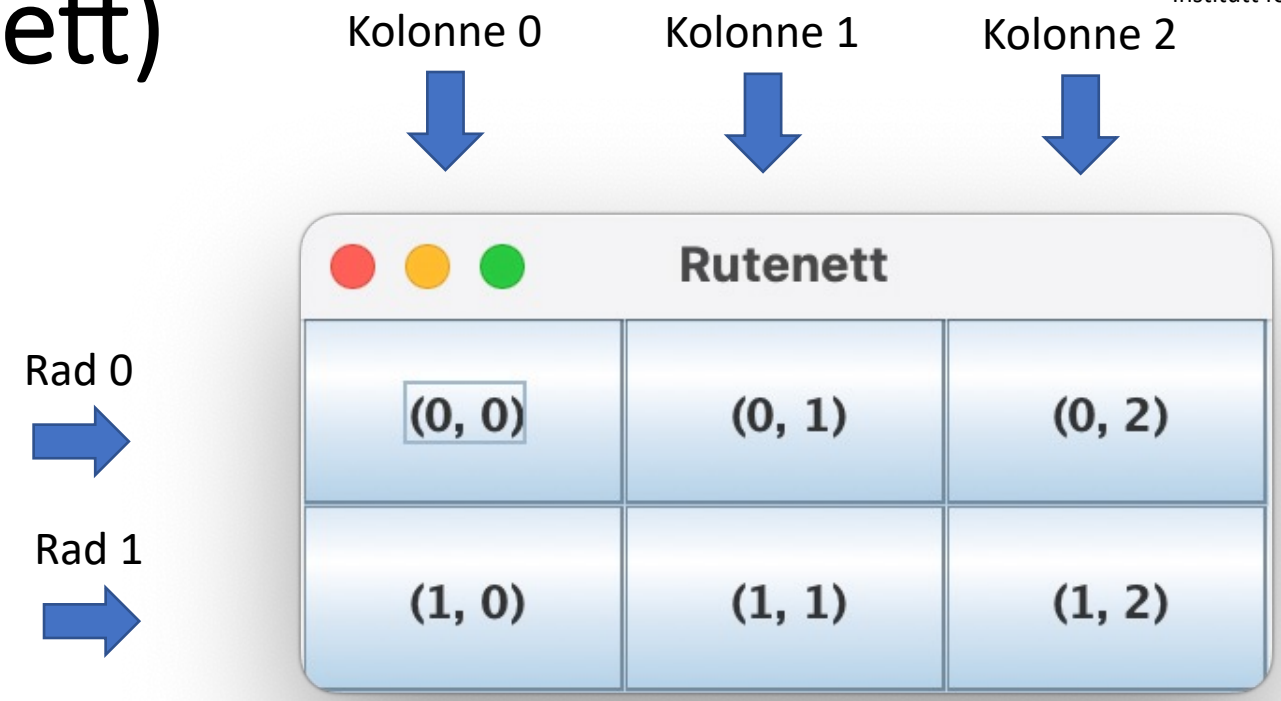
```
JPanel panel = new JPanel();
vindu.add(panel);

panel.setLayout(new BorderLayout());
panel.add(new JButton("NORTH"), BorderLayout.NORTH);
panel.add(new JButton("SOUTH"), BorderLayout.SOUTH);
panel.add(new JButton("EAST") , BorderLayout.EAST);
panel.add(new JButton("WEST"), BorderLayout.WEST);
panel.add(new JButton("CENTER"), BorderLayout.CENTER);
```



Grid Layout (Rutenett)

- Noen ganger ønsker man et rutenett:

```
JPanel panel = new JPanel();
vindu.add(panel);


panel.setLayout(new GridLayout(2,3));
panel.add(new JButton("(0, 0)"));
panel.add(new JButton("(0, 1)"));
panel.add(new JButton("(0, 2)"));
panel.add(new JButton("(1, 0)"));
panel.add(new JButton("(1, 1)"));
panel.add(new JButton("(1, 2)"))
```

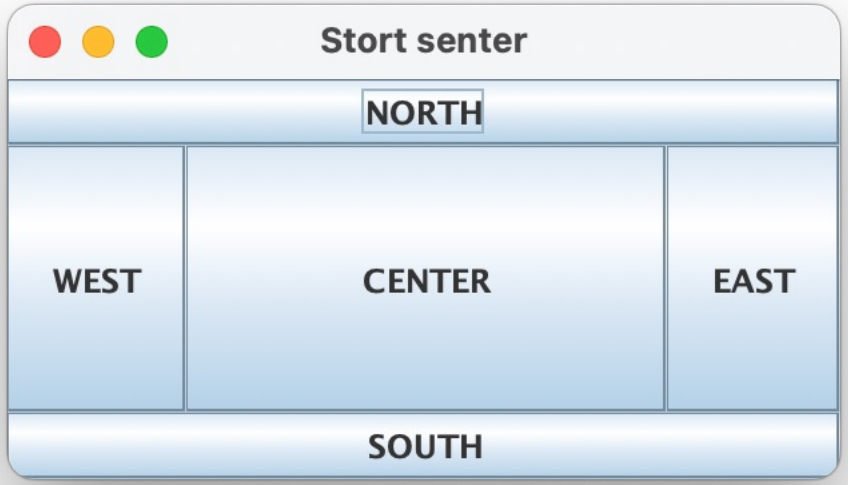


Husk: (RadNummer, KolonneNummer)

Størrelser

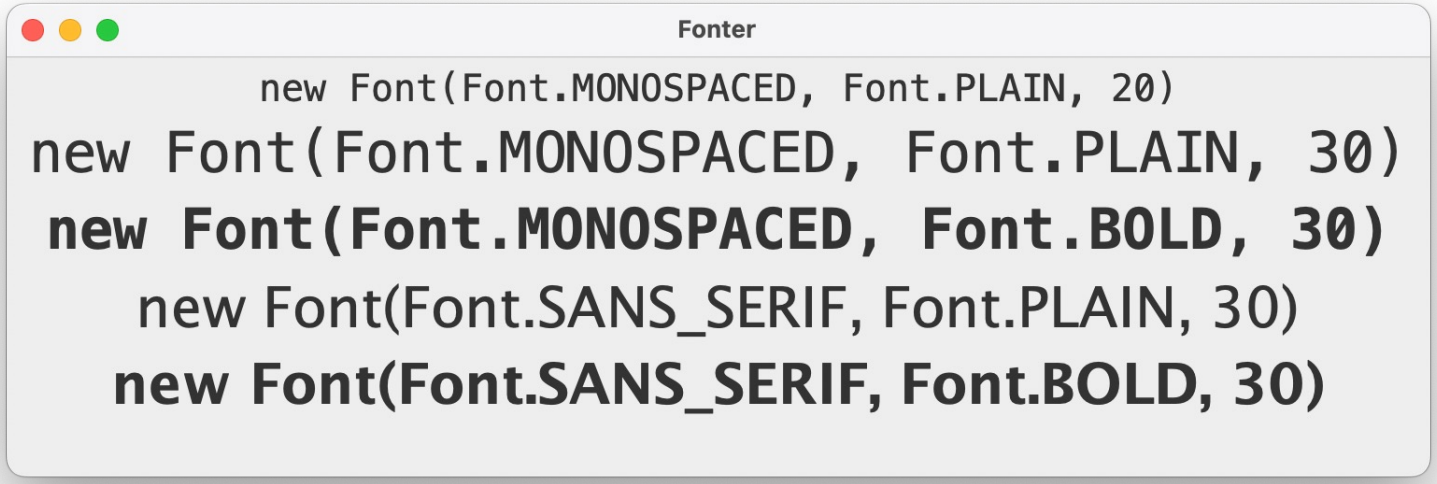
- I utgangspunktet prøver Swing å pakke elementene så tett som mulig.
  Vi kan hinte om vi ønsker noe anderledes:

```
panel.setLayout(new BorderLayout());
panel.add(new JButton("NORTH"), BorderLayout.NORTH);
panel.add(new JButton("SOUTH"), BorderLayout.SOUTH);
panel.add(new JButton("EAST") , BorderLayout.EAST);
panel.add(new JButton("WEST"), BorderLayout.WEST);

JButton sentrum = new JButton("CENTER");
sentrum.setPreferredSize(new Dimension(180,100));
panel.add(sentrum, BorderLayout.CENTER)
```



Fonter

- Fonten endres med ***xxx*.setFont(new Font(. . .))**.



Enhenten for størrelse i Swing er alltid *skjermpunkter* («pixels»).

Farger på bakgrunn eller tekst

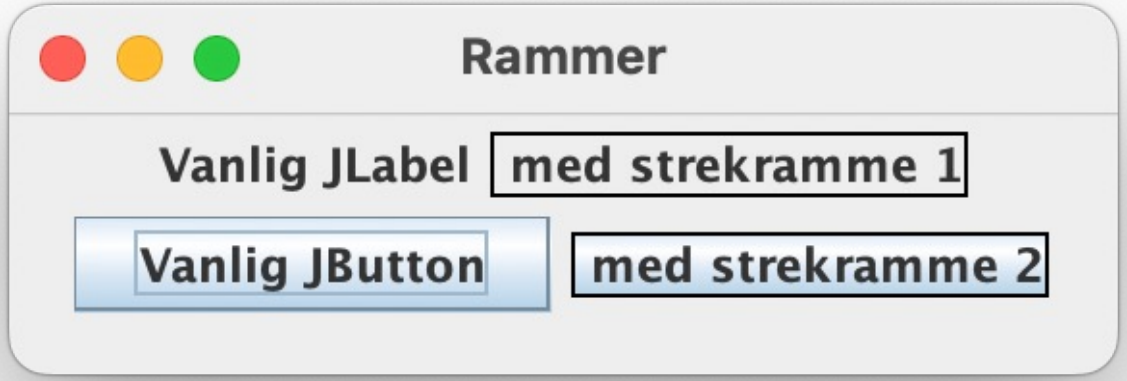
f.eks.

```
knapp1.setBackground(Color.LIGHT_GRAY);
knapp2.setBackground(new Color(255,255,0)/*gul*/);  // Rød-Grønn-Blå-koding
```


```
JLabel green = new JLabel("xxx.setForeground(Color.GREEN)");
green.setForeground(Color.GREEN);
green.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30)); panel.add(green);
```

Rammer

- Det finnes ca 15 rammer å velge blant, som **setBorder(BorderFactory.createLineBorder(Color.BLACK))** og tilsvarende.



f.eks.

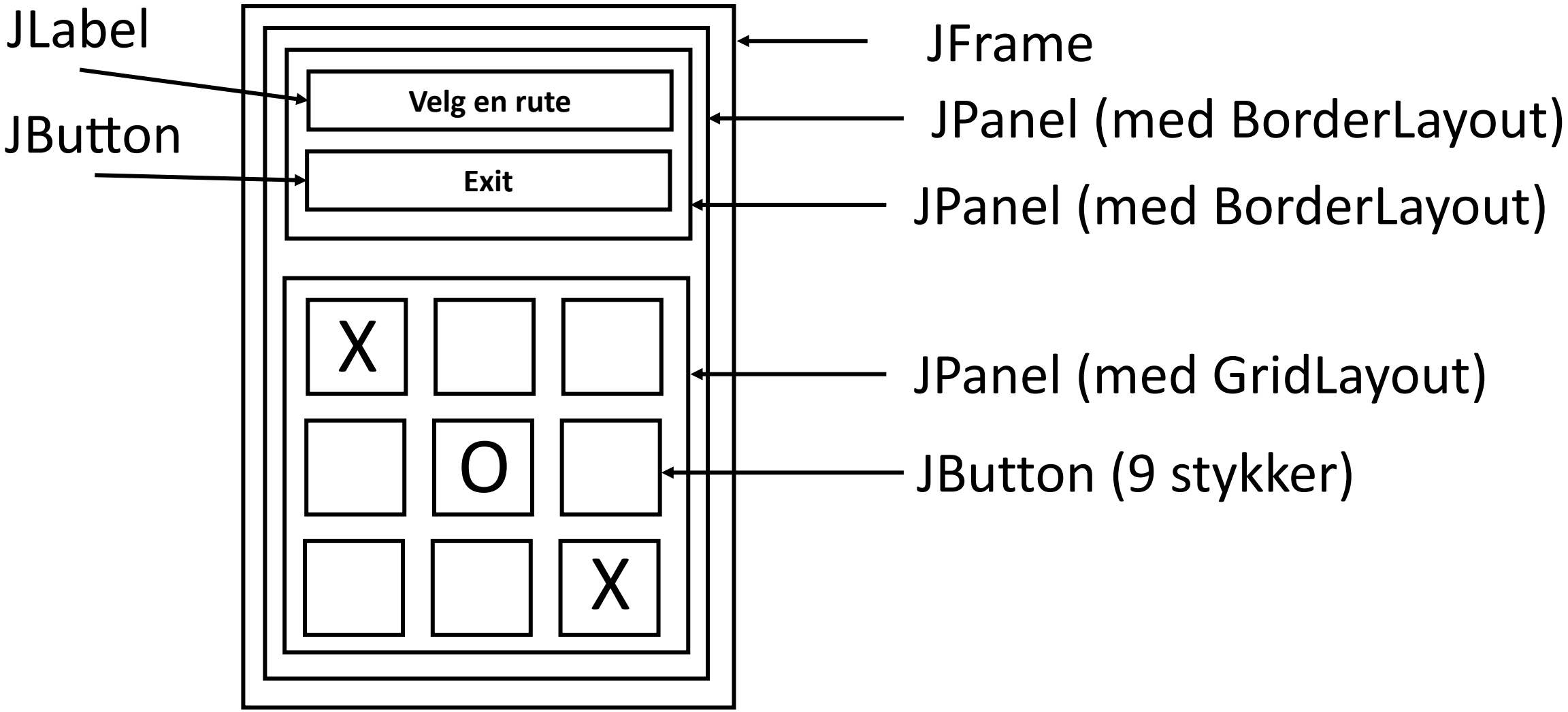
```
JLabel egenLabel = new JLabel(" med strekramme 1");
egenLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
panel.add(egenLabel);
```

Dag lovet dere: Tripp – Trapp – Tresko

Rekkefølgen når man skal skrive et GUI-program:

1. Planlegge utseende til vinduet
2. Designe programmet: F.eks. MVC
3. Planlegge brukerdialogen / hva skjer ved knappetrykk mm.
4. Programmere det hele

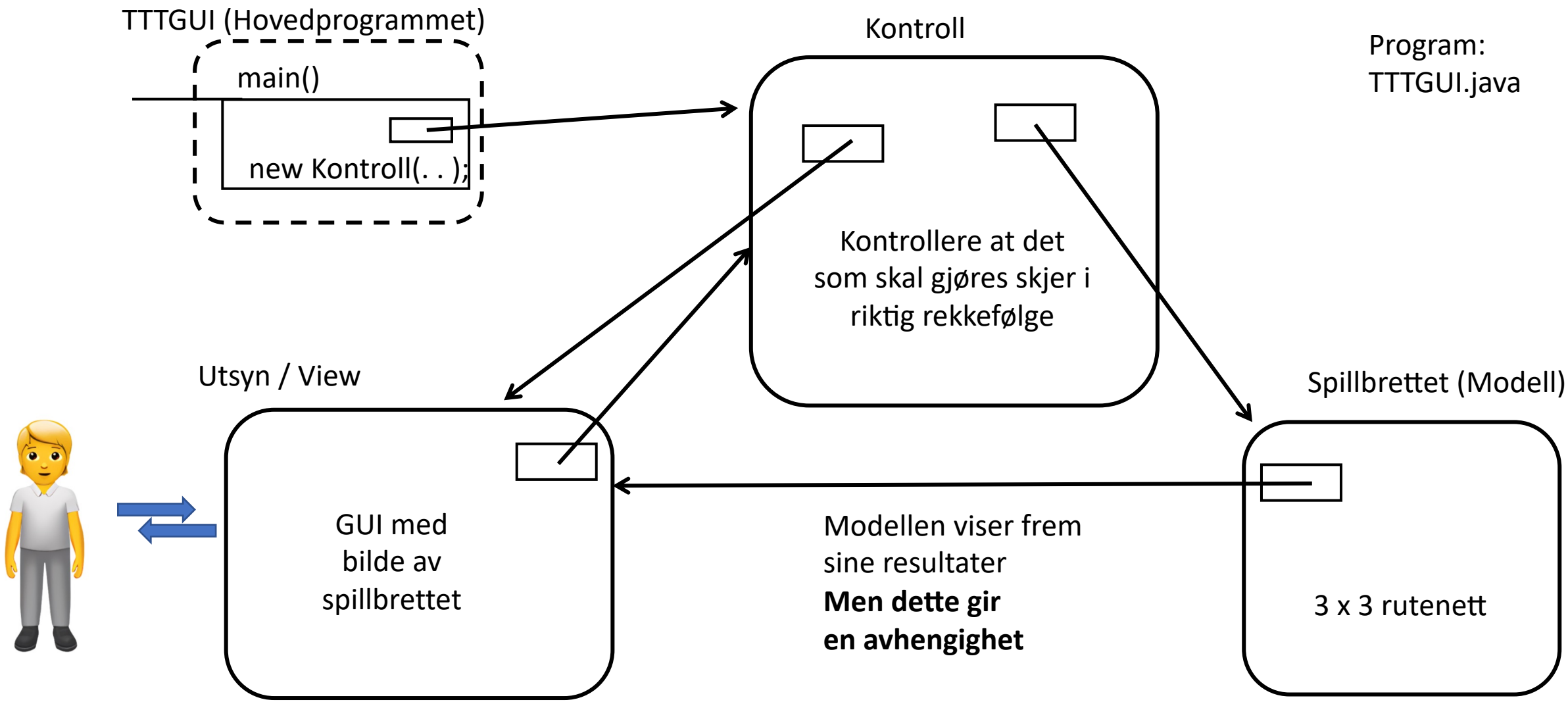
Tripp Trapp Tresko - Vindudesign



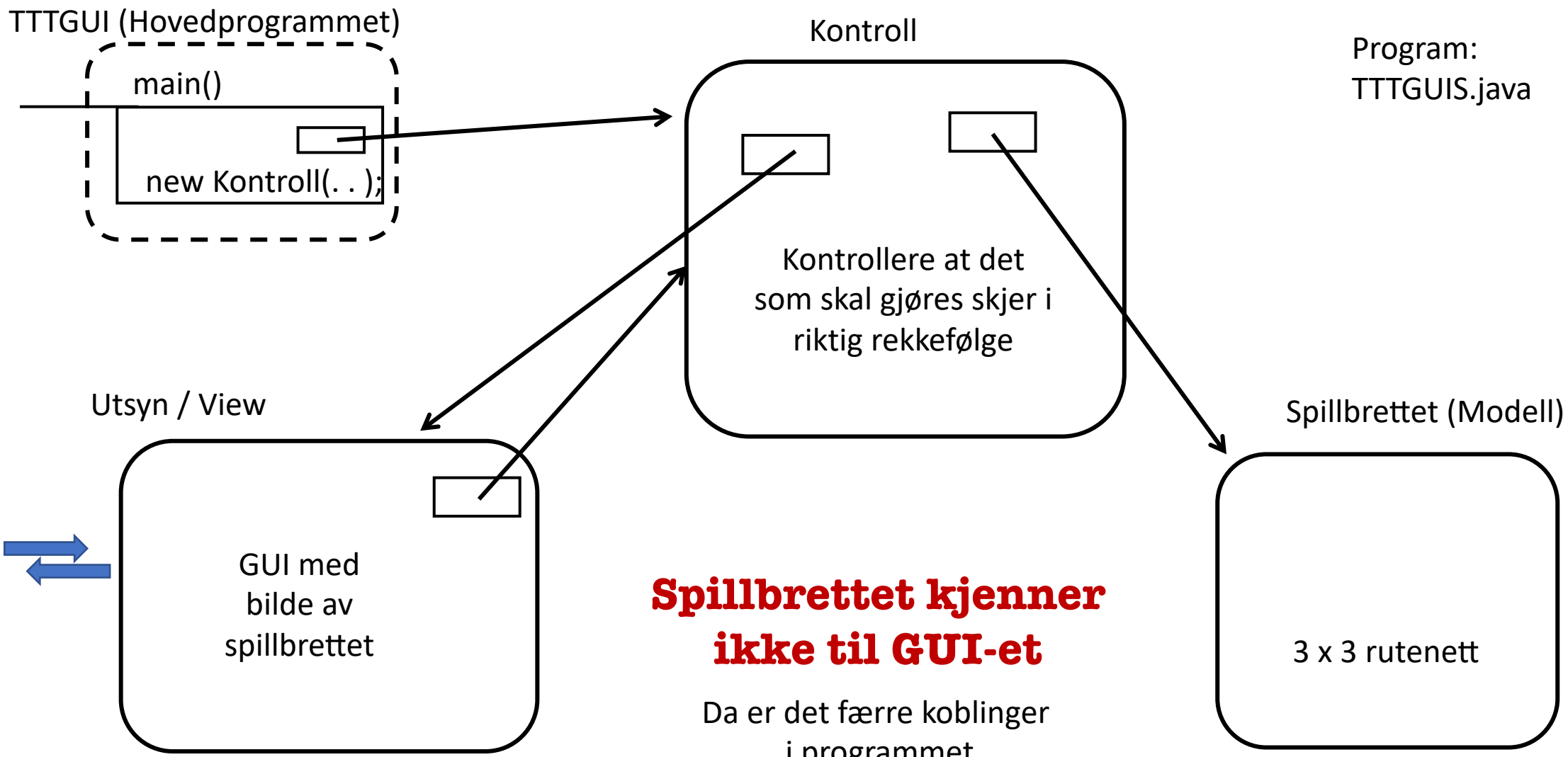
MVC-struktur

- Modellen er brettet med 3 x 3 ruter
  - Hvilke operasjoner trenger vi
    - Hva slags datastruktur skal vi bruke?
    - char[][] brett = new char[3][3];
- View er brettet tegnet ut (forrige side)
- Kontroll:
  - Hvem sin tur er det
  - Reaksjon ved ulovlige trekk
  - . . .

Tripp – Trapp – Tresko (MVC)



TTT - Refaktorert



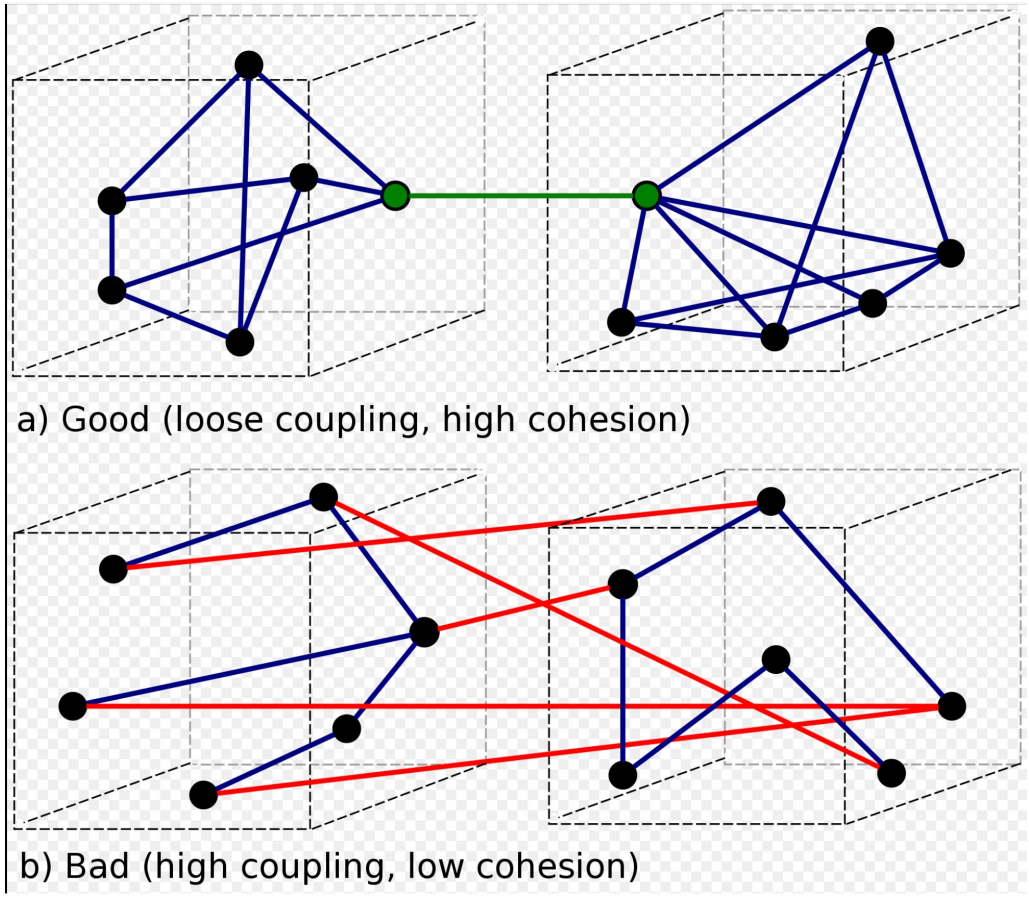
Spillbrettet kjenner ikke til GUI-et

Da er det færre koblinger i programmet

Kobling mellom objekter (og samhold)

Et objekt skal være avhengig av / være koblet til færrest mulige andre objekter.

På denne måten kan objektet brukes i flest mulig andre sammenhenger og er mindre avhengig av forandringer i andre objekter.

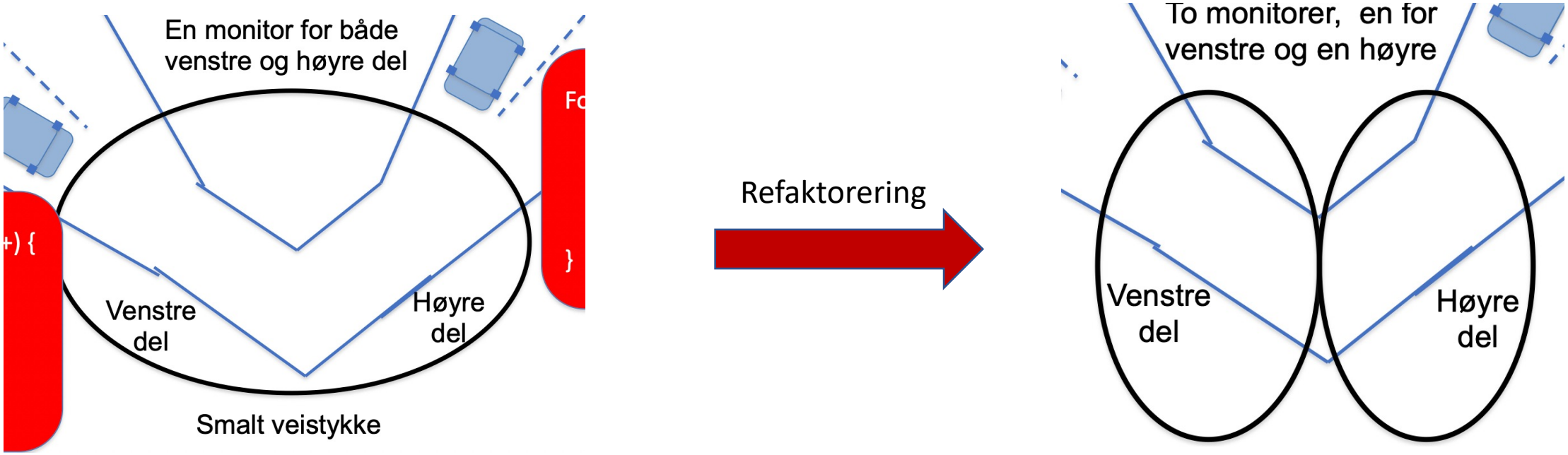


Wikipedia: Coupling (computer programming)
Wikipedia: Cohesion (computer programming)

Cohesion ("samhold"):
Egenskaper / data som hører sammen og er avhengig av hverandre skal innkapsles i de sammen objektene

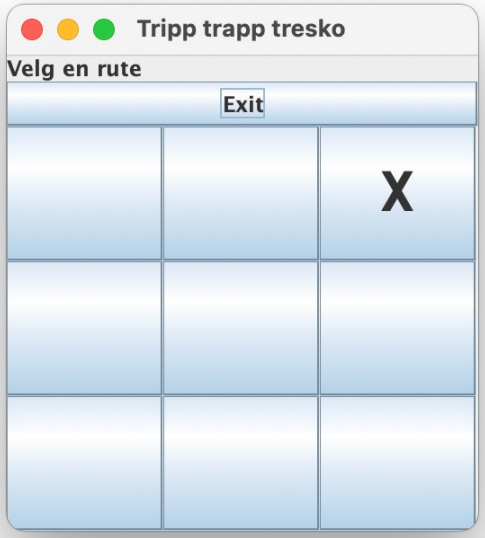
Og motsatt:
Egenskaper / data som ikke hører sammen og som ikke er avhengig av hverandre skal ikke inne i de samme objektene

Husker dere fra siste tråd-forelesning:



Fordi i klassen/monitoren til venste var det lavt samhold (cohesion) mellom de delene av objektet som definerte hver sin side.
Ved å splitte klassen i to fikk vi objekter med sterkere samhold innad.

TTTGUI-programmet



- To versjoner:
  - TTTGUI.java og TTTGUIS.java
  - Begge ligger på IN1010-siden

- Indre klasser med hendelseshåndtering er deklarert forskjellige steder
- I TTTGUI.java kaller Modellen på metoder i GUI-et
  - Det betyr at det er en kobling mellom Modellen og GUI-et (View-et)
  - Dette gjør programmeringen av Kontrollen enklere
- I TTTGUIS.java kaller Modellen ikke på metoder i GUI-et
  - Modellen har ingen peker til GUI-et
  - **Modellen er uavhengig av metoder i GUI-et**
  - -> Det er lav kobling (ingen kobling) mellom Modellen og GUI
  - Men da må Kontollen sørge for riktige kall til GUI-et

Når ikke annet er skrevet er utskriftene fra TTTGUIS.java

Starten på modellen

```
class Modell {
    // Brettet innholder
    // - blank der det er ledig (initialverdier)
    // - X der maskinen har satt
    // - O der brukeren har satt
    char[][] brett = new char[3][3];
    int antTrekk = 0;
    boolean spilletErFerdig = false;
    Modell () {
        for (int rx = 0;  rx < 3;  ++rx)
            for (int kx = 0;  kx < 3;  ++kx)
                brett[rx][kx] = ' ';
    }
```

Diverse operasjoner i modellen uten kobling til GUI-et

```
boolean lovligTrekk (int r, int k) {
    return brett[r][k] == ' ';
}
void noterTrekk (int r, int k, char spiller) {
    brett[r][k] = spiller;
    ++antTrekk;
}
void noterVinner (String vinner) {
    spilletErFerdig = true;
}
void noterUavgjort () {
    spilletErFerdig = true;
}
boolean erSpilletFerdig () {
    return spilletErFerdig;
}
boolean erBrettetFullt () {
    return antTrekk == 9;
}
```

Diverse operasjoner i modellen med kobling til GUI-et

(fra TTTGUI.java)

```
boolean lovligTrekk (int r, int k) {
    return brett[r][k] == ' ';
}

void noterTrekk (int r, int k, char spiller) {
    brett[r][k] = spiller;
    ++antTrekk;
    gui.markerTrekk(r, k, spiller);
}

void noterVinner (String vinner) {
    spilletErFerdig = true;
    gui.visStatus(vinner + " vant");
}

void noterUavgjort () {
    spilletErFerdig = true;
    gui.visStatus("Uavgjort");
}

boolean erSpilletFerdig () {
    return spilletErFerdig;
}

boolean erBrettetFullt () {
    return antTrekk == 9;
}
```

Tjeneste i modellen

Slutten av modellen

```
    boolean harVunnet (char s) {
        return
        // Sjekk radene
        brett[0][0]==s && brett[0][1]==s && brett[0][2]==s ||
        brett[1][0]==s && brett[1][1]==s && brett[1][2]==s ||
        brett[2][0]==s && brett[2][1]==s && brett[2][2]==s ||
        // Sjekk kolonnene
        brett[0][0]==s && brett[1][0]==s && brett[2][0]==s ||
        brett[0][1]==s && brett[1][1]==s && brett[2][1]==s ||
        brett[0][2]==s && brett[1][2]==s && brett[2][2]==s ||
        // Sjekk diagonalene
        brett[0][0]==s && brett[1][1]==s && brett[2][2]==s ||
        brett[0][2]==s && brett[1][1]==s && brett[2][0]==s;
    }
}
```

Kontrollen

Start

```
class Kontroll {
    GUI gui;
    Modell modell;

    Kontroll () {
        gui = new GUI(this);
        modell = new Modell();
    }
    void startSpillet () {
        laMaskinenTrekke();
    }
    void avsluttSpillet () {
        System.exit(0);
    }
```

Eksempel på operasjon i kontrollen når den ikke må kontakte GUI-et fordi det har modellen gjort

```
void laMaskinenTrekke () {
    int r, k;
    do {
        r = trekk(0,2);  k = trekk(0,2);
    } while (! modell.lovligTrekk(r,k));
    modell.noterTrekk(r, k, 'X');
    if (modell.harVunnet('X'))
        modell.noterVinner("Maskinen");
}
```

(fra TTTGUI.java)

Eksempel på operasjon i kontrollen når den må kontakte GUI-et

```
    void laMaskinenTrekke () {
        int r, k;
        do {
            r = trekk(0,2);  k = trekk(0,2);
        } while (! modell.lovligTrekk(r,k));
        modell.noterTrekk(r, k, 'X');
        gui.markerTrekk(r, k, 'X');
        if (modell.harVunnet('X')) {
            modell.noterVinner("Maskinen");
            gui.visStatus('X' + " vant");
        }
    }
```

Kontrollen: Håndtering av brukerens trekk

```
void brukervalg (int r, int k) {
    if (modell.erSpilletFerdig()) {
        gui.visStatus("Ingen flere trekk!");
        return;
    }
    if (! modell.lovligTrekk(r, k)) {
        gui.visStatus("Ulovlig trekk!");
        return;
    }
    modell.noterTrekk(r, k, 'O');
    if (modell.harVunnet('O')) {
        modell.noterVinner("Du");
        return;
    }
    laMaskinenTrekke();
    if (modell.erSpilletFerdig()) {
        return;
    }
    if (modell.erBrettetFullt()) {
        modell.noterUavgjort();
    } else {
        gui.visStatus("Velg en rute");
    }
}
```

GUI (View-et)

GUI-kontruktør del 1

```
class GUI {
    Kontroll kontroll;
    JFrame vindu;
    JPanel panel, konsoll, rutenett;
    JButton[][] ruter = new JButton[3][3];
    JLabel status;
    JButton stoppknapp;

    GUI (Kontroll k) {
        kontroll = k;

        try {
            UIManager.setLookAndFeel(
             UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(9); }

        vindu = new JFrame("Tripp trapp tresko");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

GUI-kontruktør del 2:

Sett opp hovedtegnepanel og konsoll med statusinformasjon, **stoppknapp** og **rutenett**
og gjøre vinduet synlig

```
        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        vindu.add(panel);

        konsoll = new JPanel();
        konsoll.setLayout(new BorderLayout());
        panel.add(konsoll, BorderLayout.NORTH);

        status = new JLabel("Velg en rute");
        konsoll.add(status, BorderLayout.NORTH);

        stoppknapp = new JButton("Exit");
        stoppknapp.addActionListener(new Stoppbehandler());
        konsoll.add(stoppknapp, BorderLayout.SOUTH);

        rutenett = new JPanel();
        rutenett.setLayout(new GridLayout(3,3));
        for (int rx = 0;  rx < 3;  ++rx) {
            for (int kx = 0;  kx < 3;  ++kx) {
                JButton b = new JButton(" ");
                ruter[rx][kx] = b;
                b.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));
                b.addActionListener(new Spillvelger(rx,kx));
                rutenett.add(b);
            }
        }
        panel.add(rutenett, BorderLayout.CENTER);

        vindu.pack();
        vindu.setVisible(true);
    }
```

Hendelses-håndtering

```
class Stoppbehandler implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.avsluttSpillet();
    }
}

class Spillvelger implements ActionListener {
    int rad, kol;
    Spillvelger (int r, int k) {
        rad = r;  kol = k;
    }
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.brukervalg(rad, kol);
    }
}
```

Hovedprogrammet


```
class TTTGUI {
    public static void main (String[] arg) {
    Kontroll kontroll = new Kontroll();
    kontroll.startSpillet();
    }
}
```

Disse klassene og metodene trenger man i IN1010 (Se også Horstmann (læreboka))

ActionEvent

interface ActionListener

- actionPerformed()

BorderFactory

- createLineBorder(Color c)

BorderLayout ext LayoutManager

- String CENTER, EAST, NORTH, SOUTH, WEST

Color

- Color BLACK, BLUE, ...

Dimension

FlowLayout ext LayoutManager

Font

- int BOLD, PLAIN
- String MONOSPACED, SANS_SERIF

GridLayout ext LayoutManager

JButton ext JComponent

- addActionListener (ActionListener a)

JComponent

- setBackground(Color c)
- setFont (Font f)
- setForeground(Color c)
- setOpaque(boolean b)
- setText (String t)

JFrame

- add (JComponent c)
- int EXIT_ON_CLOSE
- pack()
- setDefaultCloseOperation (int i)
- setVisible (boolean b)

JLabel ext JComponent

JPanel ext JComponent

- add (JComponent c)
- setLayout(LayoutManager m)

interface LayoutManager

I dag har du lært

- Mer om å programmere med GUI
- Om utlegg på tegneflaten (Layout managers)
- Om størrelser, fonter og farger
- Hvilke GUI-verktøy du trenger i IN1010

- Om programmeringsmønsteret Model – View – Controller
- Litt om kobling mellom objekter og samhold innad i objekter
  - Coupling and Cohesion
- Og du har sett enda et eksempel på at interface er en god måte å spesifisere et grensesnitt som kan gi opphav til flere implementasjoner.