Grafiske brukergrensesnitt i Java - del 2



Eyvind W. Axelsen

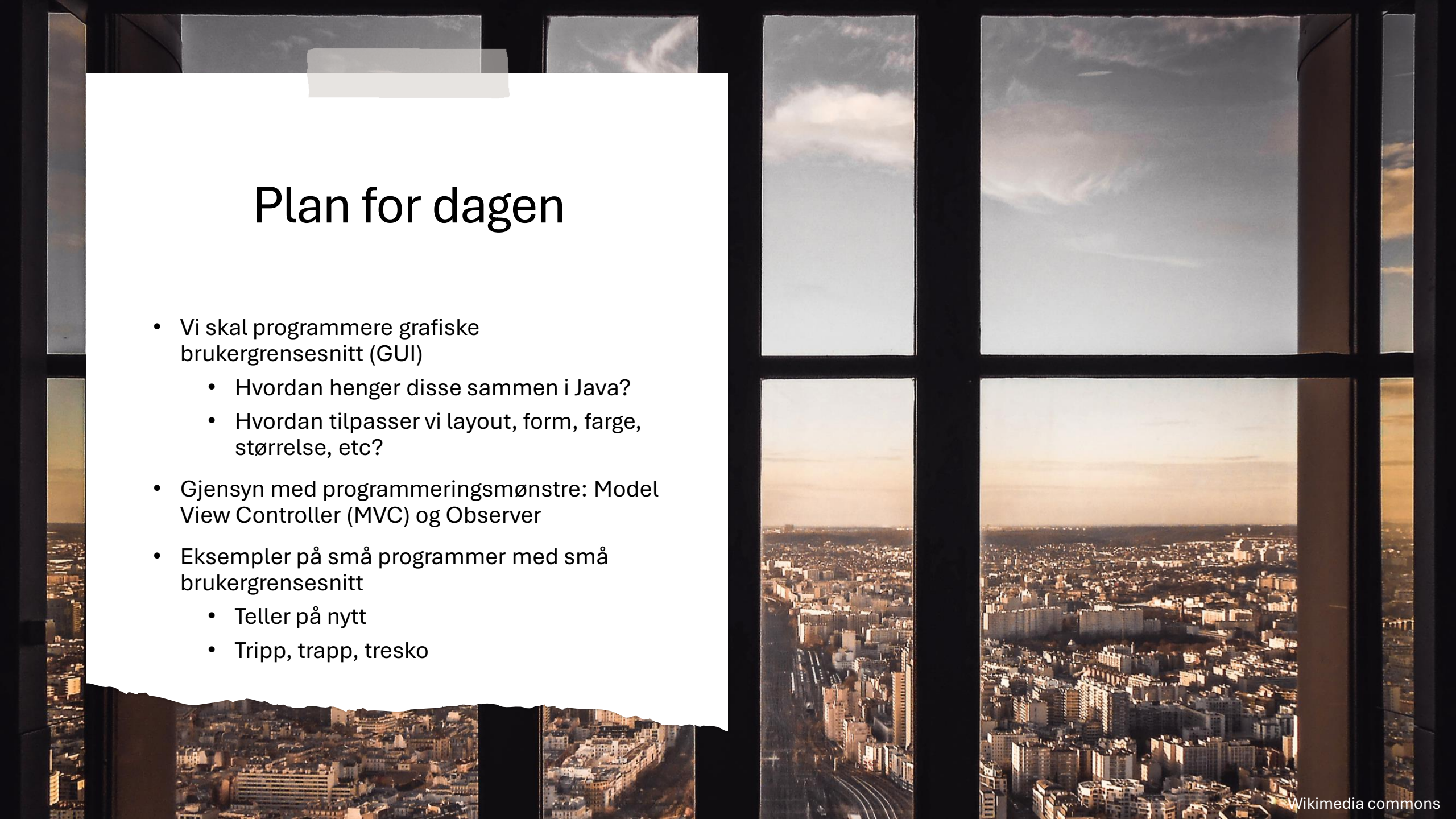
IN1010 Vår 2025

Med biter fra Dag Langmyhr og Stein Gjessings tidligere forelesninger

Illustrasjon fra https://ioflood.com

Plan for dagen

- Vi skal programmere grafiske brukergrensesnitt (GUI)
  - Hvordan henger disse sammen i Java?
  - Hvordan tilpasser vi layout, form, farge, størrelse, etc?
- Gjensyn med programmeringsmønstre: Model View Controller (MVC) og Observer
- Eksempler på små programmer med små brukergrensesnitt
  - Teller på nytt
  - Tripp, trapp, tresko

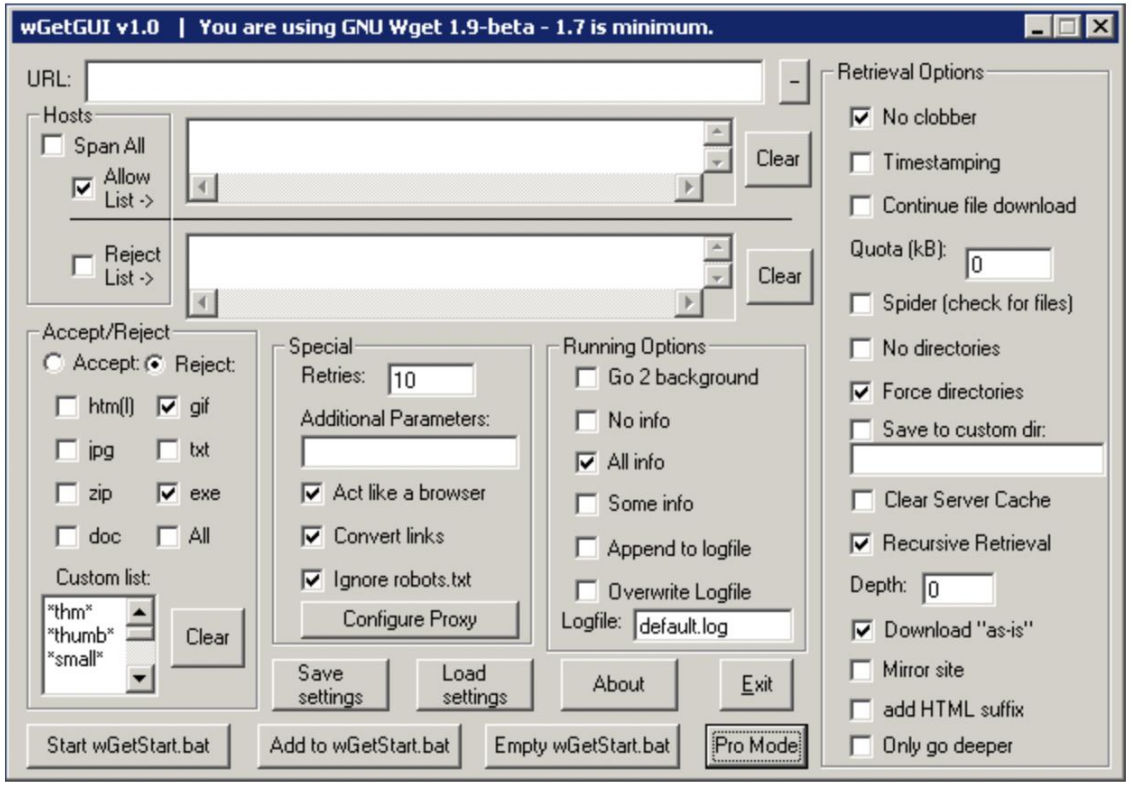


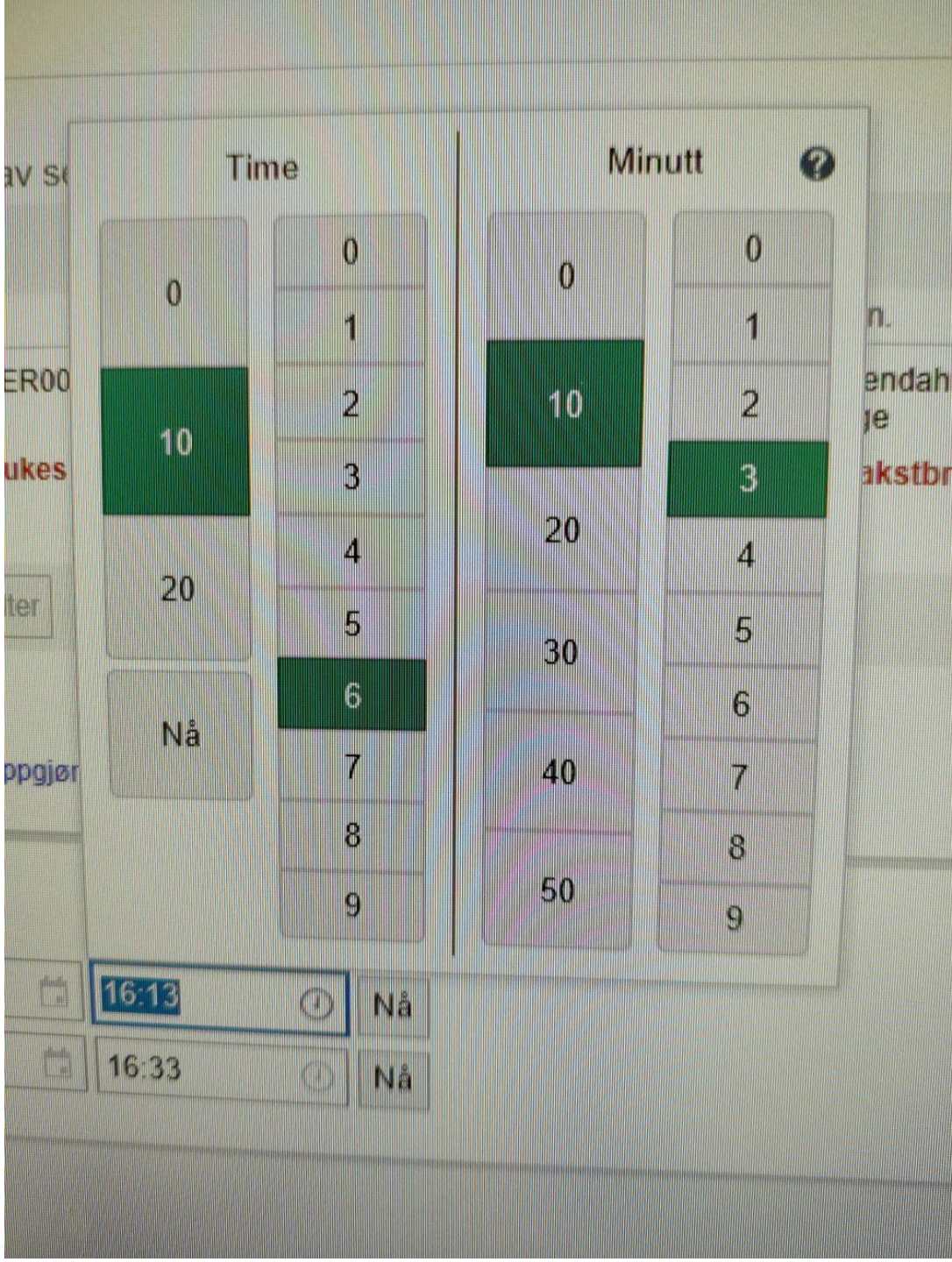
Wikimedia commons

27 Nov 2006

This Is What Happens When You Let Developers Create UI

Deep down inside every software developer, **there's a budding graphic designer waiting to get out.** And if you let that happen, you're in trouble. Or at least your users will be, anyway:





Det å lage brukergrensesnitt er ikke så lett

- Men (heldigvis!): vårt fokus her i IN1010 er ikke først å fremst å gjøre det *pent*, men å gjøre det *brukbart* og *forståelig*.

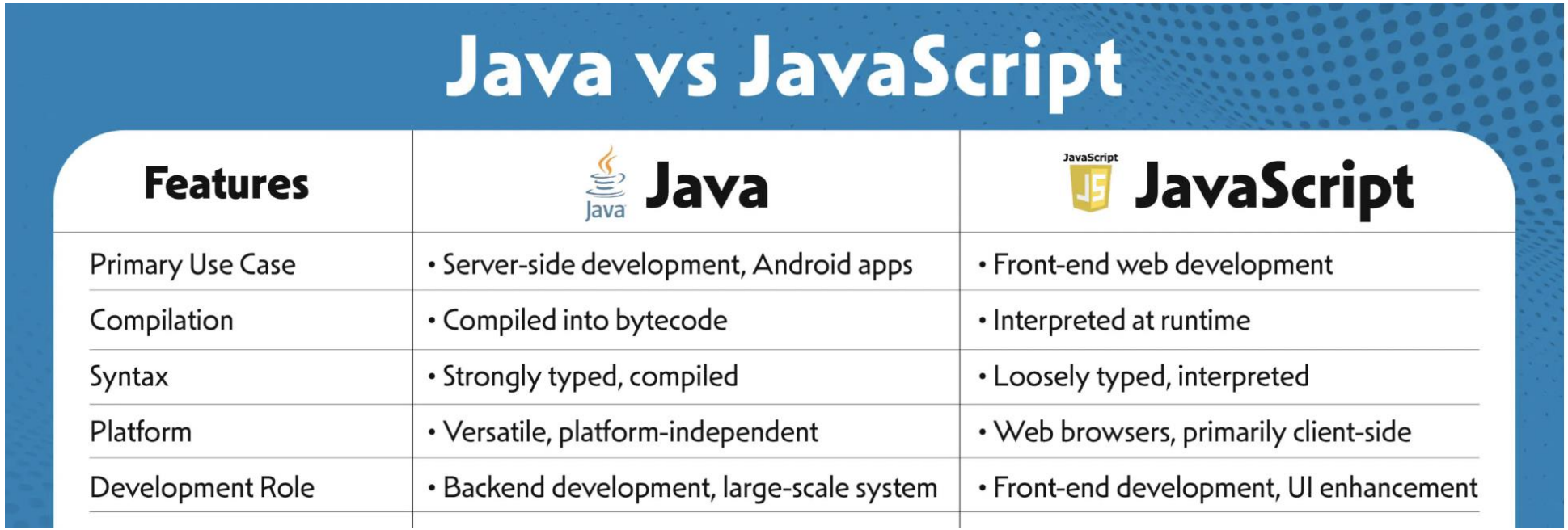
- Det er smart å tenke gjennom før man begynner å programmere:
  - Hvilke funksjoner skal programmet mitt ha?
  - Hvordan ønsker jeg å strukturere grensesnittet?



Brukergrensesnitt her og der

- Brukergrensesnitt i Java brukes først og fremst for *desktop-applikasjoner*

- Mange løsninger i dag er web-baserte. Da kan man bruke *Java* som «backend», og *JavaScript* som «frontend»



https://www.monarch-innovation.com/java-vs-javascript

Vi skal begynne med å lage et enkelt brukergrensesnitt for en *teller*

- Som vi har sett i starten av kurset
- Må kunne minst:
  - Øke med en og en
  - Vise frem gjeldende verdi
  - Nullstille

- Litt «spurv med kanoner»



Vi trenger en teller-klasse

- Holde styr på nåværende verdi
- Øke med én
- Nullstille

- Enkelt 😁 👍

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

    int hentVerdi() {
        return tellerverdi;
    }

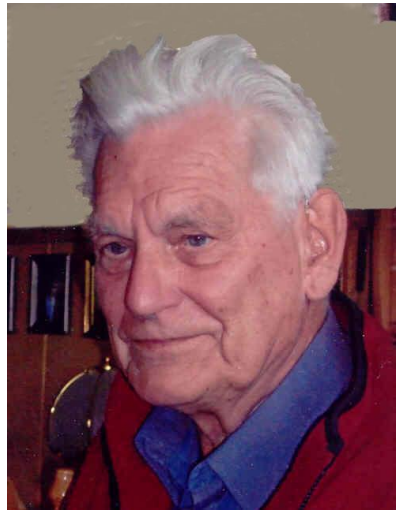
    void oekTeller() {
        ++tellerverdi;
    }

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

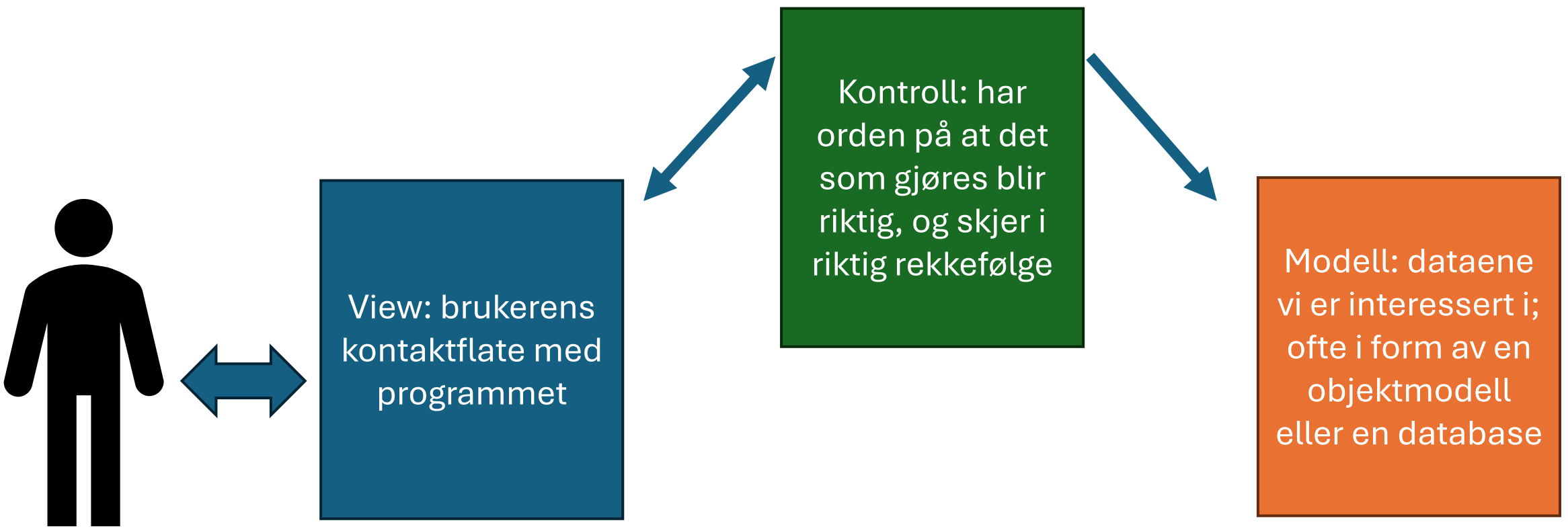
Gjensyn: Model, View, Controller (MVC)

- Model: dataene vi er interessert i
- View: brukergrensesnittet, håndtere interaksjon med brukeren
- Controller: sørge for at ting henger sammen, skjer i riktig rekkefølge, etc





MVC:
Trygve Reenskaug,
ca 1979


View: brukerens kontaktflate med programmet
Kontroll: har orden på at det som gjøres blir riktig, og skjer i riktig rekkefølge
Modell: dataene vi er interessert i; ofte i form av en objektmodell eller en database

I vår sammenheng

- Modellen: Teller-klassen
  - Vet ikke om noe annet enn seg selv
- View: brukergrensesnittet vi skal lage
  - Kjenner til kontrollen
  - Må la brukeren utføre nødvendige operasjoner
- Kontroll: når brukeren gjør noe i brukergrensesnittet, så sørger denne for at modellen oppdateres, og vice versa
  - Kjenner til view og modell

Kontrolleren

- Kjenner til brukergrensesnitt og modell
- Kan øke og nullstille telleren
- (Konstruktør: kommer tilbake til denne)

```
class Kontroll {
    private Brukergrensesnitt gui;
    private Teller modell;

    public void init() {
        gui.init(this);
    }

    void oekTeller() {
        modell.oekTeller();
        int x = modell.hentVerdi();
        gui.visFramVerdi(x);
    }

    void nullstillTeller() {
        modell.nullstillTeller();
        int x = modell.hentVerdi();
        gui.visFramVerdi(x);
    }
}
```

Brukergrensesnittet (view)

- Kontrolleren trenger å gjøre enkelte ting med det
- Vi kan lage oss et interface ut fra dette:

```
interface Brukergrensesnitt {
    void init(Kontroll k);
    void visFramVerdi(int t);
}
```

- Dette muliggjør *forskjellige* brukergrensesnitt

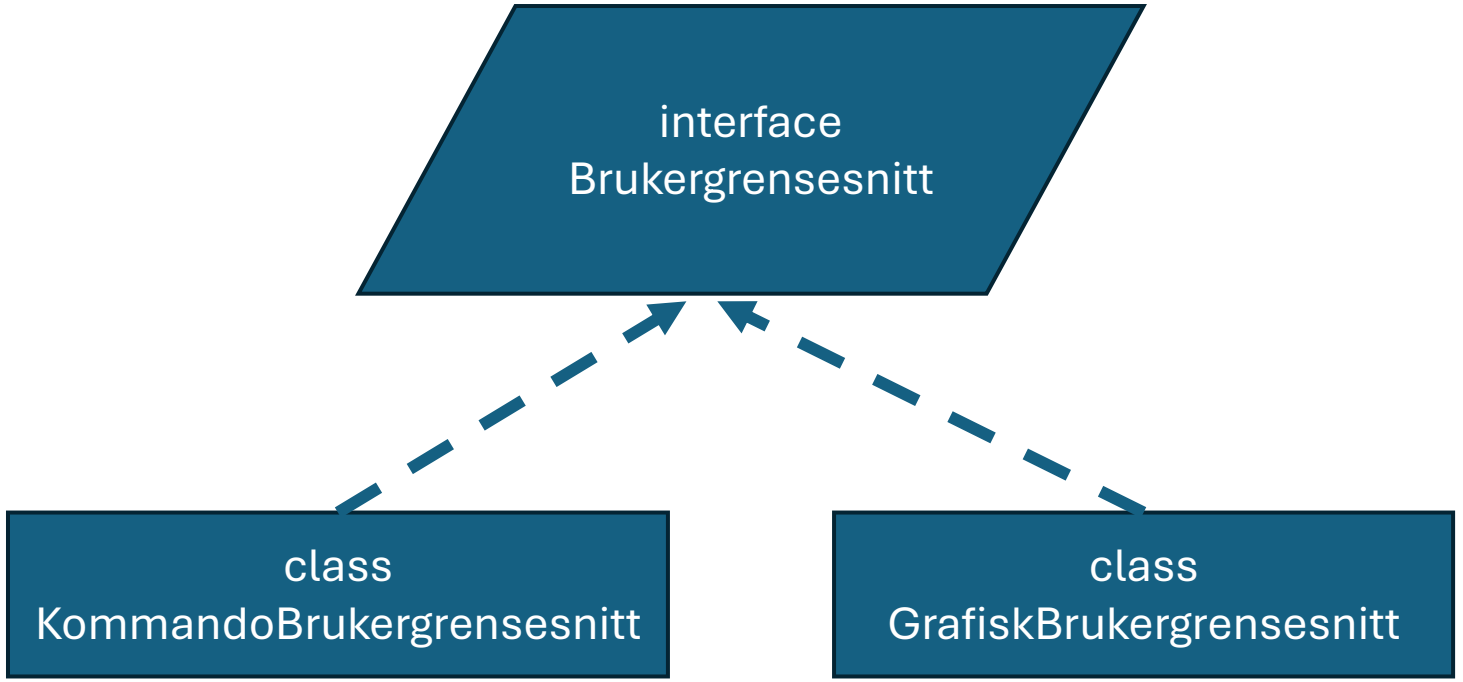
```
class Kontroll {
    private Brukergrensesnitt gui;
    private Teller modell;

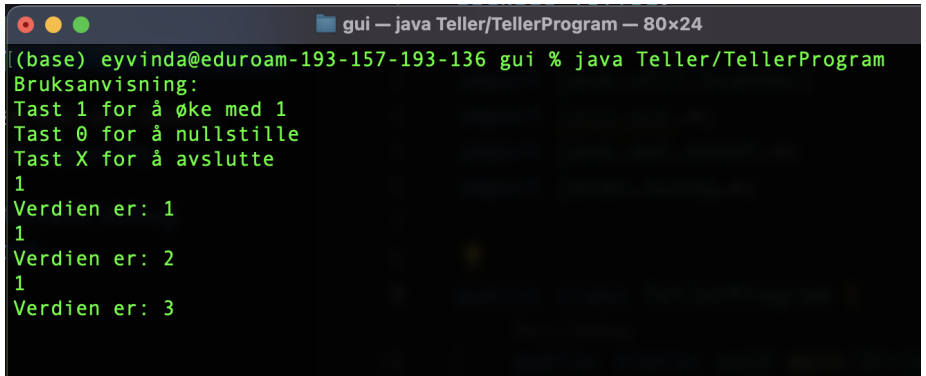
    public void init() {
        gui.init(this);
    }

    void oekTeller() {
        modell.oekTeller();
        int x = modell.hentVerdi();
        gui.visFramVerdi(x);
    }

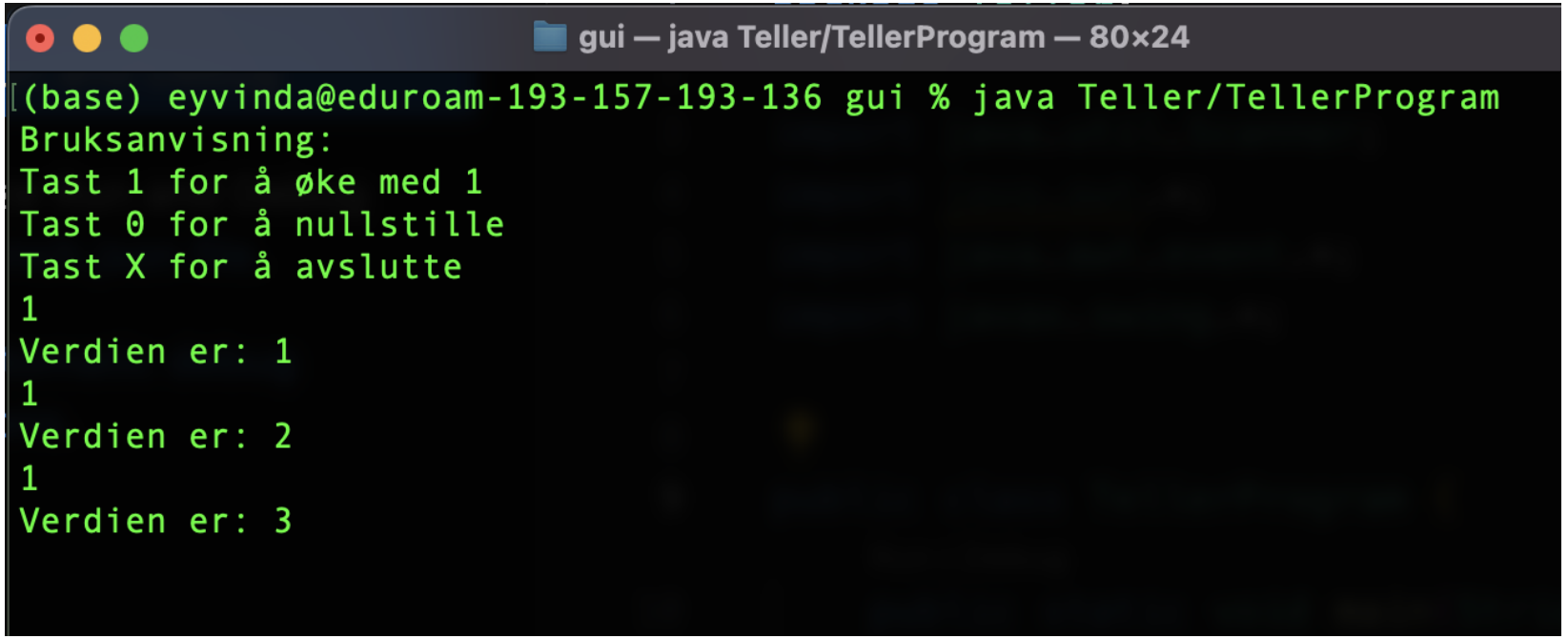
    void nullstillTeller() {
        modell.nullstillTeller();
        int x = modell.hentVerdi();
        gui.visFramVerdi(x);
    }
}
```

To alternative brukergrensesnitt








gui — java Teller/TellerProgram — 80×24
(base) eyvinda@eduroam-193-157-193-136 gui % java Teller/TellerProgram
Bruksanvisning:
Tast 1 for å øke med 1
Tast 0 for å nullstille
Tast X for å avslutte
1
Verdien er: 1
1
Verdien er: 2
1
Verdien er: 3


Teller
Verdi: 4
+1
Nullstill

```java
interface Brukergrensesnitt {
    void init(Kontroll k);
    void visFramVerdi(int t);
}
```

```
gui — java Teller/TellerProgram — 80×24
(base) eyvinda@eduroam-193-157-193-136 gui % java Teller/TellerProgram
Bruksanvisning:
Tast 1 for å øke med 1
Tast 0 for å nullstille
Tast X for å avslutte
1
Verdien er: 1
1
Verdien er: 2
1
Verdien er: 3
```

```java
class KommandoBrukergrensesnitt implements Brukergrensesnitt {
    Kontroll kontroll;

    @Override
    public void init(Kontroll k) {
        kontroll = k;
        boolean fortsett = true;
        Scanner innData = new Scanner(System.in);

        System.out.println(x:"Bruksanvisning: \nTast 1 for å øke med 1 \nTast 0

        while(fortsett) {
            String kommando = innData.nextLine();
            if(kommando.charAt(index:0) == '1')
                kontroll.oekTeller();
            else if(kommando.charAt(index:0) == '0')
                kontroll.nullstillTeller();
            else if(kommando.toUpperCase().charAt(index:0) == 'X')
                fortsett = false;
        }
        innData.close();
    }

    @Override
    public void visFramVerdi(int t) {
        System.out.println("Verdien er: " + t);
    }
}
```

La oss nå lage denne:



- Litt deklarasjoner først

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

class GrafiskBrukergrensesnitt implements Brukergrensesnitt {
    private Kontroll kontroll;
    private JButton oekButton, resetButton;
    private JFrame vindu;
    private JLabel verdiLabel;
    private JLabel verdiLedetekstLabel;
    private JPanel panel;
```

La oss nå lage denne:

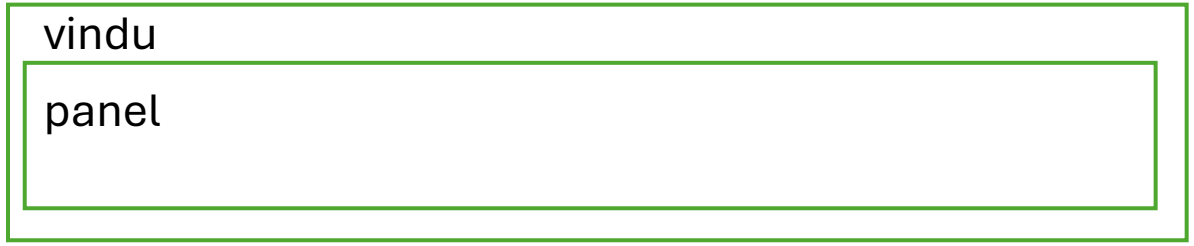


- Vi lager vinduet:

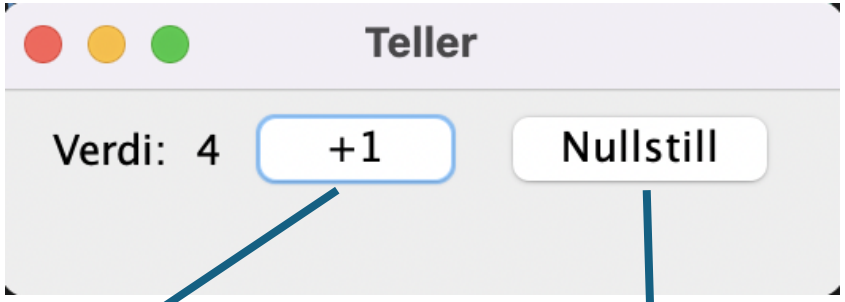
```
vindu = new JFrame(title:"Teller");
```

- Og så en tegneflate inne i vinduet som vi kan plassere andre elementer på:

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



La oss nå lage denne:



- Håndtere trykk på knapper:

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

- Hendelseshåndtering: i et brukergrensesnitt må man kunne reagere på det brukeren gjør!
- I Java håndteres dette med egne klasser/objekter som har metoder som tar seg av det som skal skje som følge av hendelsen

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

La oss nå lage denne:



- Lage knapper og koble til hendelseshåndtering:

```
oekButton = new JButton(text:" +1 ");
oekButton.addActionListener(new OekTeller());

resetButton = new JButton(text:"Nullstill");
resetButton.addActionListener(new NullstillTeller());
```

- Legge til i panelet:

```
panel.add(oekButton);
panel.add(resetButton);
```

La oss nå lage denne:



- Hele init-metoden

```java
@Override
public void init(Kontroll k) {
    kontroll = k;
    vindu = new JFrame(title:"Teller");
    panel = new JPanel();
    vindu.add(panel);
    vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    oekButton = new JButton(text:" +1 ");
    oekButton.addActionListener(new OekTeller());

    resetButton = new JButton(text:"Nullstill");
    resetButton.addActionListener(new NullstillTeller());

    verdiLedetekstLabel = new JLabel(text:"Verdi: ");
    verdiLabel = new JLabel(text:"   ");

    panel.add(verdiLedetekstLabel);
    panel.add(verdiLabel);
    panel.add(oekButton);
    panel.add(resetButton);

    vindu.pack();
    vindu.setLocationRelativeTo(c:null);
    vindu.setVisible(b:true);
}
```

Legg til alle elementer i panelet

Pakk sammen og gjør synlig, midt på skjermen

Lukk vindu når man trykker på X

Observer-pattern i praksis




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

```
oekButton = new JButton(text:" +1 ");
oekButton.addActionListener(new OekTeller());
```

Subjekt

Observatør

Hovedprogrammet lar brukeren velge type grensesnitt

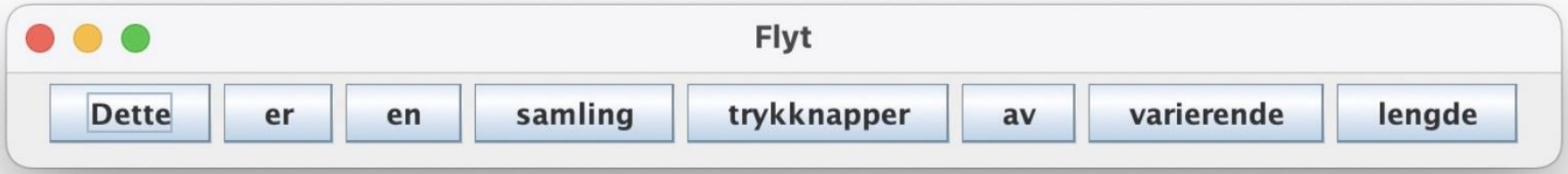
```java
public class TellerProgram {
    Run | Debug
    public static void main(String[] args){
        if(args.length == 0)
        {
            System.out.println(x:"Programmet må startes med enten G eller K som parameter");
        }
        else
        {
            String type = args[0].substring(beginIndex:0,endIndex:1).toUpperCase();
            Kontroll k = new Kontroll(type);
            k.init();
        }
    }
}
```

```
(base) eyvinda@eduroam-193-157-193-136 gui % java Teller/TellerProgram
Programmet må startes med enten G eller K som parameter
(base) eyvinda@eduroam-193-157-193-136 gui % java Teller/TellerProgram G
```



Form og farge

- For alle paneler kan vi angi en layout-type som vi ønsker oss
- Standard er FlowLayout, da kommer elementene etter hverandre etter hvert som de legges til, og «flyter» med størrelsen på vinduet

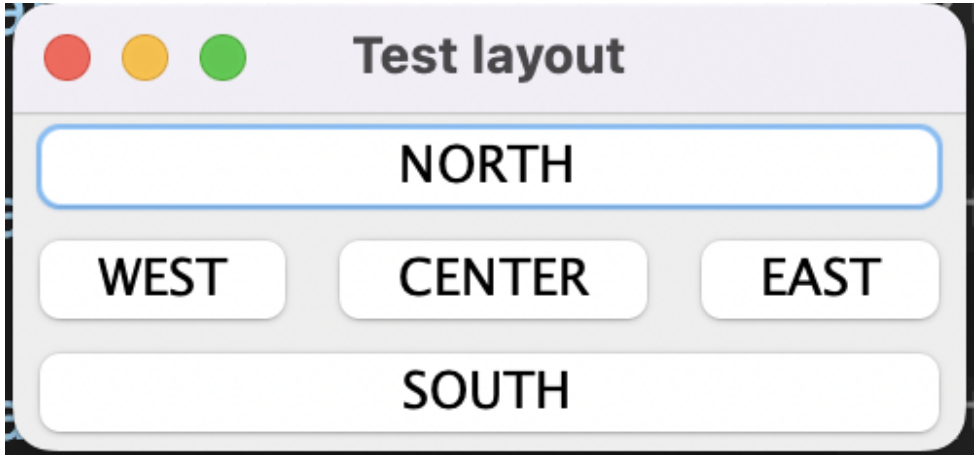






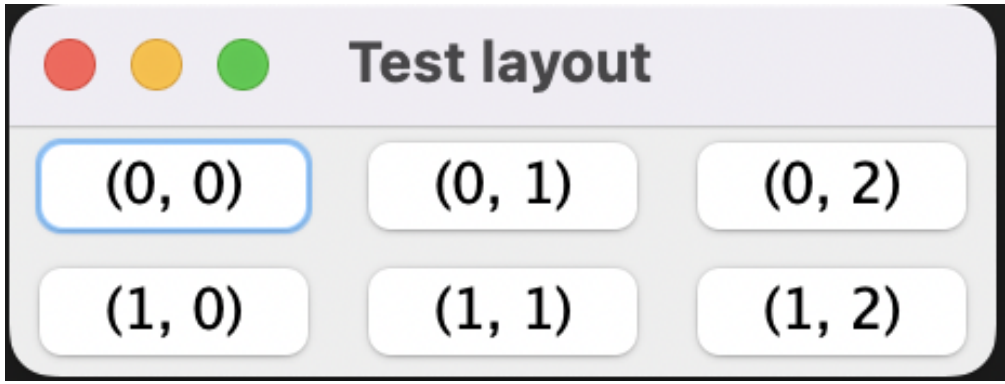
BorderLayout: når vi ønsker faste posisjoner

```
JFrame vindu = new JFrame(title:"Test layout");
JPanel panel = new JPanel();
vindu.add(panel);
panel.setLayout(new BorderLayout());
panel.add(new JButton(text:"NORTH"),
    BorderLayout.NORTH);
panel.add(new JButton(text:"SOUTH"),
    BorderLayout.SOUTH);
panel.add(new JButton(text:"EAST"),
    BorderLayout.EAST);
panel.add(new JButton(text:"WEST"),
    BorderLayout.WEST);
panel.add(new JButton(text:"CENTER"),
    BorderLayout.CENTER);
```



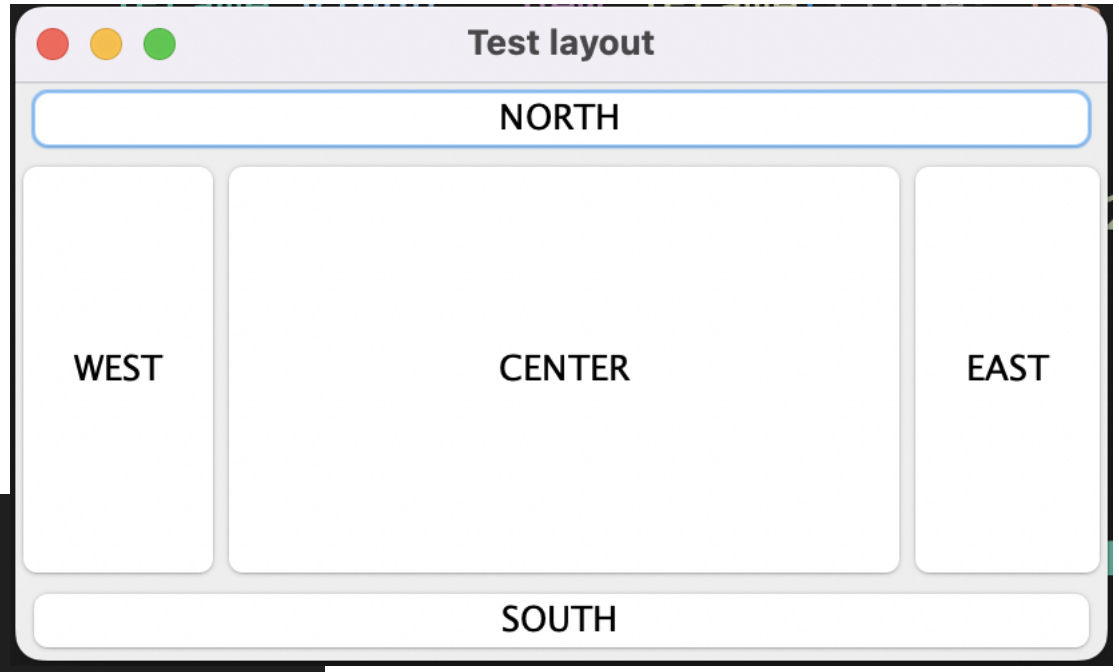
GridLayout: når vi ønsker et rutenett

```
JFrame vindu = new JFrame(title:"Test layout");
JPanel panel = new JPanel();
vindu.add(panel);
panel.setLayout(new GridLayout(rows:2,cols:3));
panel.add(new JButton(text:"(0, 0)"));
panel.add(new JButton(text:"(0, 1)"));
panel.add(new JButton(text:"(0, 2)"));
panel.add(new JButton(text:"(1, 0)"));
panel.add(new JButton(text:"(1, 1)"));
panel.add(new JButton(text:"(1, 2)"));
```



- I rutenettet angir vi posisjon som (rad, kolonne) – rad er vannrett, kolonne er loddrett

Justere størrelse



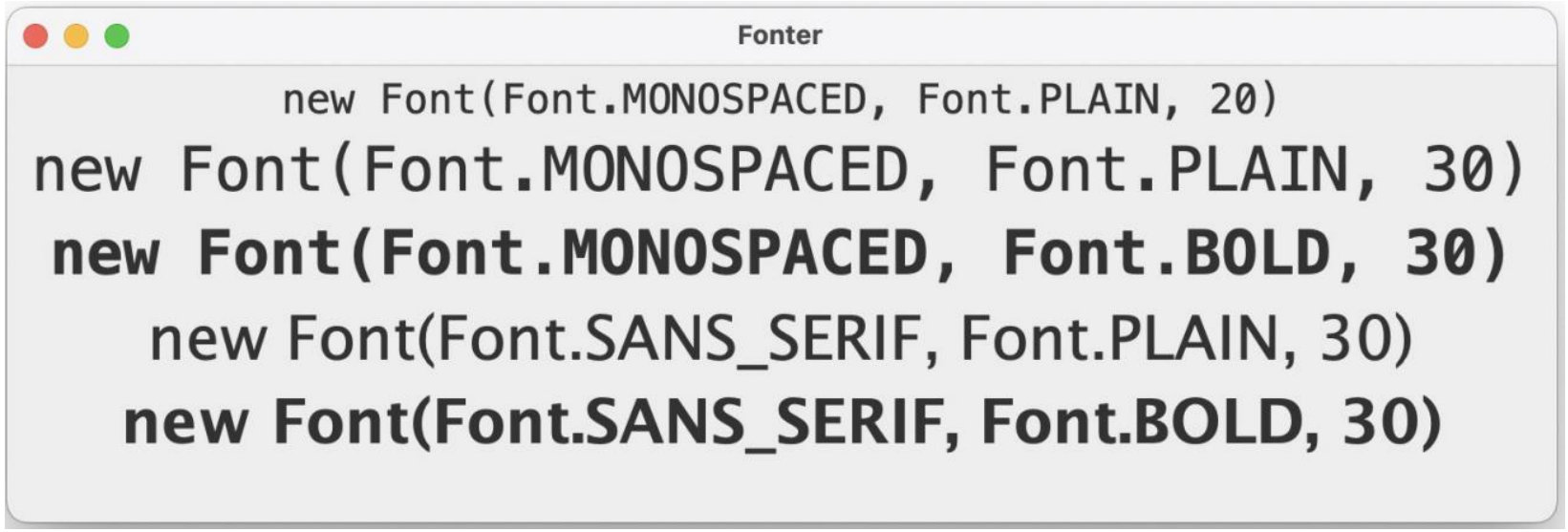

```java
JFrame vindu = new JFrame(title:"Test layout");
JPanel panel = new JPanel();
vindu.add(panel);
panel.setLayout(new BorderLayout());
panel.add(new JButton(text:"NORTH"), BorderLayout.NORTH);
panel.add(new JButton(text:"SOUTH"), BorderLayout.SOUTH);
panel.add(new JButton(text:"EAST") , BorderLayout.EAST);
panel.add(new JButton(text:"WEST"), BorderLayout.WEST);

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

Vi gir et hint om at vi ønsker en annen størrelse på sentrum-knappen. Swing kan velge å ignorere dette om nødvendig.

Fonter

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



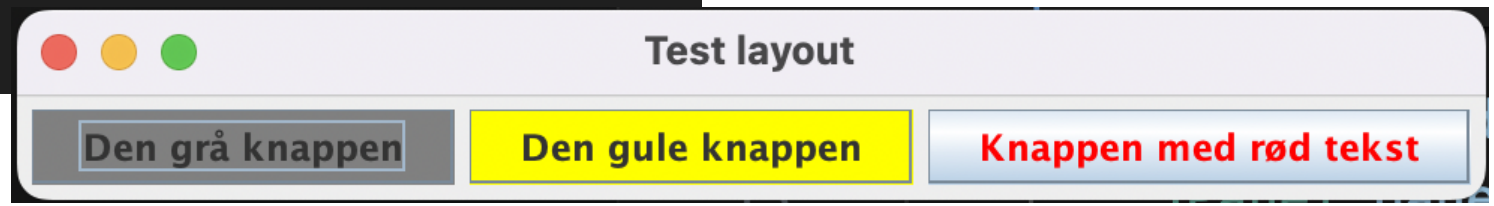
- Enhet for fontstørrelse er skjermpunkter/pixels

Farger på bakgrunn eller tekst

```
JFrame vindu = new JFrame(title:"Test layout");
JPanel panel = new JPanel();
vindu.add(panel);

var graa = new JButton(text:"Den grå knappen");
graa.setBackground(Color.GRAY);
panel.add(graa);
var gul = new JButton(text:"Den gule knappen");
gul.setBackground(Color.YELLOW);
panel.add(gul);

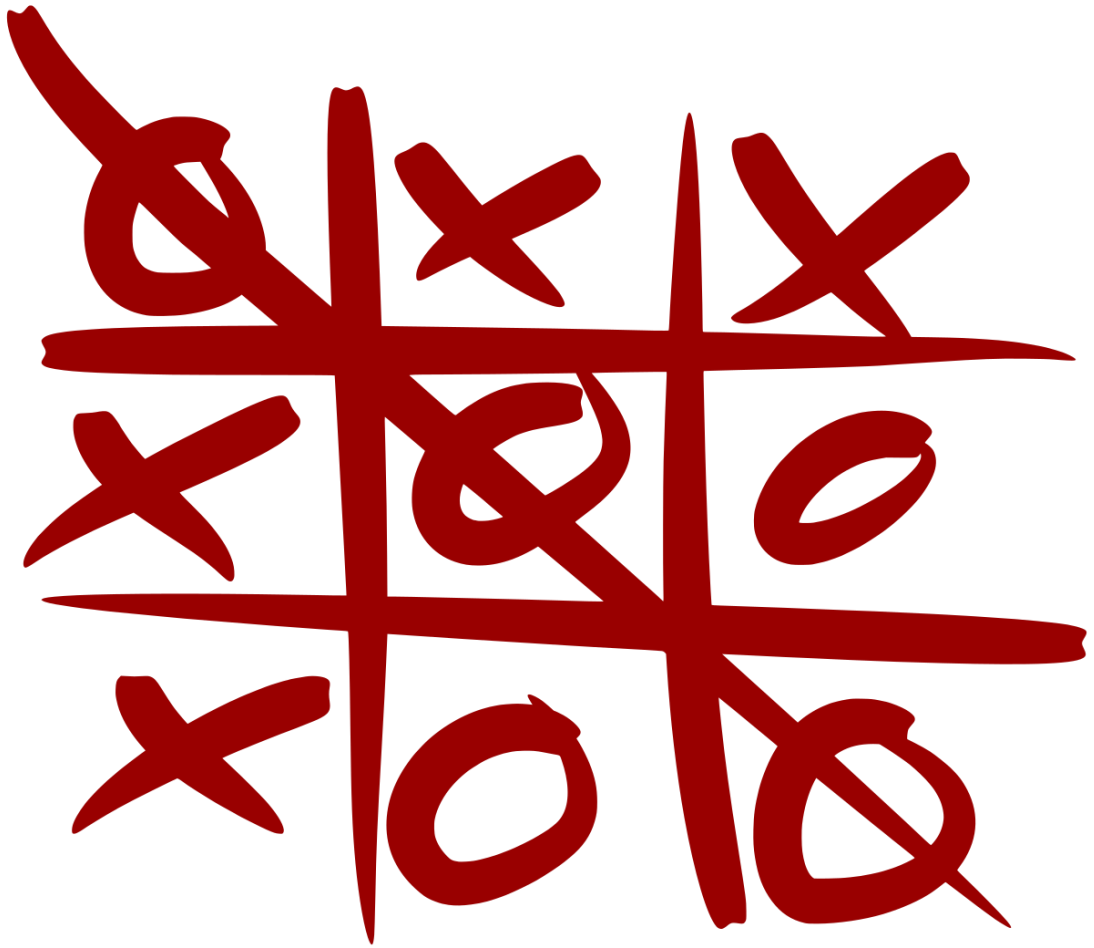
var roed = new JButton(text:"Knappen med rød tekst");
roed.setForeground(Color.RED);
panel.add(roed);
```



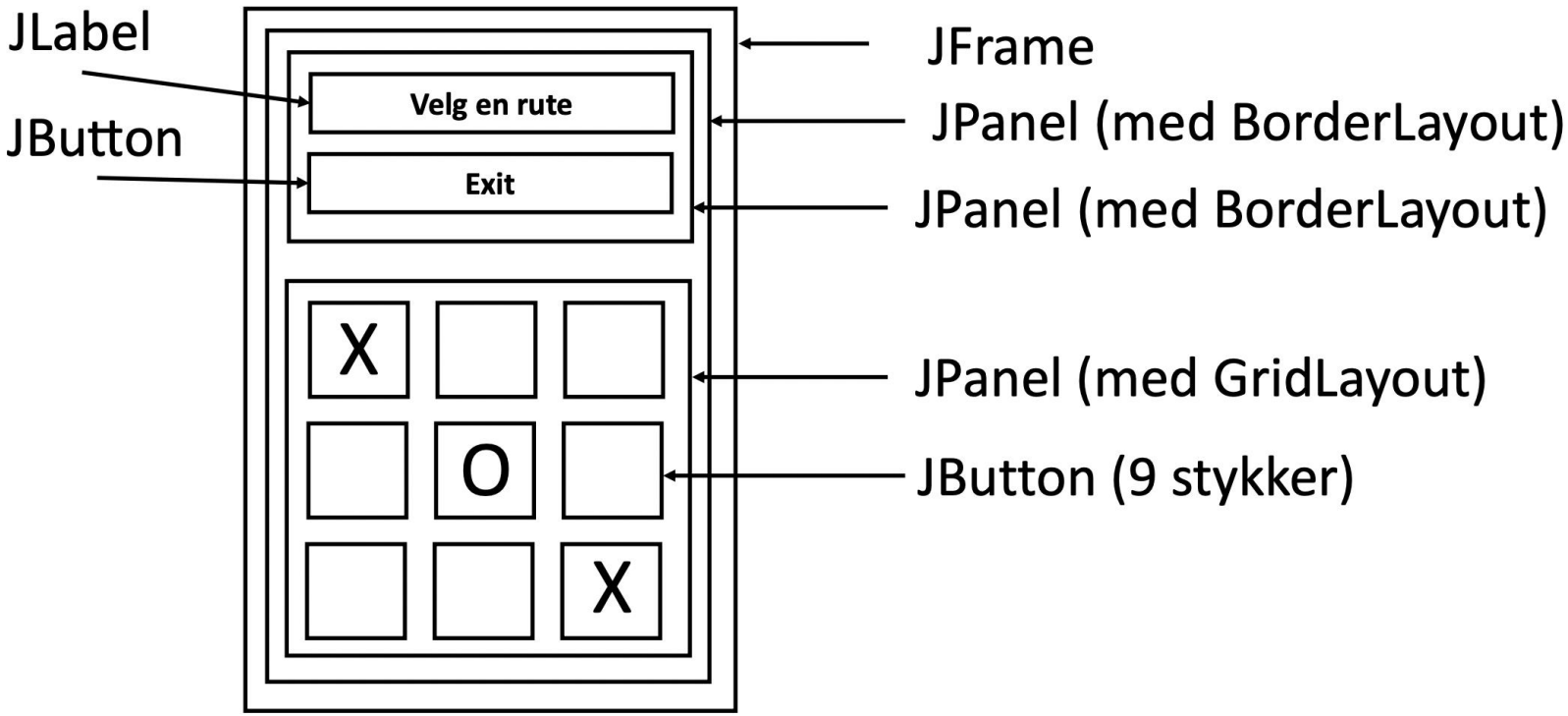
Nytt eksempel – tripp trapp tresko (TTT)

Ting å tenke på når man skal lage et nytt GUI-program:

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



Design-skisse for programmet vi vil lage

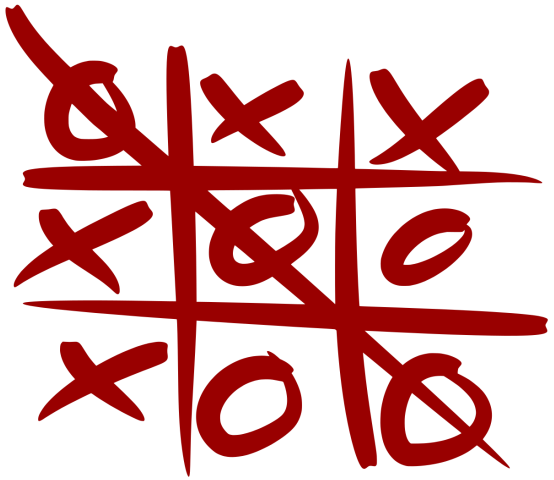


MVC-struktur

- Modellen er brettet vårt med 3x3 ruter
  - Hvilke operasjoner trenger vi å kunne utføre på modellen?
  - Hva slags datastruktur skal vi bruke?
    - Vi bruker char[][] brett = new char[3][3]; her, men kunne gått for noe annet også
  - Hvilke regler har spillet?
    - Avveining: hvor mye logikk skal vi plassere i modellen, og hvor mye/hvilken skal vi plassere i kontrolleren? De lærde strides…
- View-et er skjermbildet skissert på forrige slide
- Kontrolleren:
  - Hvem sin tur er det?
  - Hvordan håndtere lovlige og ulovlige trekk?
  - Kommunisere når noen har vunnet
  - …

Modellen

```
class Modell {
    // Brettet innholder:
    // - Blank der det er ledig (initialverdier)
    // - X der maskinen har valgt
    // - O der brukeren har valgt
    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] = ' ';
    }
```



Modellen har også logikk for spillets tilstand

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

Et trekk er lovlig om det ikke allerede er satt et merke der

Metoden noterTrekk brukes for å sette et merke (O eller X) på en gitt posisjon

Totalt har brettet 3x3=9 ruter

Modellen har også logikk for spillets tilstand

```
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;
}
```

En av spillerne har vunnet dersom O eller X har tre markeringer på en rad, i en kolonne eller på en diagonal

Kontrolleren

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

    Kontroll () {
        gui = new GUI(this);
        modell = new Modell();
    }
    void startSpillet () {
        laMaskinenTrekke();
    }
    void laMaskinenTrekke () {
        int r, k;
        do {
            r = trekk(a:0,b:2);  k = trekk(a:0,b:2);
        } while (! modell.lovligTrekk(r,k));
        modell.noterTrekk(r, k, spiller:'X');
        gui.markerTrekk(r, k, c:'X');
        if (modell.harVunnet(s:'X')) {
            modell.noterVinner(vinner:"Maskinen");
            gui.visStatus('X' + " vant");
        }
    }
    private int trekk (int a, int b) {
        // Trekk et tilfeldig heltall x slik at a <= x <= b.
        return (int)(Math.random()*(b-a+1)) + a;
    }
```

Kontrolleren kjenner til både GUI og modell

Maskinen er ikke en veldig smart motspiller, den bare trekker tilfeldige posisjoner helt til den finner et lovlig trekk

Math.random brukes til å trekke «tilfeldige» tall

```
void brukervalg (int r, int k) {
    if (modell.erSpilletFerdig()) {
        gui.visStatus(tekst:"Ingen flere trekk!");
        return;
    }
    if (! modell.lovligTrekk(r, k)) {
        gui.visStatus(tekst:"Ulovlig trekk!");
        return;
    }
    modell.noterTrekk(r, k, spiller:'O');
    gui.markerTrekk(r, k, c:'O');
    if (modell.harVunnet(s:'O')) {
        gui.visStatus('O' + " vant");
        System.out.println(x:"Jeg vant");
        modell.noterVinner(vinner:"Du");
        return;
    }
    laMaskinenTrekke();
    if (modell.erSpilletFerdig())
        return;
    if (modell.erBrettetFullt()) {
        modell.noterUavgjort();
        gui.visStatus(tekst:"Uavgjort");
    }
    else
        gui.visStatus(tekst:"Velg en rute");
}
```

Håndtering av brukerens trekk

Sjekk om trekket gjør at brukeren har vunnet; hvis ikke er det maskinens tur

Brukergrensesnittet

```java
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;

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

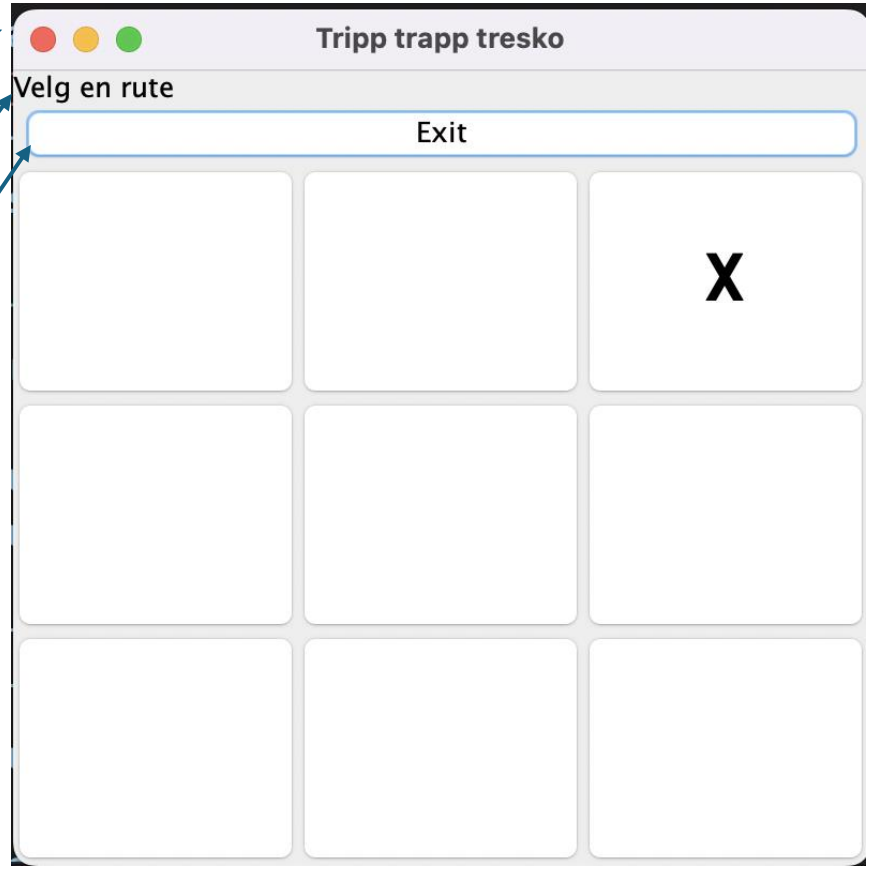
        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        vindu.add(panel);

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

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

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

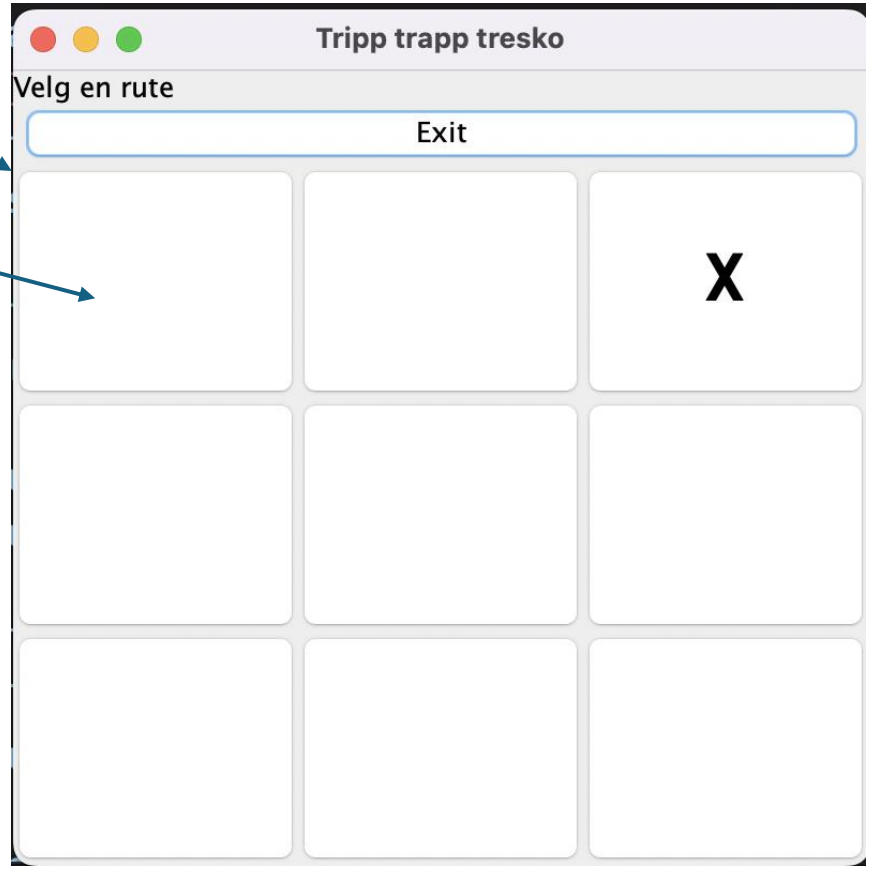
Variabler for de ulike elementene



Brukergrensesnittet

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

vindu.pack();
vindu.setSize(width:400,height:400);
vindu.setLocationRelativeTo(c:null);
vindu.setVisible(b:true);
```



Håndtere hendelser (brukervalg)

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

vindu.pack();
vindu.setSize(width:400,height:400);
vindu.setLocationRelativeTo(c:null);
vindu.setVisible(b:true);
```

```
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);
    }
}
```

Viktige klasser og metoder for GUI i IN1010

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

Oppsummering

- Nå kan vi programmere brukergrensesnitt i Java! 💪
  - Posisjonering i vinduet
  - Varianter av layout
  - Hendelseshåndtering
- Det er ofte lurt å tenke seg ut en skisse av hvordan programmet skal se ut før man begynner å programmere
- Det kan være fornuftig å bruke MVC-mønsteret
  - Model – modellen, dataene vi ønsker å representere
  - View – brukergrensesnittet, det som brukeren interagerer med
  - Controller – holder styr på forholdet mellom modell og brukergrensesnitt
- Neste uke: Invarianter med Lars

