GUI – Grafiske brukergrensesnitt (del 2)

IN1010, 28. april 2026

Magnus Skjegstad

Med materiale fra Eivind W. Axelsen, Dag Langmyhr og Stein Gjessings tidligere forelesninger

Plan for i dag

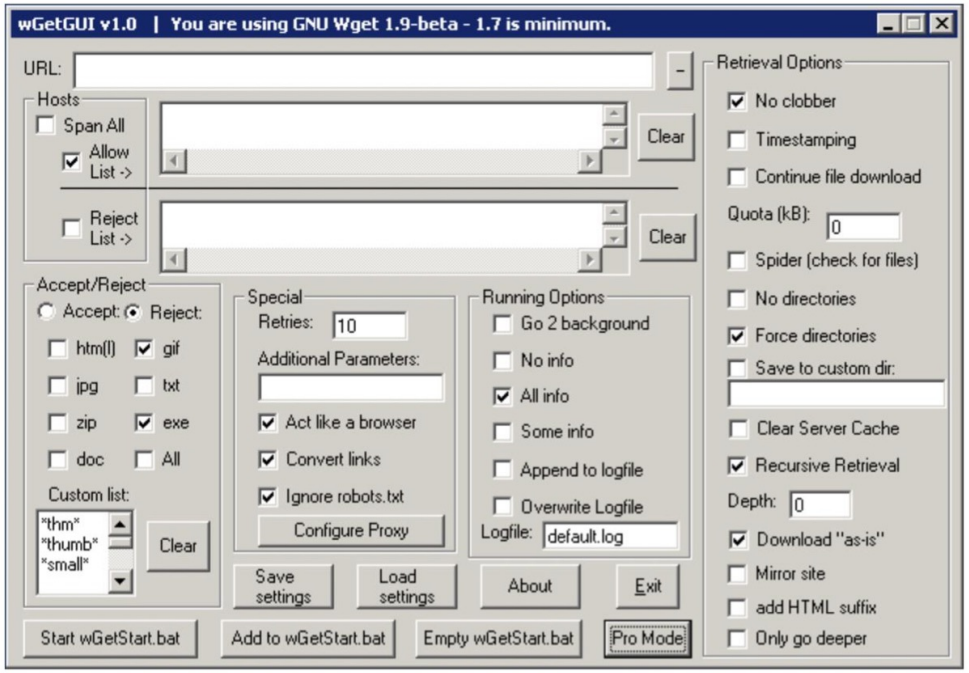
- Vi skal programmere grafiske brukergrensesnitt (GUI)
  - Hvordan henger disse sammen i Java?
  - Hvordan tilpasser vi farge, form, layout, størrelse osv?
- Gjensyn med programmeringsmønstre: Model View Controller (MVC) og Observer
- Eksempler på små programmer med små brukergrensesnitt
  - Teller
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

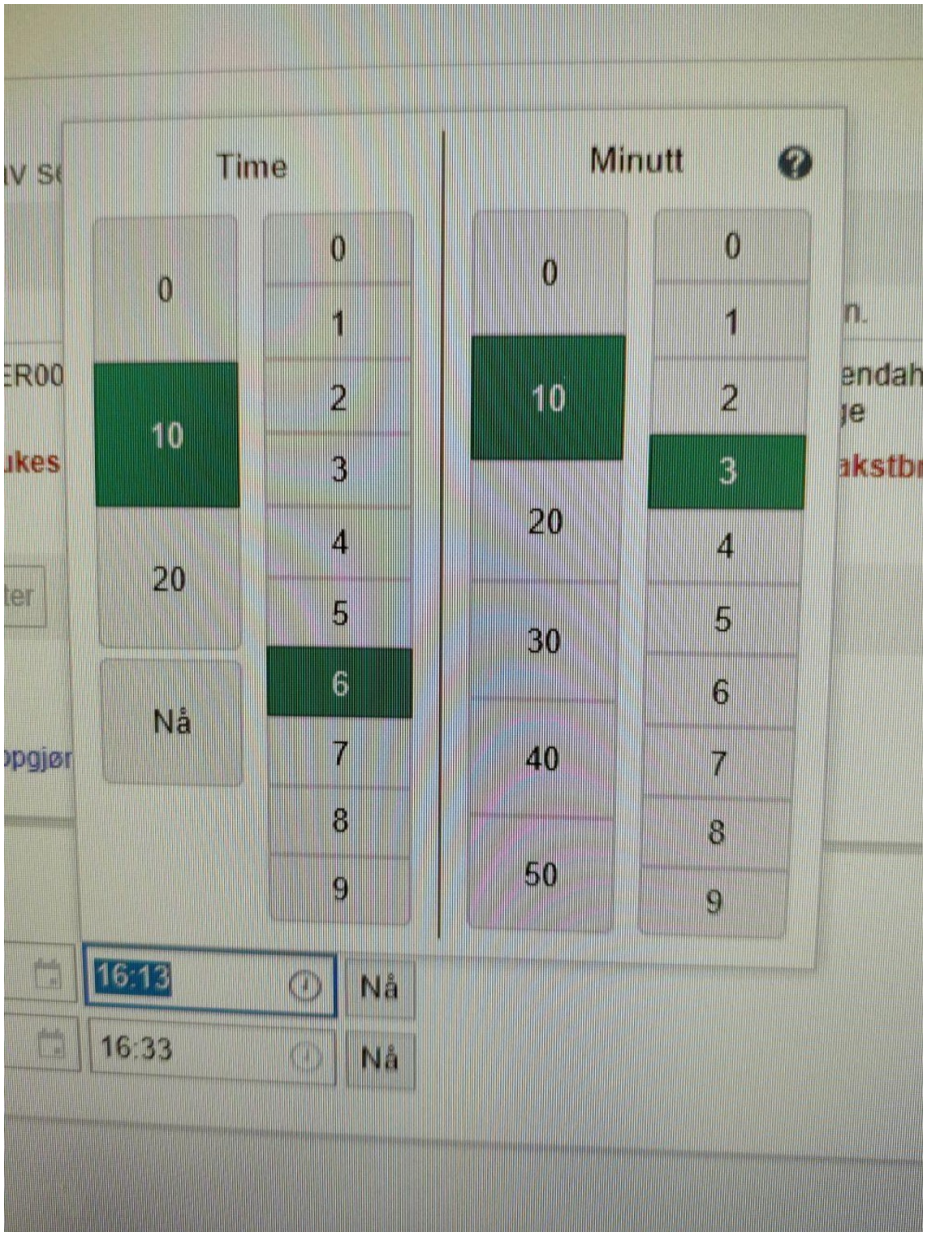
CODING HORROR
programming and human factors

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:





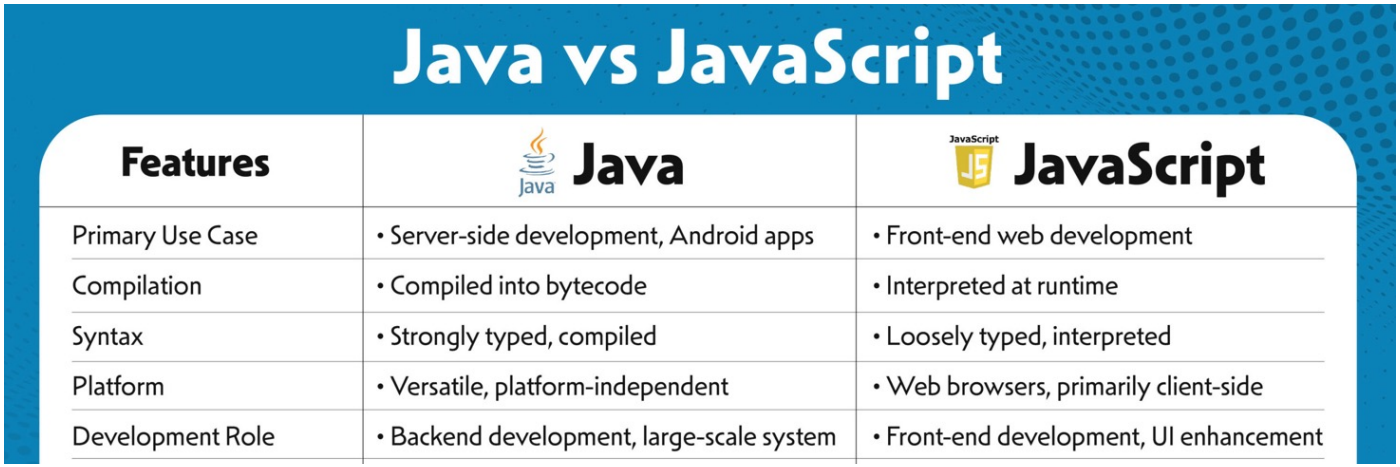
Å lage brukergrensesnitt er ikke så lett

- Men vårt fokus i IN1010 er ikke først og fremst at det blir 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 brukergrensesnittet?

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».



Java vs JavaScript

Features	Java	JavaScript
Primary Use Case	• Server-side development, Android apps	• Front-end web development
Compilation	• Compiled into bytecode	• Interpreted at runtime
Syntax	• Strongly typed, compiled	• Loosely typed, interpreted
Platform	• Versatile, platform-independent	• Web browsers, primarily client-side
Development Role	• Backend development, large-scale system	• Front-end development, UI enhancement

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

Vi skal begynne med et enkelt brukergrensesnitt for teller

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

- Litt «spurv med kanoner»



Vi trenger en teller-klasse

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

- Lett!

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

    int hentVerdi() {
        return tellerverdi;
    }

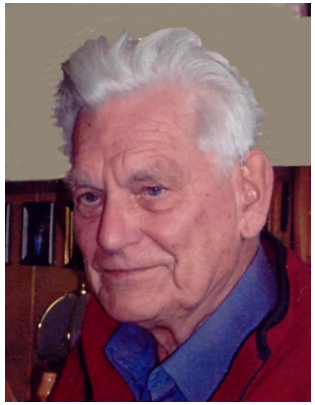
    void økTeller() {
        ++tellerverdi;
    }

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

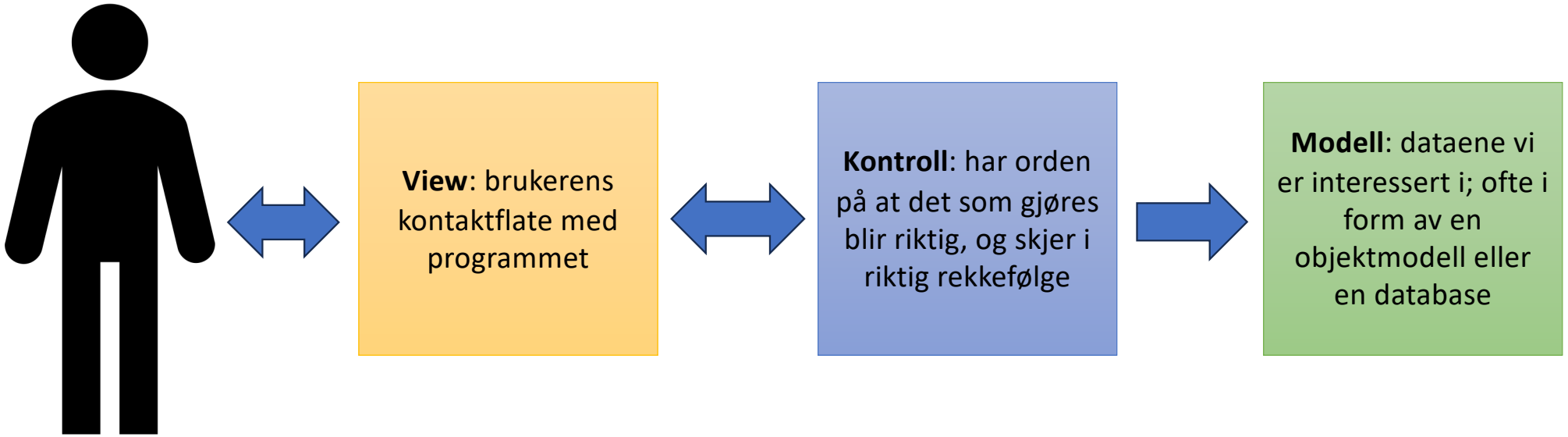
Gjensyn: Model, View, Controller (MVC)

- 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, foreslått 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

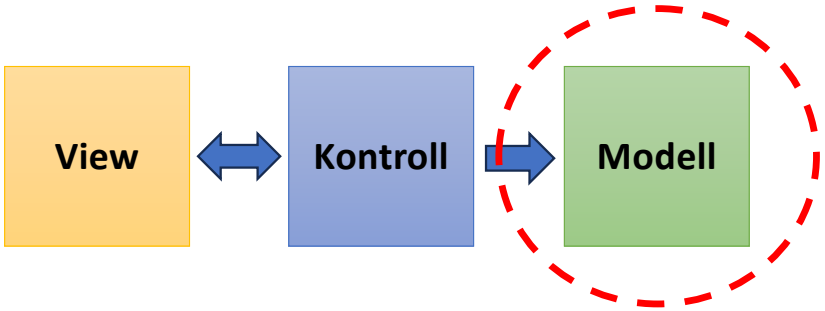
I vår sammenheng

- Modellen: Teller-klassen
  - Vet ikke om noe annet enn seg selv
- View: brukergrensesnittet vi skal lage
  - Kjenner til kontrolleren
  - Må la brukeren utføre nødvendig operasjoner
- Kontroll: når brukeren gjør noe i brukergrensesnittet, så sørger denne for at modellen oppdateres – og at view oppdateres.
  - Kjenner til view og model



Modell

- Teller-klassen fra tidligere




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

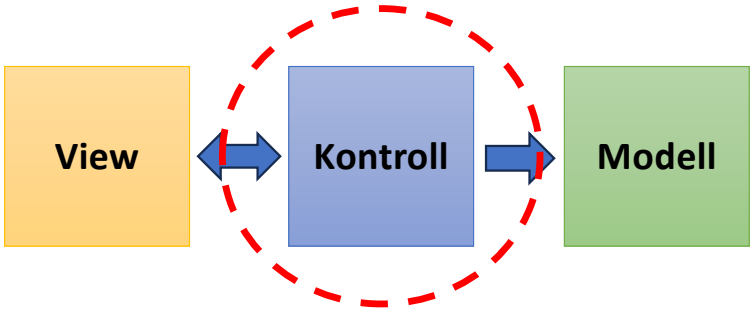
    int hentVerdi() {
        return tellerverdi;
    }

    void økTeller() {
        ++tellerverdi;
    }

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

Kontroller

- Kjenner til brukergrensesnitt og modell
- Kan øke og nullstille teller
- (Konstruktør: denne kommer senere)




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

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

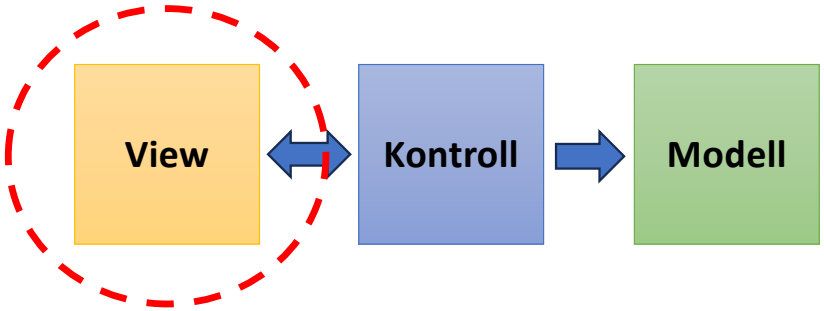
    void økTeller() {
        modell.økTeller();
        int x = modell.hentVerdi();
        gui.visFramVerdi(x);
    }

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

Brukergrensesnittet (View)

- Kontrolleren trenger å oppdatere noen verdier. Vi kan lage et interface for dette:

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



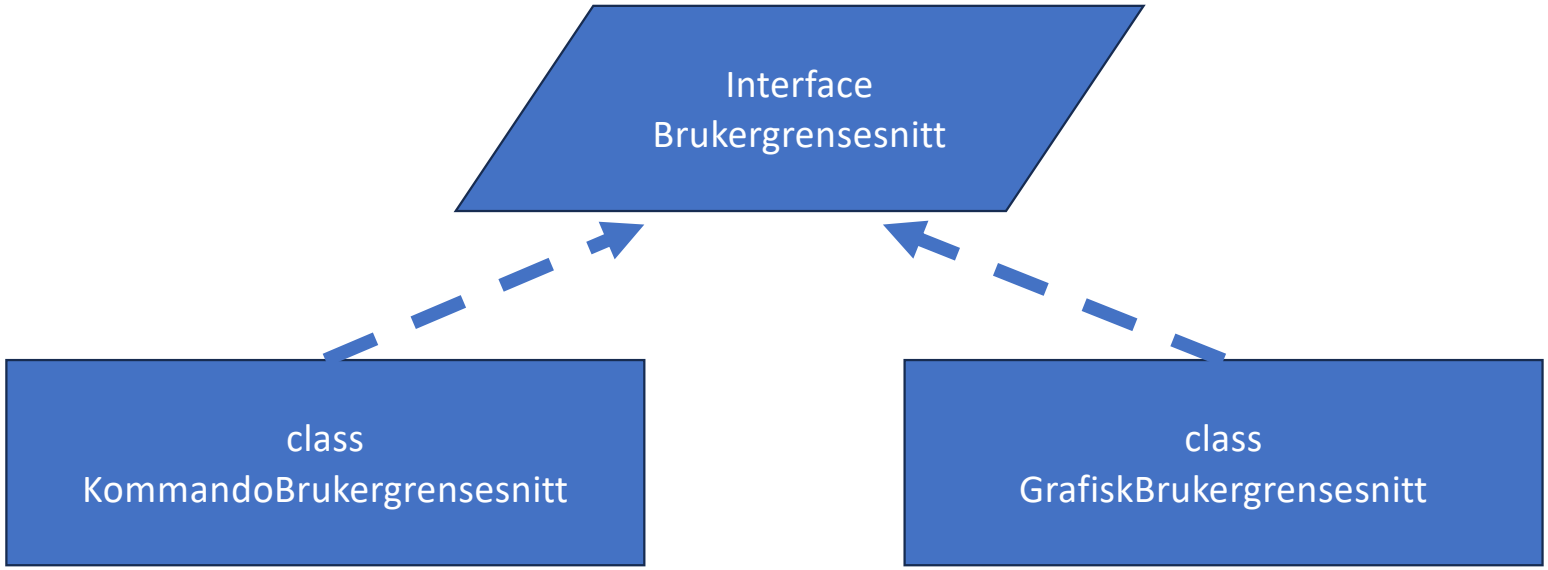

```
class Kontroll {
    private Brukergrensesnitt gui;
    private Teller modell;

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

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

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

Alternative brukergrensesnitt



```
moskjegs@eduroam-193-157-170-149 del1 % java TellerProgram K
Typen er: 'K'
Bruksanvisning:
Tast 1 for å øke med 1
Tast 0 for å nullstille
Tast X for å avslutte
1
Verdien er: 1
0
Verdien er: 0
X
```



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

```
moskjegs@eduroam-193-157-170-149 del1 % java TellerProgram K
Typen er: 'K'
Bruksanvisning:
Tast 1 for å øke med 1
Tast 0 for å nullstille
Tast X for å avslutte
1
Verdien er: 1
0
Verdien er: 0
X
```

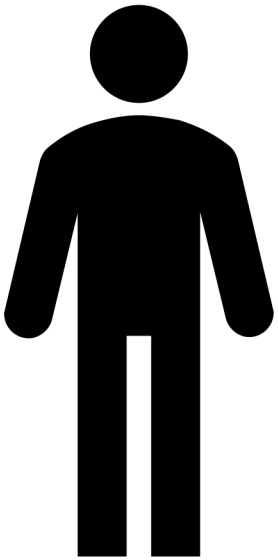
```java
class KommandoBrukergrensesnitt implements Brukergrensesnitt {
    Kontroll kontroll;

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

        System.out
                .println("Bruksanvisning: \nTast 1 for å øke m

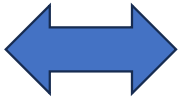
        while (fortsett) {
            String kommando = innData.nextLine();
            if (kommando.charAt(0) == '1')
                kontroll.oekTeller();
            else if (kommando.charAt(0) == '0')
                kontroll.nullstillTeller();
            else if (kommando.toUpperCase().charAt(0) == 'X')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

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

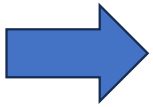




View: brukerens kontaktflate med programmet



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

Vi implementerer begge grensesnittene



Vi skal nå lage det grafiske brukergrensesnittet

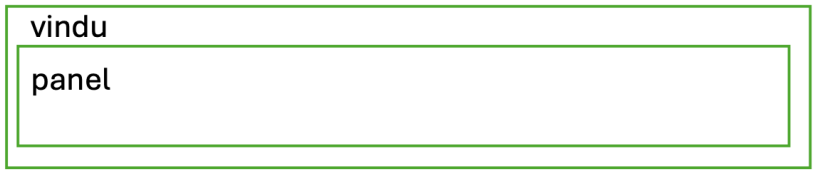


- Vi lager vinduet:

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

- Og tegneflaten som vi kan plassere andre elementer på:

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



Vi skal nå lage det grafiske brukergrensesnittet




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

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

ActionEvent-er håndteres av egne klasser som kaller kontrolleren.

Hendelsestråden til Swing kaller metodene.

Vi skal nå lage det grafiske brukergrensesnittet



- Vi må også lage knappene og koble til hendelseshåndtering:

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

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

- … og legge til knappene i panelet:

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

Vi skal nå lage det grafiske brukergrensesnittet




```
@Override
public void init(Kontroll k) {
    kontroll = k;

    vindu = new JFrame("Teller");
    panel = new JPanel();
    vindu.add(panel);
    vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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

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

    panel.add(verdiLedetekstLabel);
    panel.add(verdiLabel);
```

Legg alle elementer i panelet

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

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

Lukk vindu når man trykker på X

Pakk sammen og gjør synlig midt på skjermen

Observatør-mønster i praksis




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

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

Subjekt

Observatør

Hovedprogrammet lar bruker velge grensesnitt

```java
public class TellerProgram {
    public static void main(String[] args){
        String type = "";
        if(args.length == 0)
        {
            type = "G";

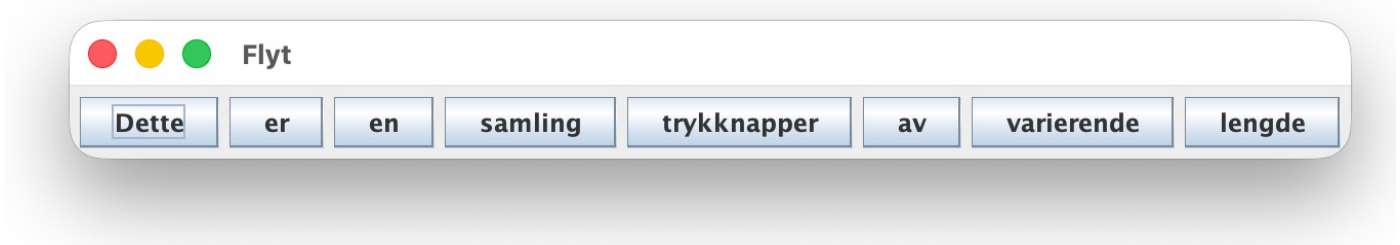
        }
        else
        {
            type = args[0].substring(0,1).toUpperCase();
        }
        Kontroll k = new Kontroll(type);
        k.init();
    }
}
```




```
moskjegs@eduroam-193-157-170-149 del1 % java TellerProgra
Typen er: 'K'
Bruksanvisning:
Tast 1 for å øke med 1
Tast 0 for å nullstille
Tast X for å avslutte
1
Verdien er: 1
0
Verdien er: 0
X
```

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.



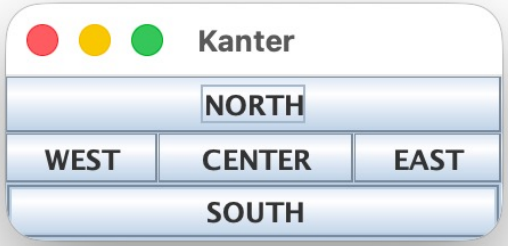




BorderLayout: når vi ønsker faste posisjoner

```
JFrame vindu = new JFrame("Kanter");
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);
```

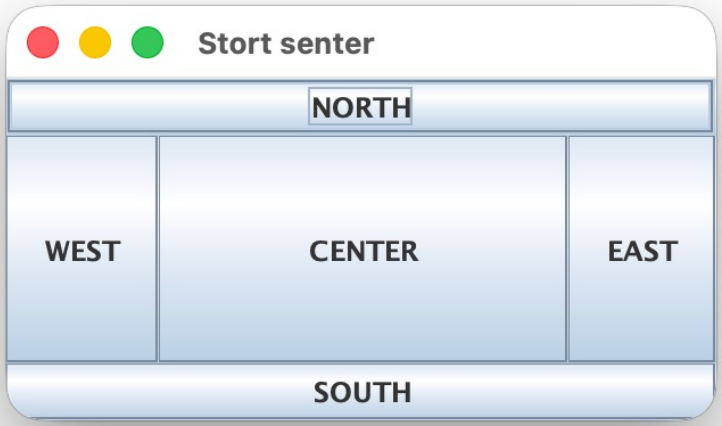


Juster størrelse

```
JFrame vindu = new JFrame("Stort senter");
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);

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

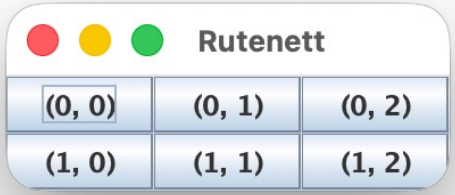


Vi gir et hint om at vi ønsker en annen størrelse på sentrum-knappen. Swing kan velge å ignorere dette om nødvendig.

GridLayout: når vi ønsker et rutenett

```
JFrame vindu = new JFrame("Rutenett");
JPanel panel = new JPanel();
vindu.add(panel);

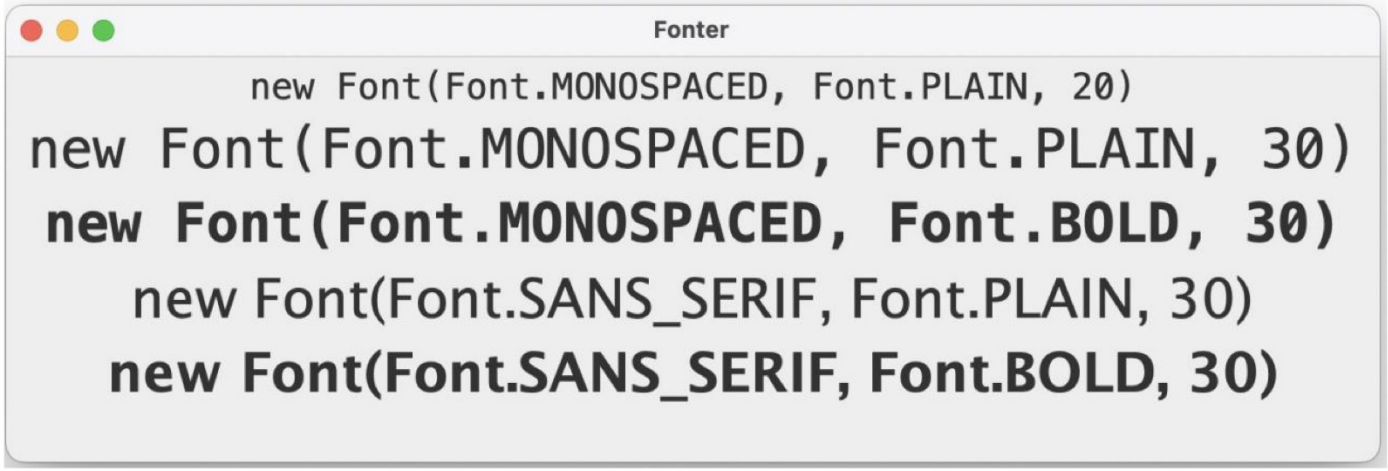
panel.setLayout(new GridLayout(2,3));
panel.add(new JButton("(0, 0)"));
panel.add(new JButton("(0, 1)"));
panel.add(new JButton("(0, 2)"));
panel.add(new JButton("(1, 0)"));
panel.add(new JButton("(1, 1)"));
panel.add(new JButton("(1, 2)"));
```



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

Fonter

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



- Enhet for fontstørrelse er skjermpunkter/pixels

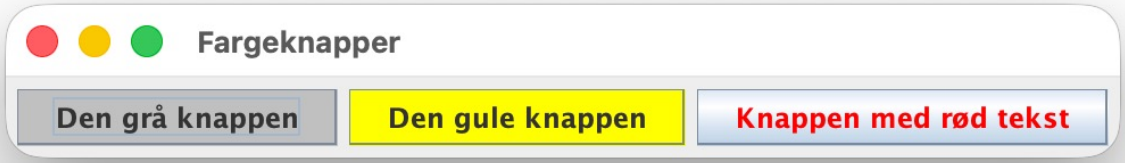
```java
JFrame vindu = new JFrame("Fargeknapper");
JPanel panel = new JPanel();
vindu.add(panel);

JButton knapp1 = new JButton("Den grå knappen");
knapp1.setBackground(Color.LIGHT_GRAY);
panel.add(knapp1);

JButton knapp2 = new JButton("Den gule knappen");
knapp2.setBackground(new Color(255,255,0)/*gul*/);
panel.add(knapp2);

JButton knapp3 = new JButton("Knappen med rød tekst");
knapp3.setForeground(Color.RED);
panel.add(knapp3);
```

Farge på bakgrunn eller tekst



Fargelegge kanter

- Kan være nyttig under feilsøking eller hvis man vil se hvordan Java har strukturert vinduet.




```
JFrame vindu = new JFrame("Test av layout");
vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JPanel ytrePanel = new JPanel();
vindu.add(ytrePanel);
ytrePanel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 5));

JPanel indrePanel1 = new JPanel();
indrePanel1.setBorder(BorderFactory.createLineBorder(Color.RED, 5));
JButton knapp1 = new JButton("En knapp");
indrePanel1.add(knapp1);
ytrePanel.add(indrePanel1);

JPanel indrePanel2 = new JPanel();
indrePanel2.setBorder(BorderFactory.createLineBorder(Color.BLUE, 5));
JButton knapp2 = new JButton("Enda en knapp");
indrePanel2.add(knapp2);
ytrePanel.add(indrePanel2);
```

Layout-eksempler

- Vi tester noen av disse

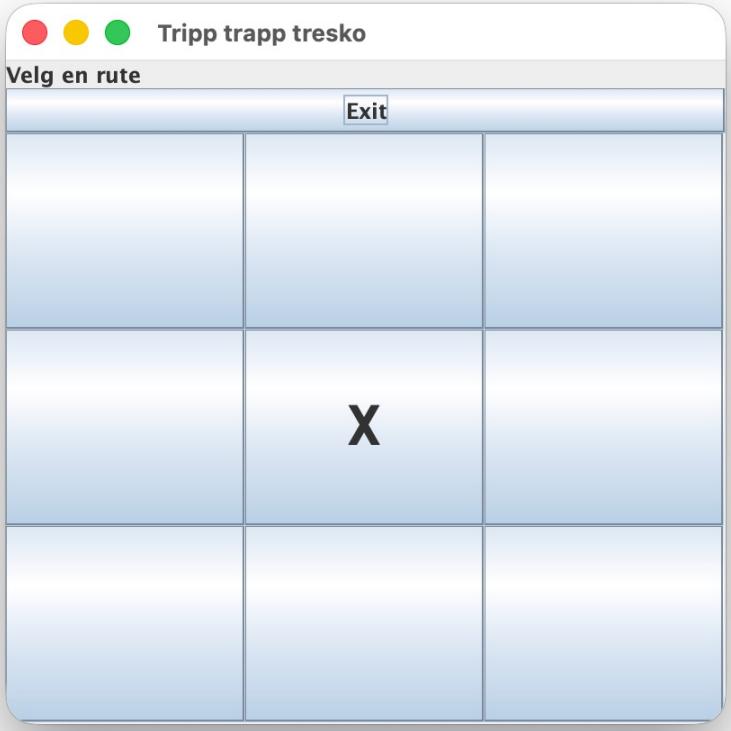
```
JFrame vindu = new JFrame("Test av layout");
vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

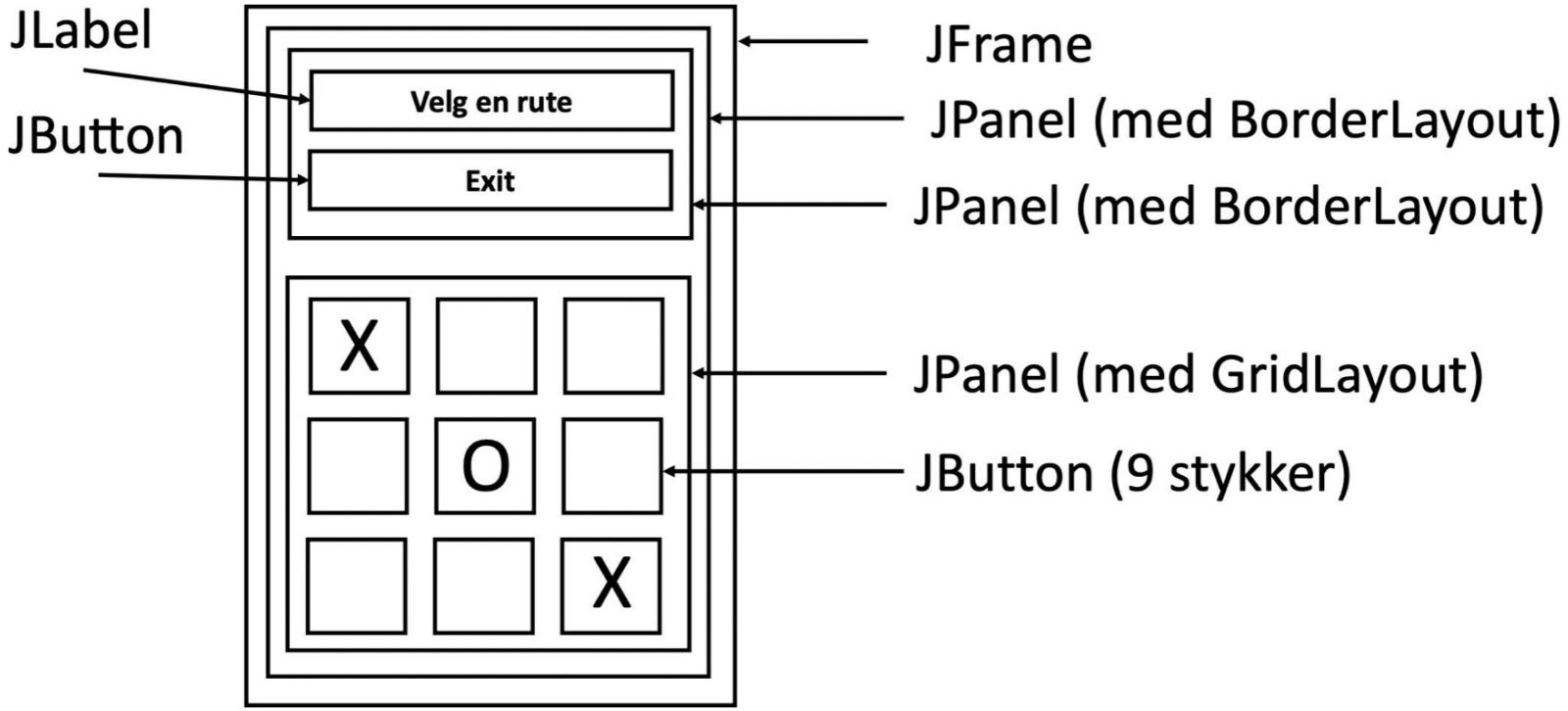
panel.setLayout(new GridLayout(2, 3));
JButton knapp1 = new JButton("Dette");
knapp1.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));
panel.add(knapp1);
JButton knapp2 = new JButton("er");
knapp2.setBackground(Color.YELLOW);
panel.add(knapp2);
JButton knapp3 = new JButton("mange");
knapp3.setForeground(Color.RED);
panel.add(knapp3);
JButton knapp4 = new JButton("forskjellige");
panel.add(knapp4);
JButton knapp5 = new JButton("knapper");
panel.add(knapp5);
JButton knapp6 = new JButton("!");
panel.add(knapp6);
```

Nytt eksempel – tripp-trapp-tresko (TTT)

- Vi skal gå gjennom spillet fra sist forelesning
- Ting å tenke på når man skal lage et nytt GUI-program:
  - Planlegge utseendet til vinduet
  - Designe programmet: F.eks MVC
  - Planlegge brukerdialogen / hva skjer ved knappetrykk osv.
  - Programmere det hele!



Design-skisse for programmet vi skal lage



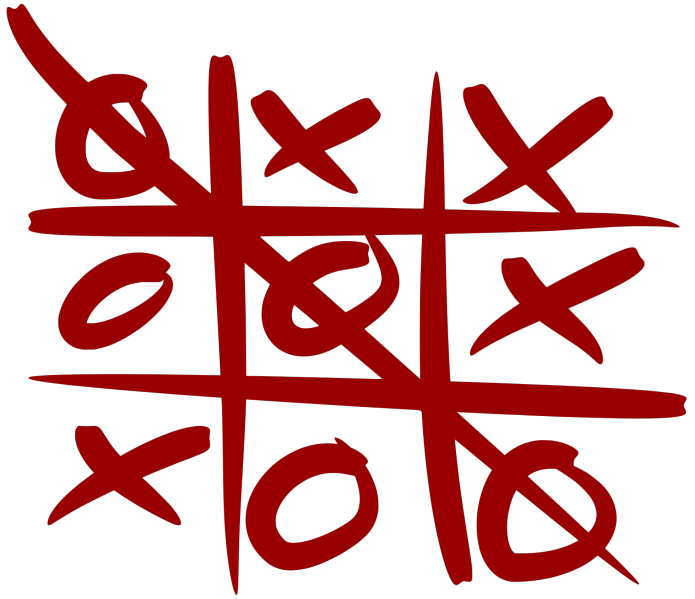
MVC-struktur

- **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 mye annet!
  - Hvilke regler har spillet?
    - Avveining: hvor mye logikk skal vi plassere i modellen, og hvor mye skal vi plassere i kontrolleren?
- **View** er skjermbildet skissert fra forrige slide
- **Kontrolleren** er ansvarlig for:
  - Hvem sin tur er det?
  - Hvordan håndtere lovlige/ulovlige trekk?
  - Kommunisere at 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;
    String vinner = "";

    Modell () {
        for (int rx = 0;  rx < 3;  ++rx)
            for (int kx = 0;  kx < 3;  ++kx)
                brett[rx][kx] = ' ';
    }
```



(fra Wikipedia-siden «Tac tac toe»)

Modellen har også logikk for spillets tilstand

- 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

```
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;
}
```

Når vinner man?

```
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 spiller vinner hvis O eller X har tre markeringer i en rad, kolonne eller diagonalt.

```
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(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");
        }
    }
    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

Kontrolleren kjenner
både GUI og modell

Maskinen er ikke en veldig
smart motspiller – den bare
trekker tilfeldige posisjoner til
den finner er lovlig trekk.

Math.random brukes til
å trekke «tilfeldige»
tall.

```
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');
    gui.markerTrekk(r, k, 'O');
    if (modell.harVunnet('O')) {
        gui.visStatus('O' + " vant");
        modell.noterVinner("Du");
        return;
    }
    laMaskinenTrekke();
    if (modell.erSpilletFerdig())
        return;
    if (modell.erBrettetFullt()) {
        modell.noterUavgjort();
        gui.visStatus("Uavgjort");
    }
    else
        gui.visStatus("Velg en rute");
}
```

Håndtering av brukerens trekk

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

Sjekk om spillet er ferdig (maskinen vant) eller om brettet er fullt (uavgjort).

Brukergrensesnittet

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

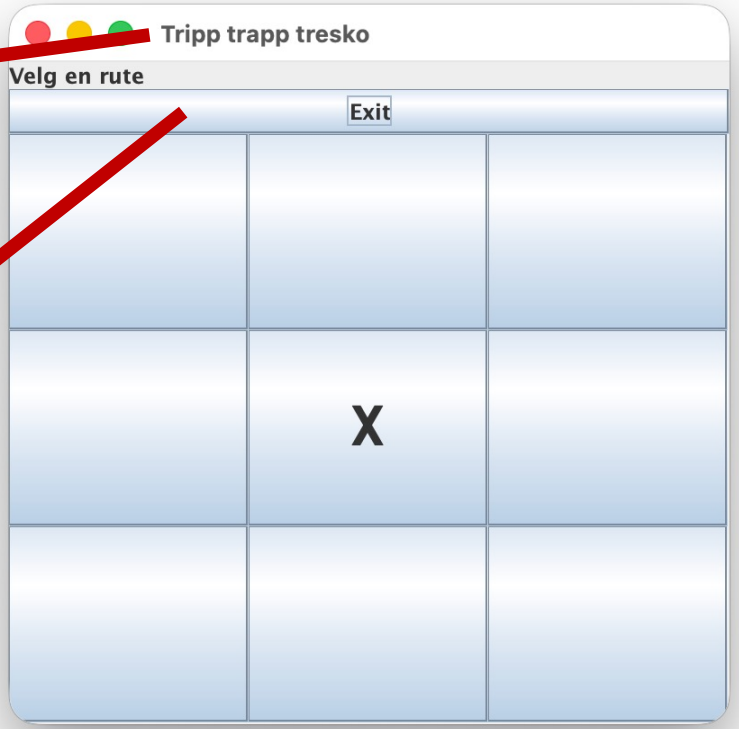
        panel = new JPanel();
        panel.setLayout(new BorderLayout());
        vindu.add(panel);

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

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

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

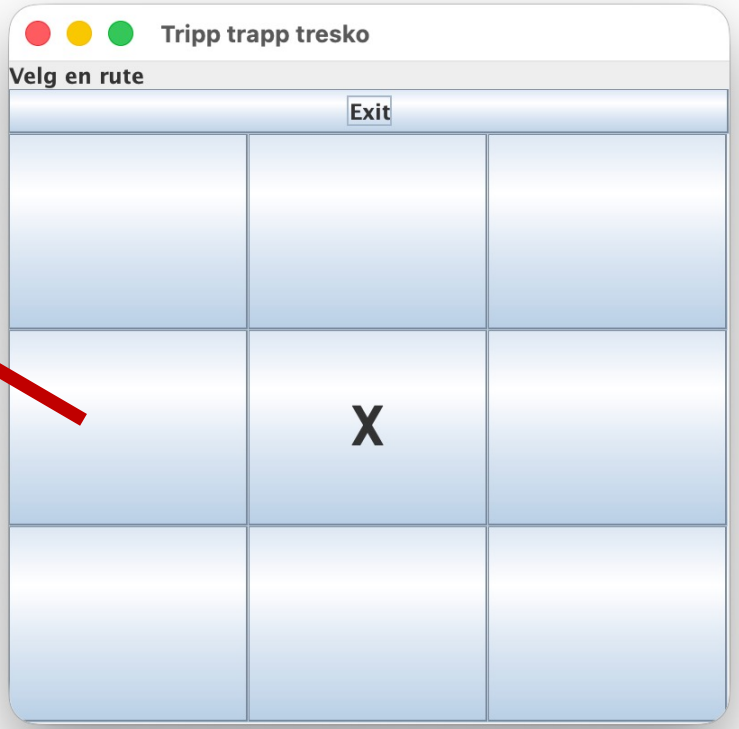
Variabler for de ulike elementene.



Brukergrensesnittet

```
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.setSize(400,400);
vindu.setLocationRelativeTo(null);
vindu.setVisible(true);
```



Håndtere hendelser (brukervalg)

```
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.setSize(400,400);
vindu.setLocationRelativeTo(null);
vindu.setVisible(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 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)

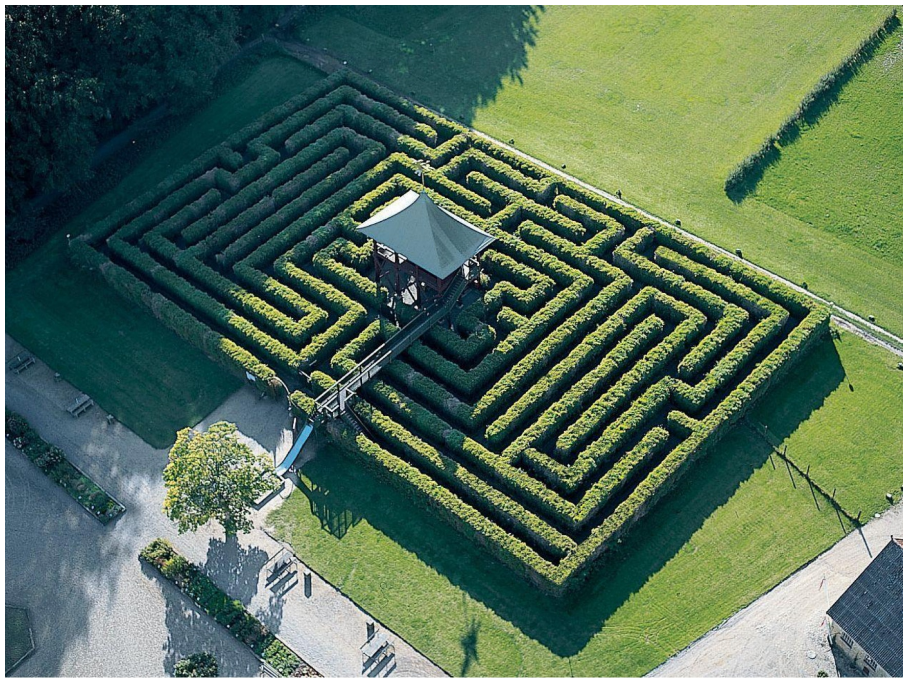
JFrame
- add (JComponent c)
- int EXIT_ON_CLOSE
- pack()
- setDefaultCloseOperation (int i)
- setVisible (boolean b)

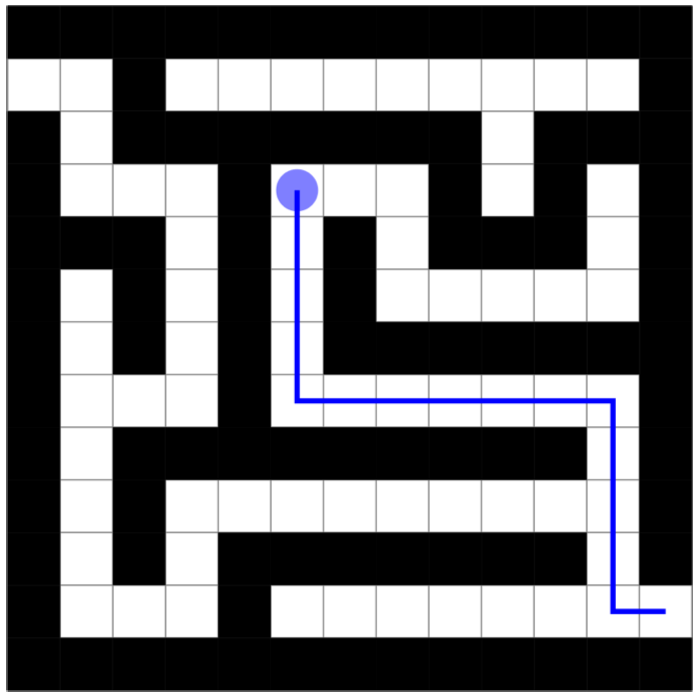
JLabel ext JComponent
JPanel ext JComponent
- add (JComponent c)
- setLayout(LayoutManager m)

interface LayoutManager

Innlevering 6

- Fortsatt tid igjen (innelvering 4. mai)
- Her skal man lage et program som finner veien gjennom en labyrint





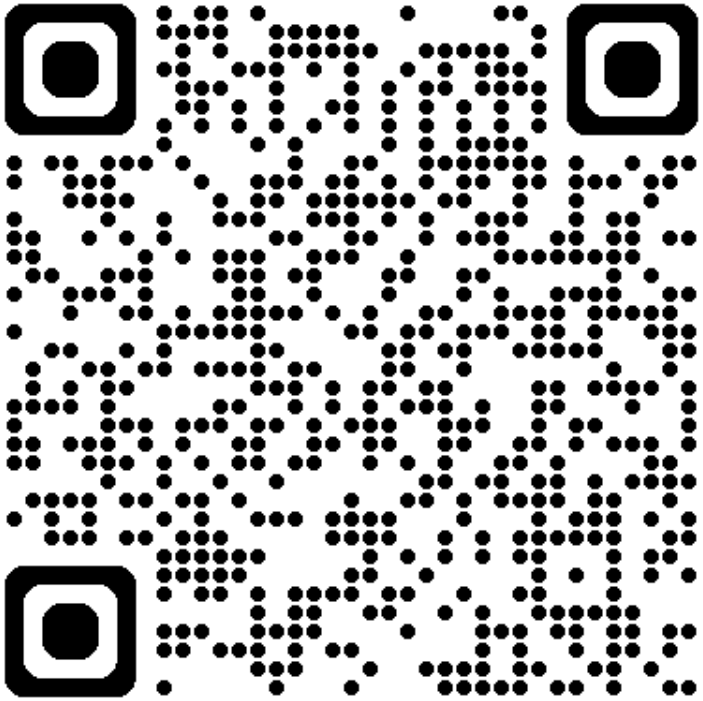
Oppsummering

- Nå kan vi programmere brukergrensesnitt i Java!
  - Posisjonering av vinduet
  - Varianter av layout
  - *Hendelseshåndtering*
- Det er ofte lurt å tegne 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 representere
  - View – brukergrensesnittet, det som brukeren interagerer med
  - Controller – holder styr på forholdet mellom modell og brukergrensesnitt
- Neste uke fortsetter Dag



Læringsmiljøundersøkelsen ved MN

- For første gang gjennomfører MN en egen læringsmiljøundersøkelse.
- Undersøkelsen skal bidra til fakultetets arbeid med læringsmiljøet.
- Alle studenter som svarer på undersøkelsen innen 7. mai er med i trekningen av *ett* av totalt 10 universalgavekort på 500,-. Det gjennomføres en ny trekning annenhver dag i perioden 15. april til 7. mai.



https://www.mn.uio.no/kurt/aktuelt/aktuelle-saker/2026/leringsmiljoundersokelse-ved-mn.html