GUI («Graphical User Interface») del 2

- Å programmer GUI
  - Grafisk design
  - Programoppbygging (MVC)
  - Sekvensdiagrammer
  - Programmeringen
- OO-strukturen av Swing
- Tilpasning av GUI-utseendet
- Tripp-trapp-tresko

Se også

- Big Java kapittel 10–11
- Programkoden i https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/programmer/GUI/

Hvordan lage GUI-programmer

Siden det kan skje så mye og metodekall kan oppstå når man minst venter det, er det viktig å ha god struktur på GUI-programmene.

1. Design av skjermbildet
2. Bestemme hvilke metoder som tar seg av hva
3. Avgjøre kontrollflyten

Eksempel: en teller

Hva skal en teller gjøre?

- Det skal vise antall trykk.
- For hvert trykk skal antallet øke med 1.
- Det skal også finnes en knapp for nullstilling.

Design

Vi bør starte med å designe skjermbildet fordi det er så veldig arbeidsomt å endre det senere:

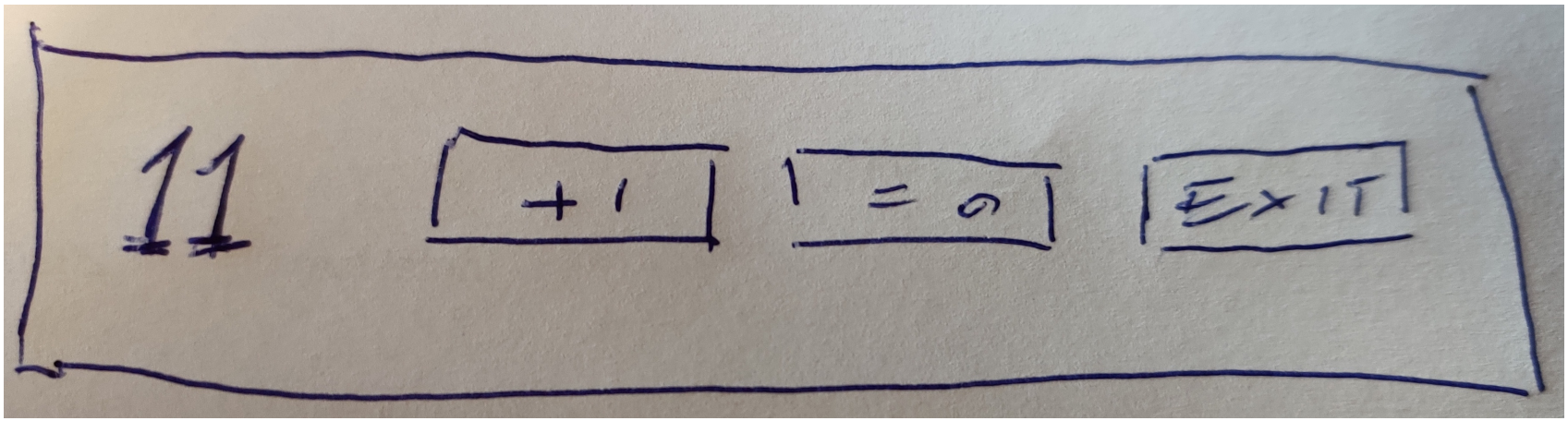
- Hvilke GUI-elementer skal vi ha med?
- Hvordan skal de stå i forhold til hverandre?

Design

Vi bør starte med å designe skjermbildet fordi det er så veldig 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:



MVC-struktur

Det er viktig å gi programmet en god struktur, og MVC er et hjelpemiddel:

Model lagrer det som trengs for å representere programmets tilstand.
I vårt program: tellerverdien

MVC-struktur

Det er viktig å gi programmet en god struktur, og MVC er et hjelpemiddel:

Model lagrer det som trengs for å representere programmets tilstand.
I vårt program: tellerverdien

View oppretter GUI-vinduet og tar seg av brukerinteraksjonen (dvs knappetrykk).

MVC-struktur

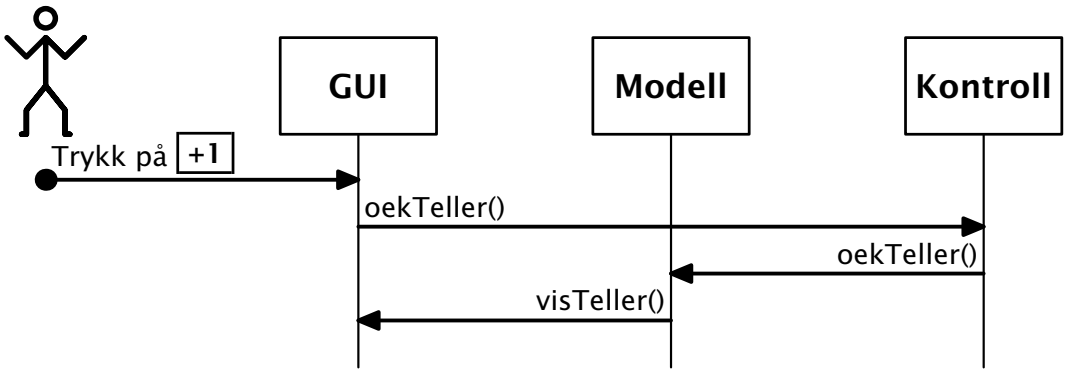
Det er viktig å gi programmet en god struktur, og MVC er et hjelpemiddel:

Model lagrer det som trengs for å representere programmets tilstand.
I vårt program: tellerverdien

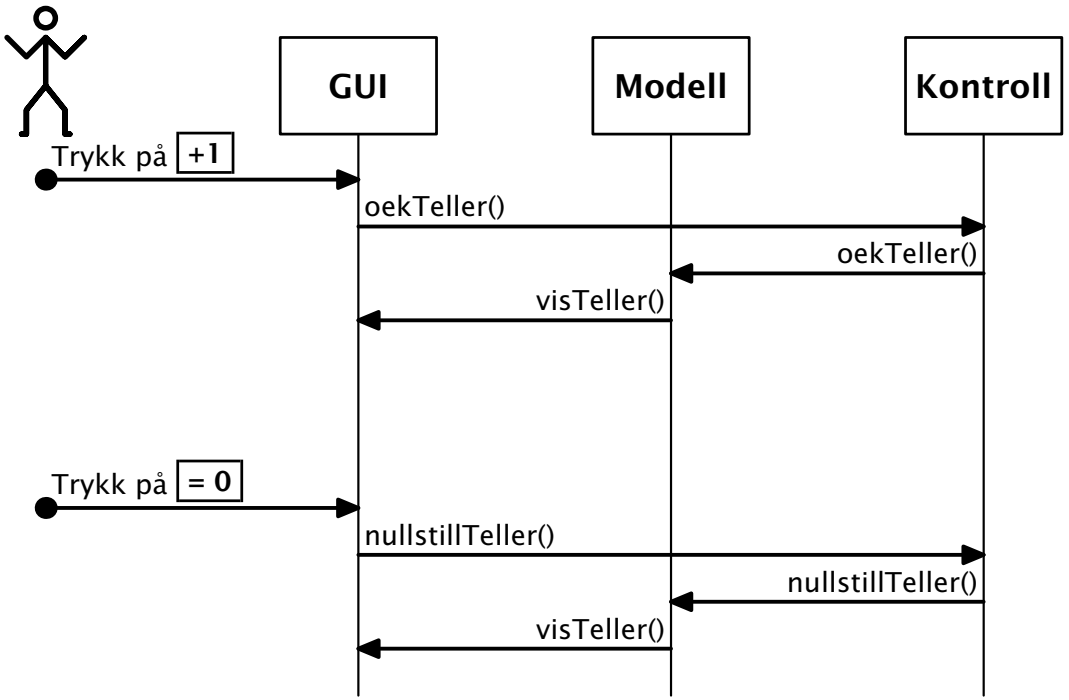
View oppretter GUI-vinduet og tar seg av brukerinteraksjonen (dvs knappetrykk).

Controller styrer det hele.

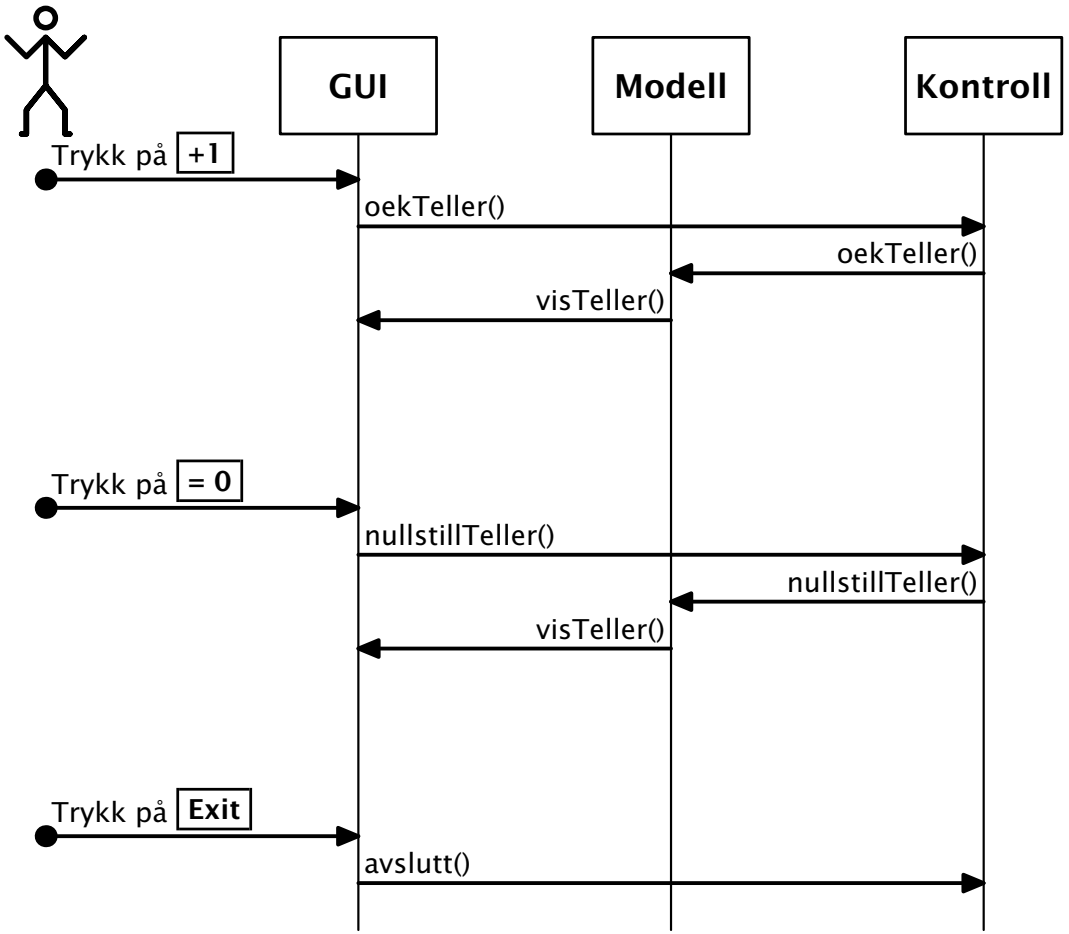
UML sekvensdiagram



UML sekvensdiagram



UML sekvensdiagram



Programmet: Modellen

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

    Modell (GUI g) {
        gui = g;
    }

    void oekTeller () {
        ++tellerverdi;
        gui.visTeller(tellerverdi);
    }

    void nullstillTeller () {
        tellerverdi = 0;
        gui.visTeller(tellerverdi);
    }
}
```

Programmet: Kontrollen

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

    Kontroll () {
        gui = new GUI(this);
        modell = new Modell(gui);
    }

    void oekTeller () {
        modell.oekTeller();
    }

    void nullstillTeller () {
        modell.nullstillTeller();
    }

    void avslutt () {
        System.exit(0);
    }
}
```

Programmet: GUI-en

GUI 1: Start

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

    GUI (Kontroll k) {
        kontroll = k;
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
```

GUI 2: Vinduet

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

GUI 2: Vinduet

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

GUI 3: Hovedtegneflaten

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

GUI 2: Vinduet

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

GUI 3: Hovedtegneflaten

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

GUI 4: Tellerinformasjonen

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

GUI 5: Trykknappen «+1»

```
tell = new JButton(" +1 ");
class OekTeller implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.oekTeller();
    }
}
tell.addActionListener(new OekTeller());
```

GUI 5: Trykknappen «+1»

```
tell = new JButton(" +1 ");
class OekTeller implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.oekTeller();
    }
}
tell.addActionListener(new OekTeller());
```

GUI 6: Trykknappen «=0»

```
resett = new JButton(" = 0 ");
class Nuller implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.nullstillTeller();
    }
}
resett.addActionListener(new Nuller());
```

GUI 7: Stoppknappen

```
slutt = new JButton("Exit");
class Stopper implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.avslutt();
    }
}
slutt.addActionListener(new Stopper());
```

GUI 7: Stoppknappen

```
slutt = new JButton("Exit");
class Stopper implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.avslutt();
    }
}
slutt.addActionListener(new Stopper());
```

GUI 8: Legg alle komponentene på tegneflaten

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

GUI 7: Stoppknappen

```
slutt = new JButton("Exit");
class Stopper implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.avslutt();
    }
}
slutt.addActionListener(new Stopper());
```

GUI 8: Legg alle komponentene på tegneflaten

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

GUI 9: Klargjør GUI-vinduet

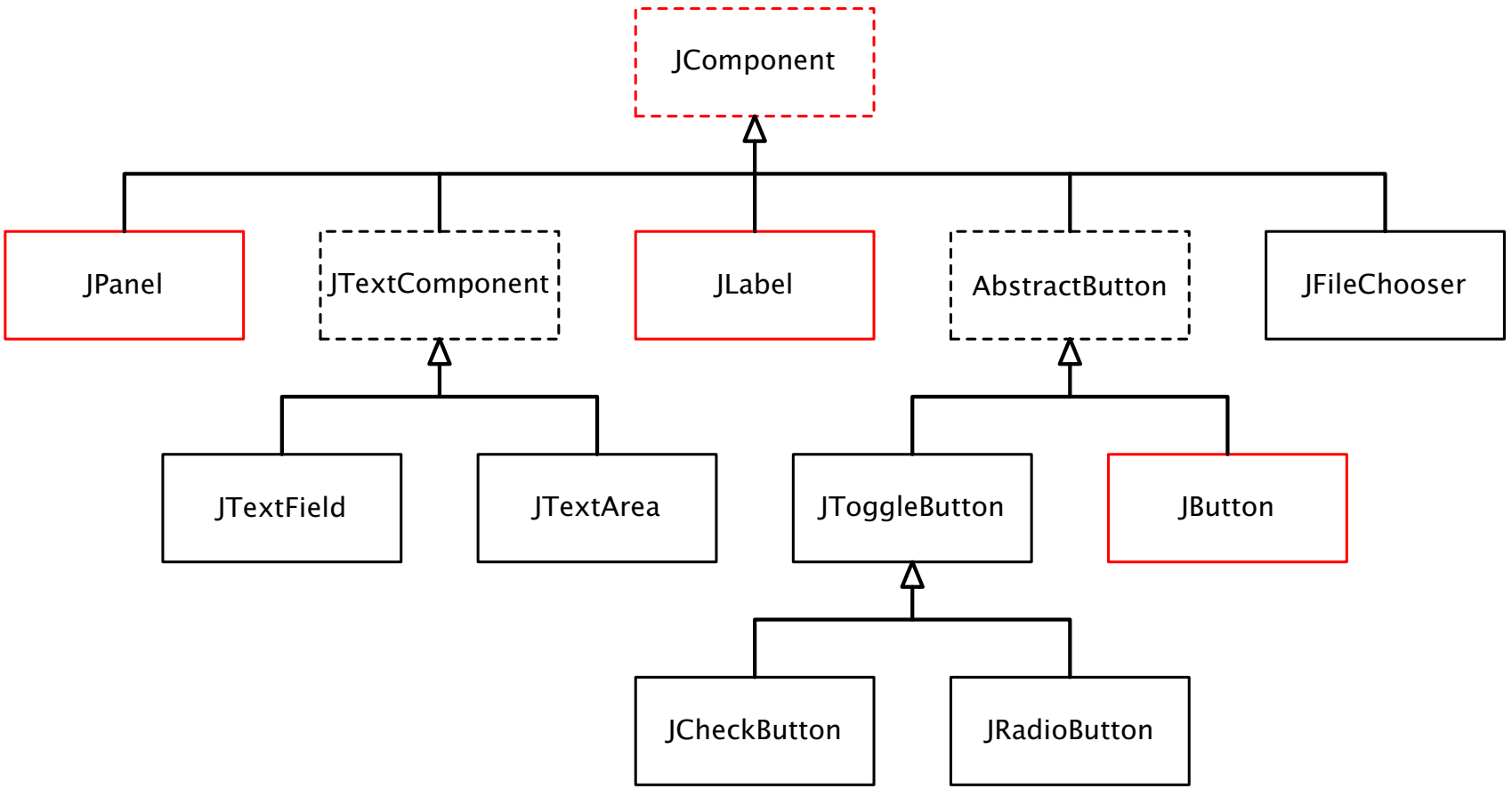
```
vindu.pack();  vindu.setVisible(true);
```

Programmet: Hovedprogrammet

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

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

Swing er objektorientert



I IN1010 skal vi bruke JFrame og de markert med rødt.

Variere utseendet

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

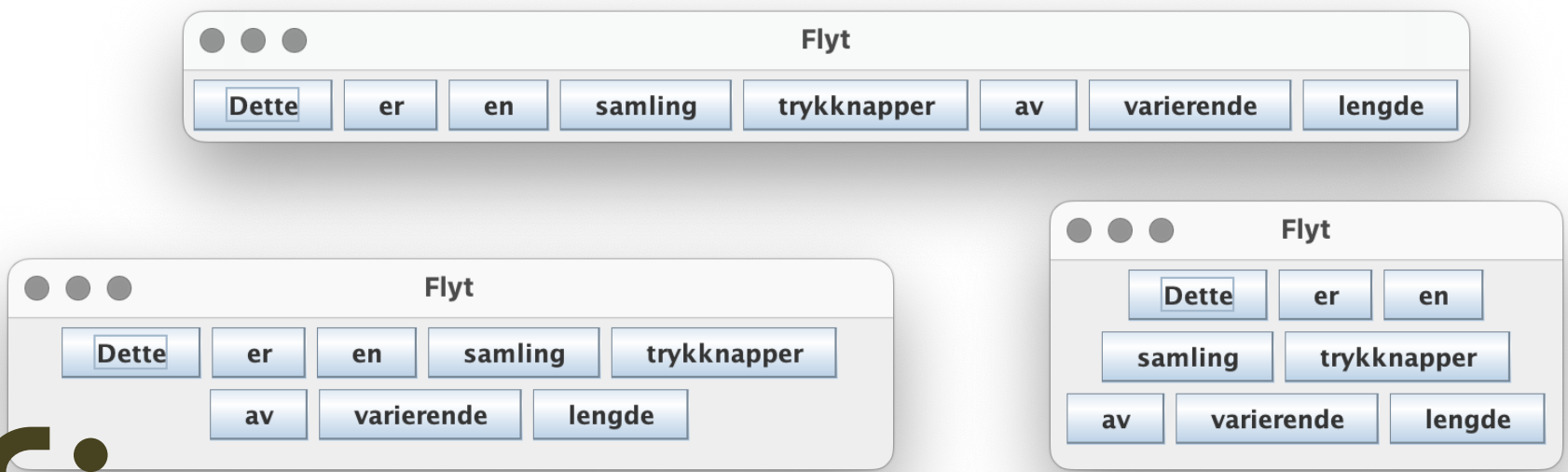
- plassering i tegnepanelet
- størrelse
- font
- farge (både tekst og bakgrunn)
- ramme

Plassering på tegneflaten

For alle tegneflater (dvs JPanel) kan vi angi hvordan innholdet skal plasseres.

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:

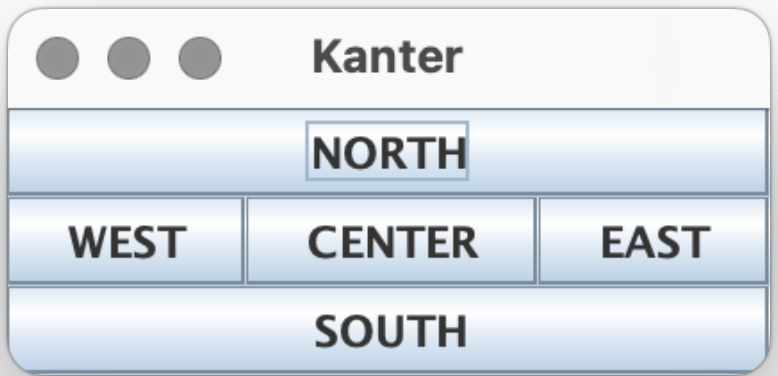


BorderLayout

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);
```

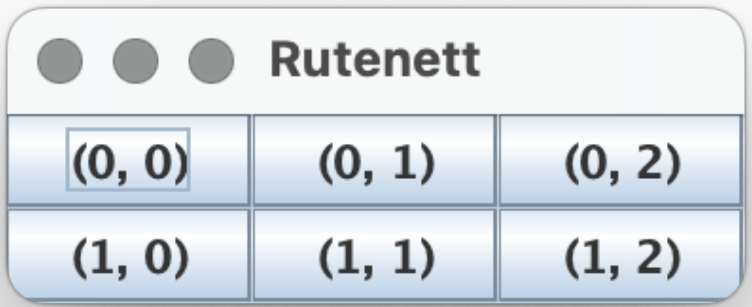


GridLayout

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)"));
```

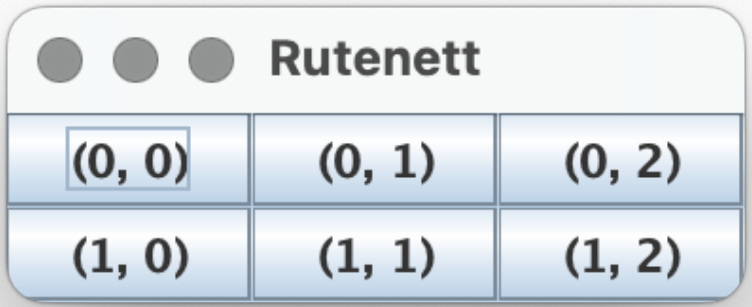


GridLayout

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)"));
```



NB!

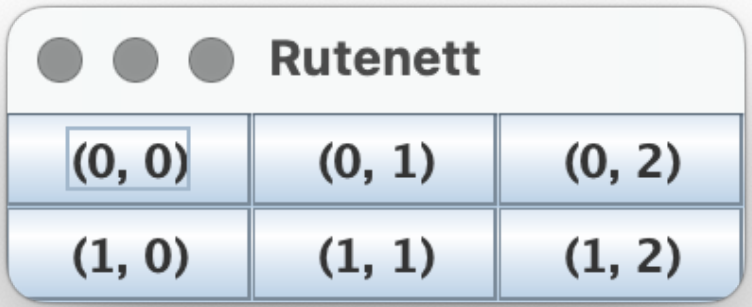
I 2-dimensjonale array-er angir vi alltid posisjonen med **(rad, kolonne)**.

GridLayout

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)"));
```



NB!

I 2-dimensjonale array-er angir vi alltid posisjonen med **(rad, kolonne)**.

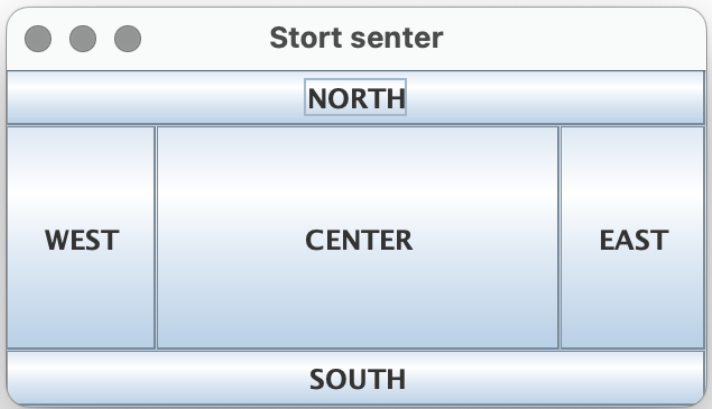
Rader går vannrett, kolonner går loddrett!

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);
```

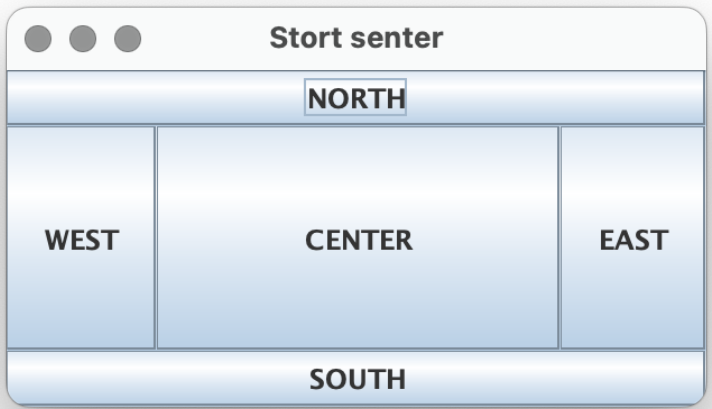


Størrelser

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);
```

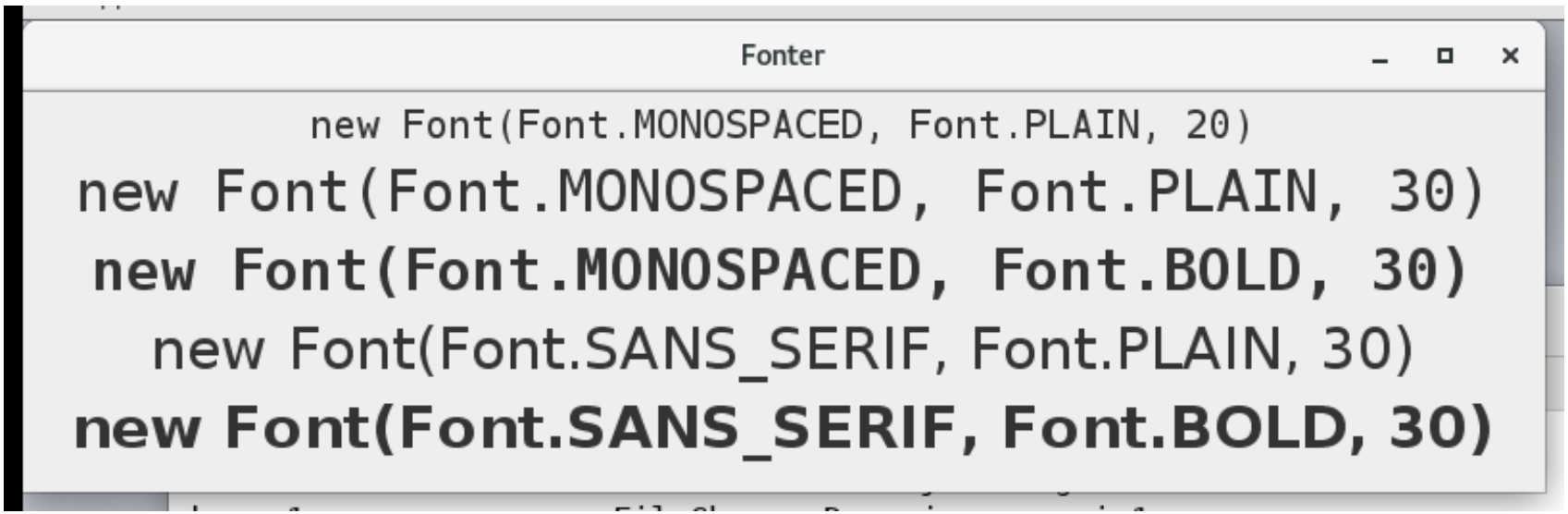


NB!

Dette er bare *hint*! Swing kan velge å overstyre dette.

Fonter

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



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

Bakgrunnsfarge

I Swing kan vi angi bakgrunnsfarge i elementene:

```
JButton knapp1 = new JButton("Den ene knappen");
knapp1.setOpaque(true);
knapp1.setBackground(Color.LIGHT_GRAY);
knapp1.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));
panel.add(knapp1);

JButton knapp2 = new JButton("Den andre knappen");
knapp2.setOpaque(true);
knapp2.setBackground(new Color(255,255,0)/*gul*/);
knapp2.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));
panel.add(knapp2);
```

Fargene

Farger lages med **new Color(r,g,b)** men noen farger er predefinert:

BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW

Fargene

Farger lages med **new Color(r,g,b)** men noen farger er predefinert:

BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW

Synlighet

Noen «look-and-feel» lar noen elementer være gjennomskinnelige, dvs at bakgrunnen ikke tegnes så vi kan se hva som ligger bak. Når vi skal angi en bakgrunnsfarge, bør vi derfor slå av dette med ***xxx*.setOpaque(true)**.

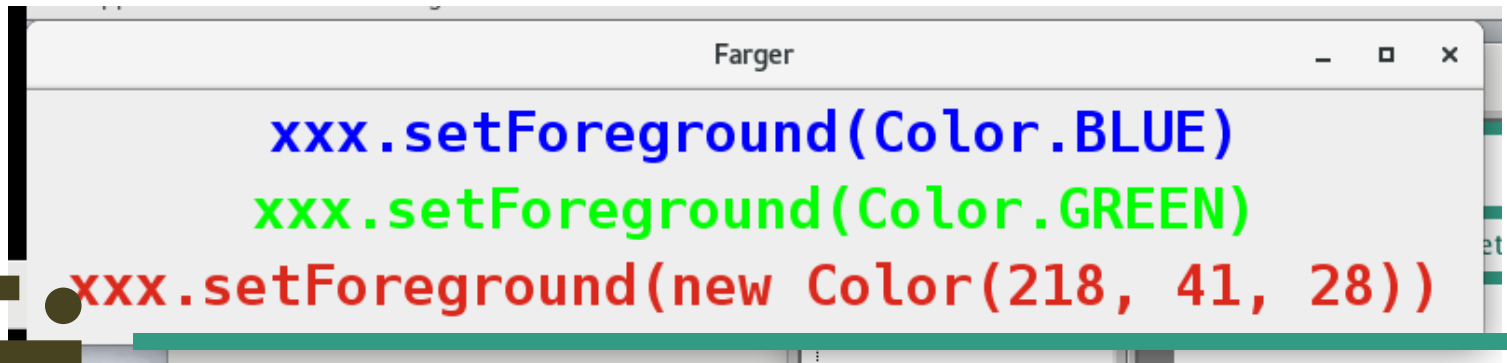
Tekstfarge

Vi kan også angi fargen på teksten i elementene:

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

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

JLabel uio = new JLabel("xxx.setForeground(new Color(218, 41, 28))");
uio.setForeground(new Color(218, 41, 28));
uio.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));
panel.add(uio);
```



Rammer

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

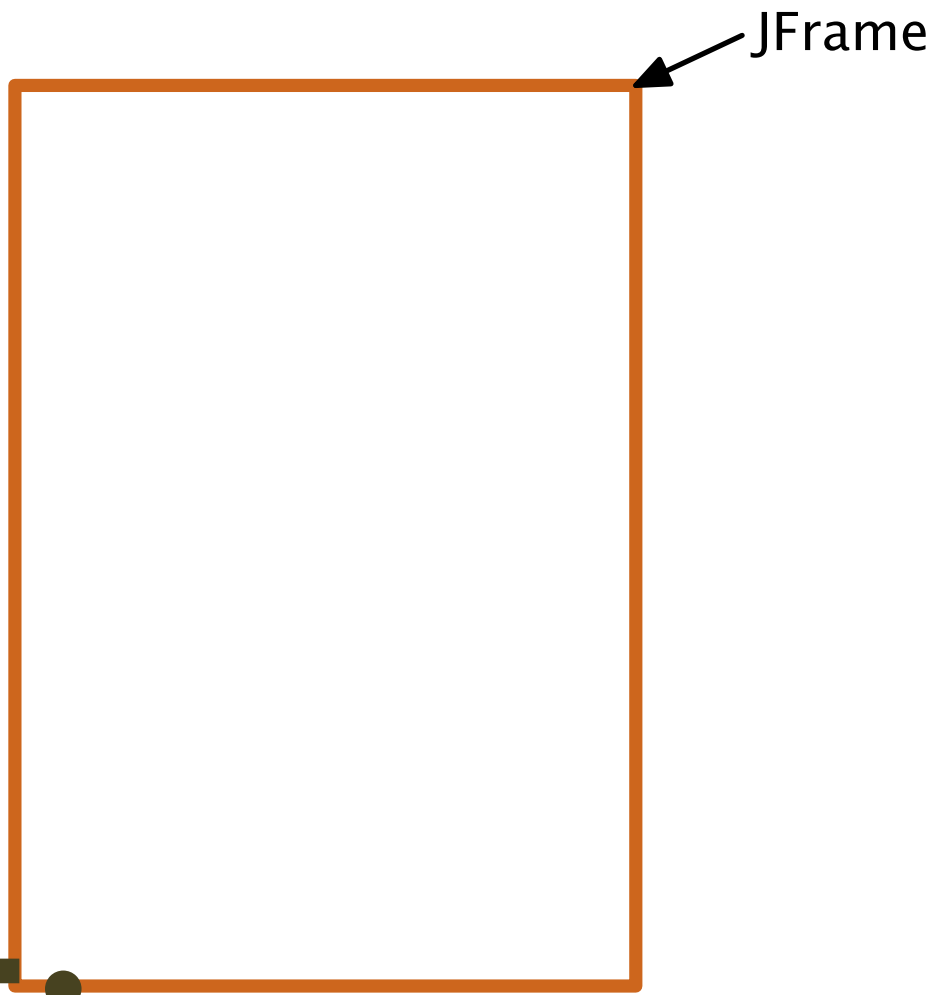
(I IN1010 vil vi kun bruke standardramme og LineBorder.)

Et nytt eksempel: Tripp-trapp-tresko

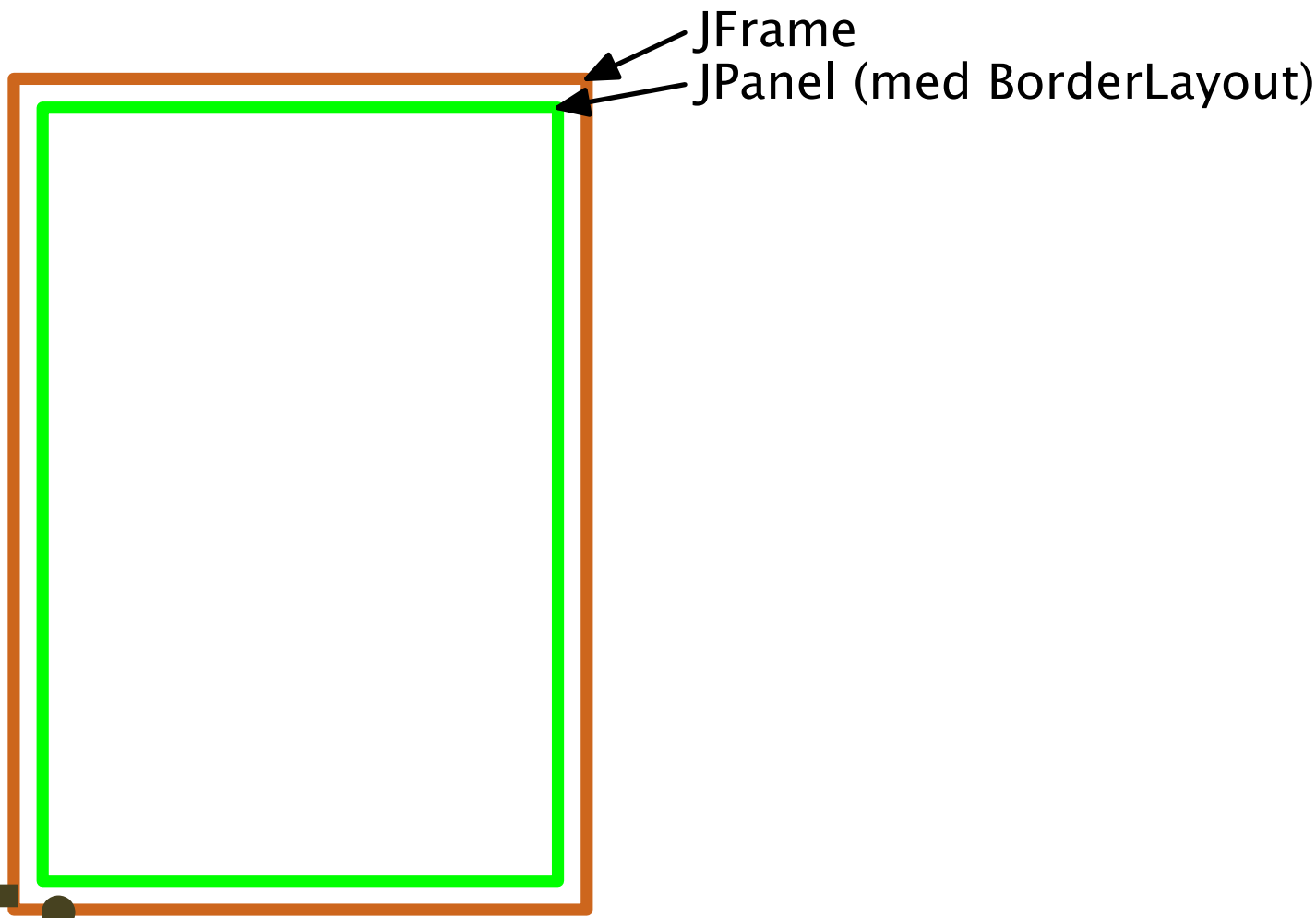
Rekkefølgen når man skal skrive et GUI-program:

1. Tegne vinduet
2. MVC-oversikt
3. Sekvensdiagram
4. Programmering

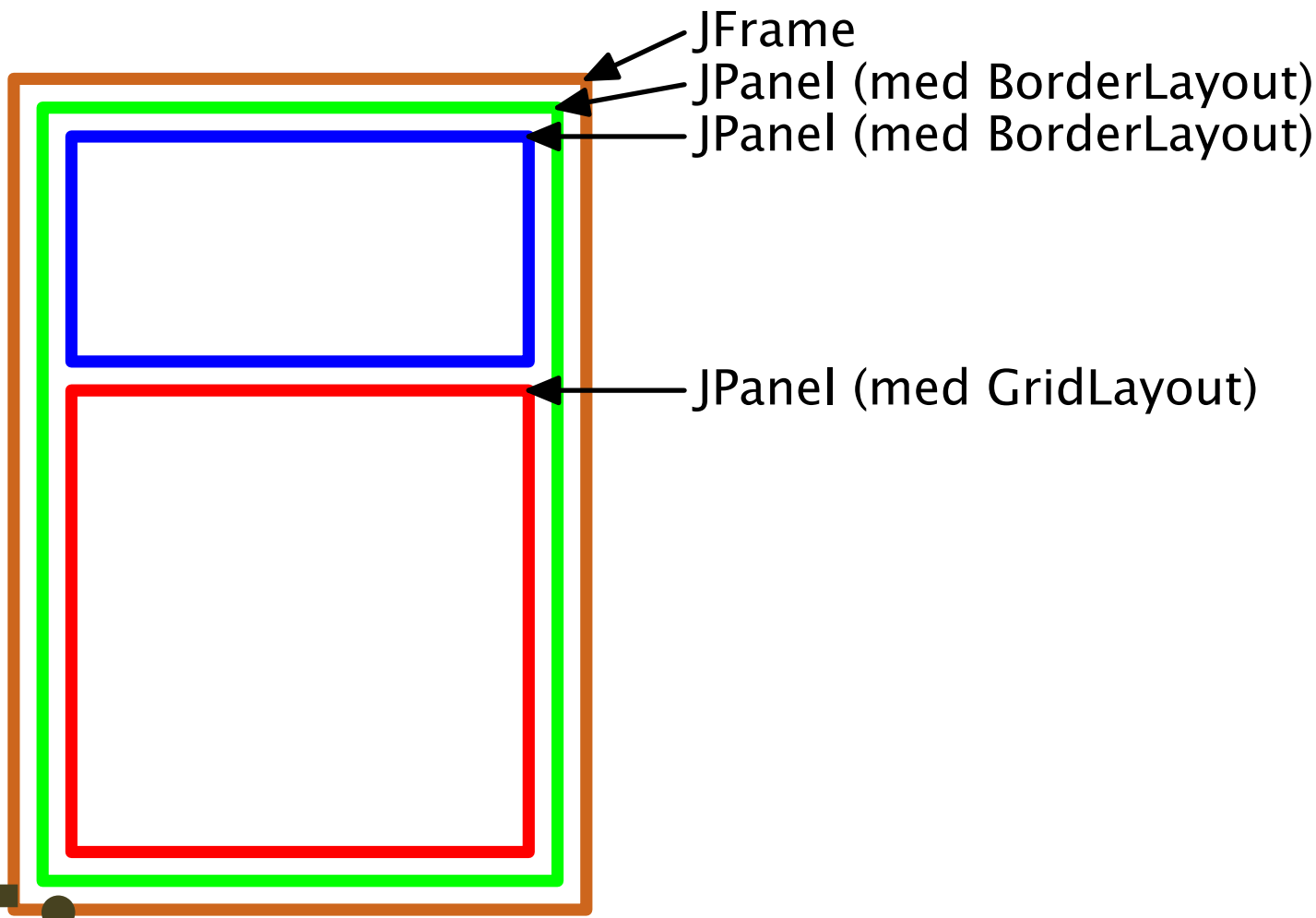
TTT: Design



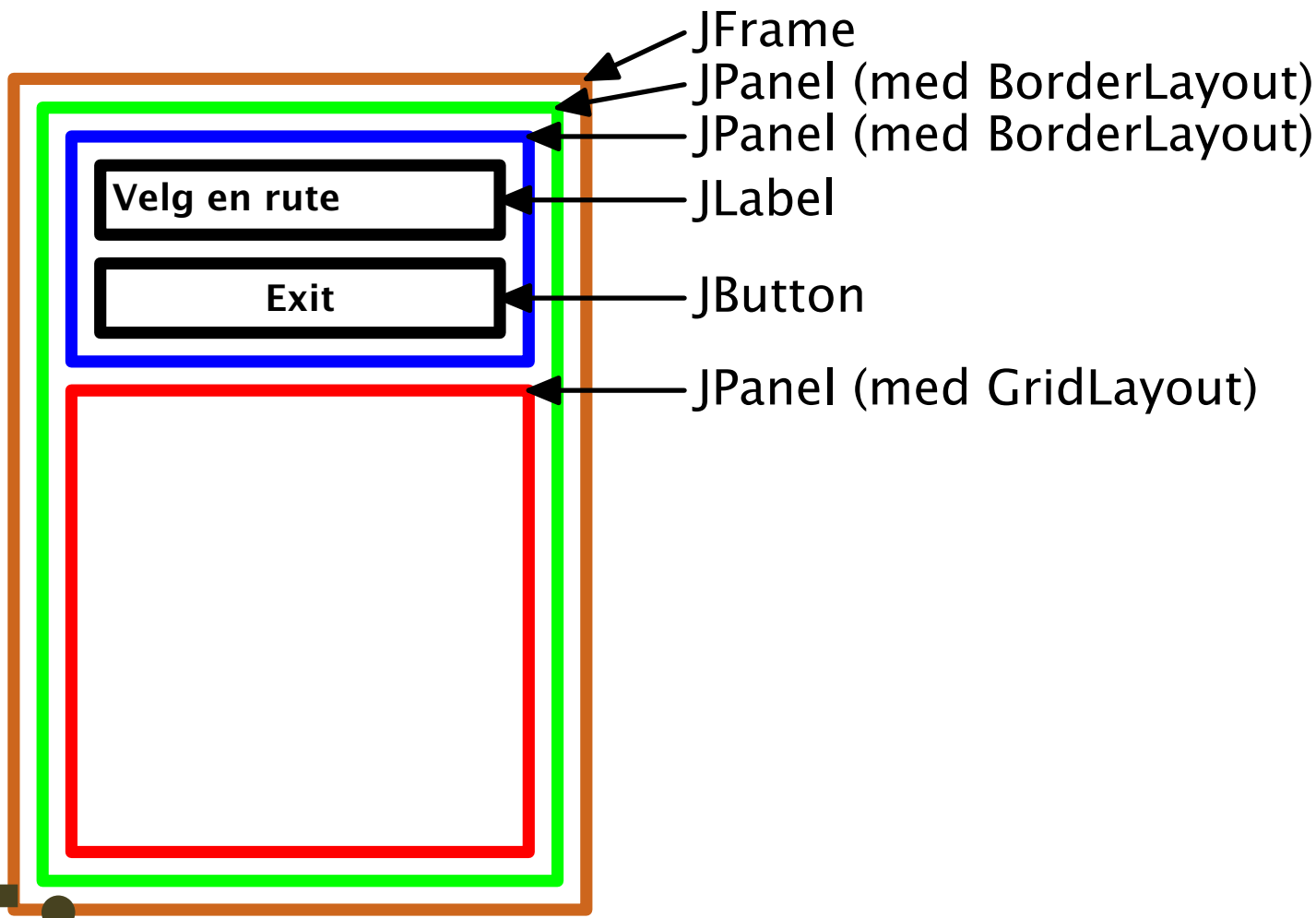
TTT: Design



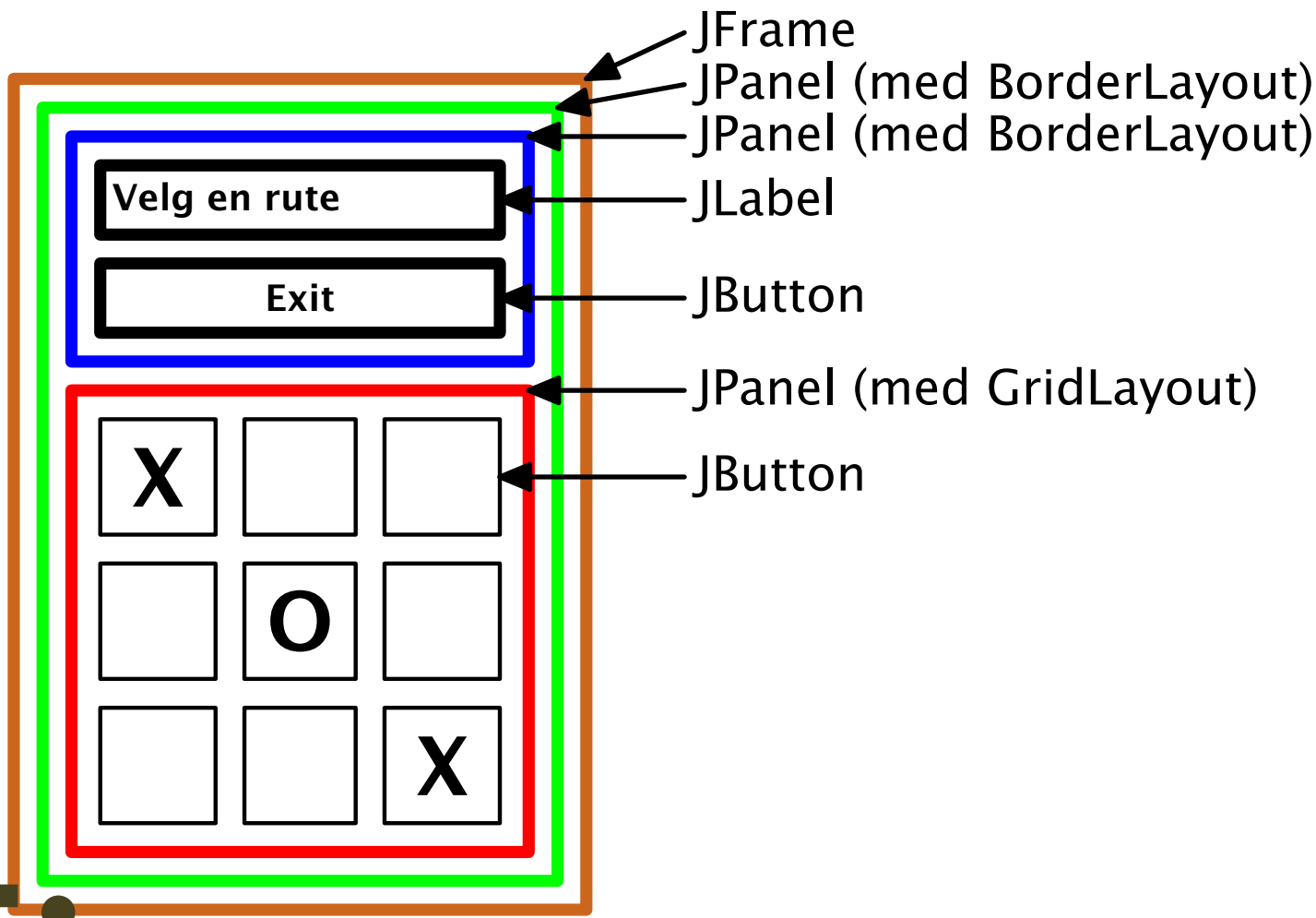
TTT: Design


JFrame
JPanel (med BorderLayout)
JPanel (med BorderLayout)
JPanel (med GridLayout)

TTT: Design


JFrame
JPanel (med BorderLayout)
JPanel (med BorderLayout)
Velg en rute
JLabel
Exit
JButton
JPanel (med GridLayout)

TTT: Design


JFrame
JPanel (med BorderLayout)
JPanel (med BorderLayout)
Velg en rute
JLabel
Exit
JButton
JPanel (med GridLayout)
X
JButton
O
X

TTT: MVC-struktur

Modell bør inneholde

TTT: MVC-struktur

Modell bør inneholde

- brett: en 3×3-array med innhold av rutene (' ', 'X', 'O')

TTT: MVC-struktur

Modell bør inneholde

- brett: en 3×3-array med innhold av rutene (' ', 'X', 'O')
- antTrekk: antall trekk gjort

TTT: MVC-struktur

Modell bør inneholde

- brett: en 3×3-array med innhold av rutene (' ', 'X', 'O')
- antTrekk: antall trekk gjort
- spilletErFerdig: spillets status

TTT: MVC-struktur

Modell bør inneholde

- brett: en 3×3-array med innhold av rutene (' ', 'X', 'O')
- antTrekk: antall trekk gjort
- spilletErFerdig: spillets status

View oppretter GUI-vinduet og registrerer knappetrykk.

TTT: MVC-struktur

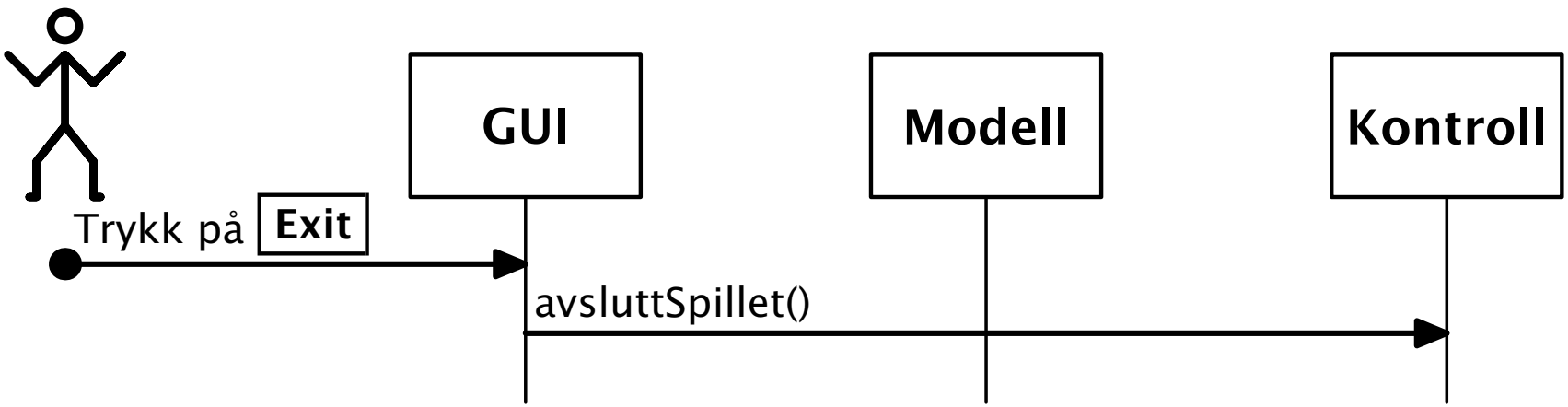
Modell bør inneholde

- brett: en 3×3-array med innhold av rutene (' ', 'X', 'O')
- antTrekk: antall trekk gjort
- spilletErFerdig: spillets status

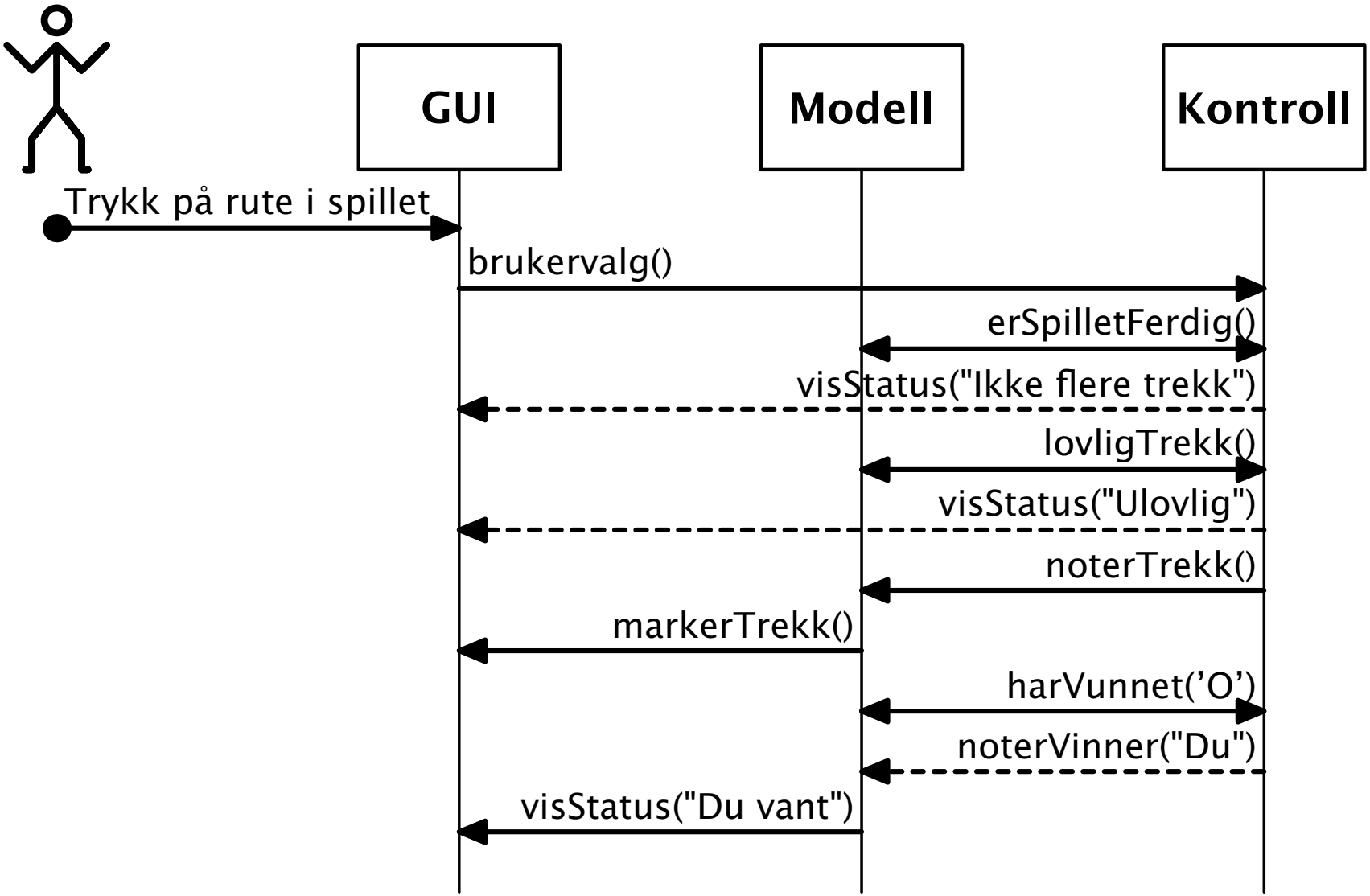
View oppretter GUI-vinduet og registrerer knappetrykk.

Controller styrer det hele.

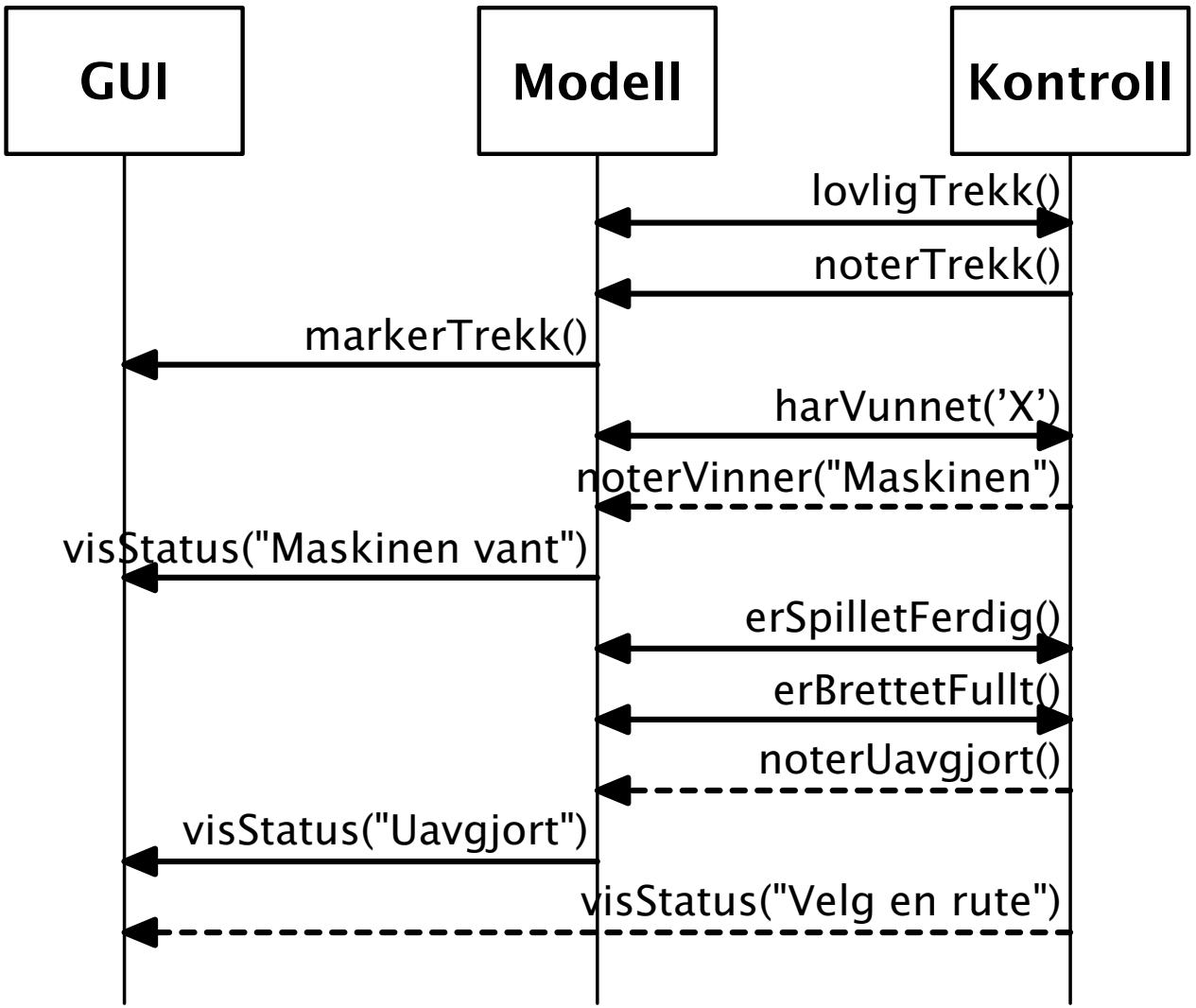
TTT: Sekvensdiagram



Fase 3: Sekvensdiagram



Fase 3: Sekvensdiagram



Programmet: Modellen

Variabler og konstruktør:

```
class Modell {
    GUI gui;
    char[][] brett = new char[3][3];
    int antTrekk = 0;
    boolean spilletErFerdig = false;

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

Diverse operasjoner:

```
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;
}
```

Sjekk om en gitt spiller har vunnet:

```
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;
}
```

Programmet: Kontrollen

Variabler, konstruktør, startmetode og avslutning:

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

    Kontroll () {
        gui = new GUI(this);
        modell = new Modell(gui);
    }

    void startSpillet () {
        laMaskinenTrekke();
    }

    void avsluttSpillet () {
        System.exit(0);
    }
```

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");
}
```

Når datamaskinen skal trekke:

```
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");
}
```

Nyttig hjelperutine for å velge tilfeldig:

```
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;
}
```

Programmet: GUI-en

GUI: Variabler og første del av konstruktøren:

```
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 2: Sett opp hovedtegnepanel og konsoll med statusinformasjon og stoppknapp:

```
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");
class Stoppbehandler implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        kontroll.avsluttSpillet();
    }
}
stoppknapp.addActionListener(new Stoppbehandler());
konsoll.add(stoppknapp, BorderLayout.SOUTH);
```

GUI 3: Sett opp rutenettet:

```
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));

        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);
            }
        }
        b.addActionListener(new Spillvelger(rx,kx));
        rutenett.add(b);
    }
}
panel.add(rutenett, BorderLayout.CENTER);
```

GUI 4: . . . og det var den initieringen:

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

GUI 5: Diverse operasjoner:

```
void markerTrekk (int r, int k, char c) {
    ruter[r][k].setText(Character.toString(c));
}

void visStatus (String tekst) {
    status.setText(tekst);
}
```

Programmet: Hovedprogrammet

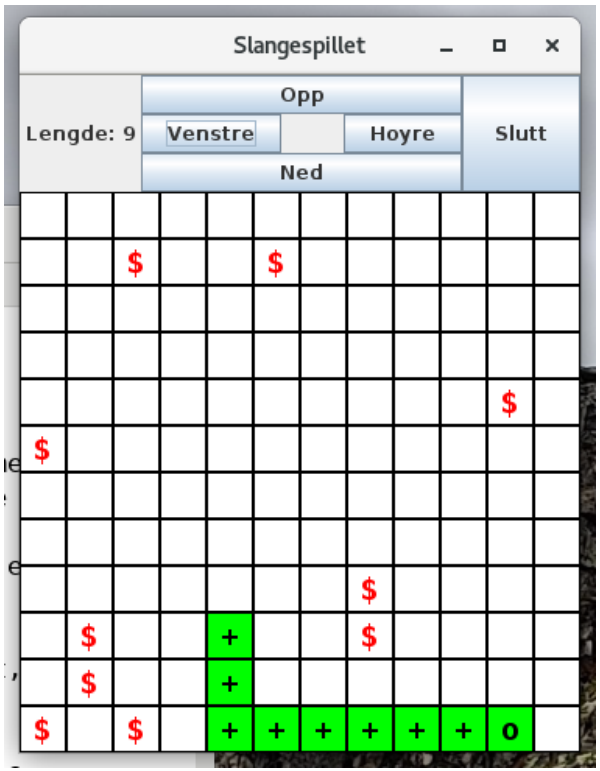
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

// Hovedprogrammet

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

Obligatorisk oppgave 7: Slangespillet

Nå er dere klare til å programmere *Slangespillet*, et enkelt dataspill.



Hvilke klasser og metoder trenger vi?

Disse klassene og metodene trenger man i IN1010. (Se også Big Java D.)

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