GUI («Graphical User Interface») del 1

- Om GUI
- AWT og Swing
  - Hvordan lage et GUI-vindu
  - Hvordan kommunisere med brukeren
  - Hvordan håndtere knappetrykk
- Hendelsesorientert programmering
  - Hva skjer når hendelser oppstår?

Se også på

- Big Java kapittel 10–11
- Programkoden til eksemplene ligger i https://www.uio.no/studier/emner/matnat/ifi/IN1010/v25/programmer/GUI/

Tripp-trapp-tresko med og uten GUI

```
$ java TrippTrappTreskoEnkel
Maskinen valgte rute 9
Ditt trekk? 5
Maskinen valgte rute 4
Ditt trekk? 1
Maskinen valgte rute 3
Ditt trekk? 2
Maskinen valgte rute 8
Ditt trekk? 6
Maskinen valgte rute 7
Maskinen vant!
```

```
$ java TrippTrappTresko
Maskinen valgte rute 1
+---+---+---+
| X |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+

      :

+---+---+---+
| X |   | O |
+---+---+---+
|   | O | X |
+---+---+---+
|   | X |   |
+---+---+---+

Ditt trekk? 7
Du vant!
+---+---+---+
| X |   | O |
+---+---+---+
|   | O | X |
+---+---+---+
| O | X |   |
+---+---+---+
```



Fordeler med GUI

- Mer intuitivt å bruke
- Færre muligheter for brukerfeil
- Visuelt mer tiltalende

Ulemper

- Mer komplisert å programmere
- Mange ulike GUI-biblioteker å velge blant
- Svært få biblioteker fungerer for både Linux, Mac og Windows.

Historien om Qt

- Opprinnelig laget av det norske firmaet *Trolltech* ved Sannerbrua i Oslo i 1994.
- Hovedideen var at programmer skrevet i C++ kunne linkes med et Qt-bibliotek og så kjøre på ulike systemer.
- Sannsynligvis det mest brukte *generelle* vindusbiblioteket i verden i 1990- og 2000-årene.
- Solgt til Nokia i 2008 (og siden til Digia og The Qt Company).
- Brukes i dag i Google Earth, Walt Disney animation studios, Tesla-biler, ...



Hvorfor GUI i IN1010?

- Allmennkunnskap for programmerere
- Et godt eksempel på bruk av OO
- Viser en ny programmeringstankegang: *Hendelsesorientert programmering*
- Et godt eksempel på bruk av parallellisering og tråder
- Demonstrerer programmeringsmønsteret MVC («model-view-controller»)

Java og GUI

Java har alltid hatt som mål at programmene skal kunne kjøres uendret på alle platformer.

AWT («Abstract Window Toolkit») fra 1995:
et ganske enkelt grensesnitt mot OS-ets vindussystem

Swing fra 2007:
et mer avansert og generelt system for vinduer, trykknapper etc bygget oppå AWT

JavaFX fra 2012:
et enda mer komplett og velstrukturert system

Fra 2018: JavaFX ikke lenger en del av standard Java men et frittstående produkt.

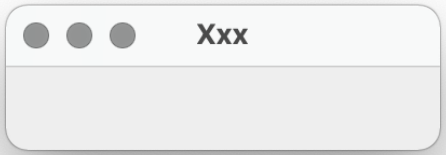
Demo 1: Et minimalt GUI-program

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

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
        JFrame vindu = new JFrame("Xxx");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

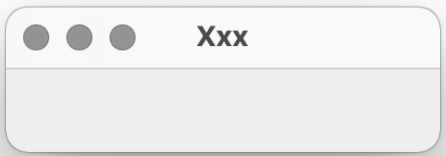
        vindu.pack();
        vindu.setLocationRelativeTo(null);
        vindu.setVisible(true);
    }
```



Importere klasser

Disse klassene dekker vårt behov i IN1010:

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



Et meget godt råd

Standard i Swing er å lage grafikk som ligner på OSets, men vi *bør absolutt* angi standard Swing-utseende:

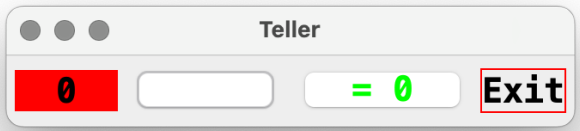
```
try {
    UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName());
} catch (Exception e) { System.exit(1); }
```

«Look and feel» i Swing

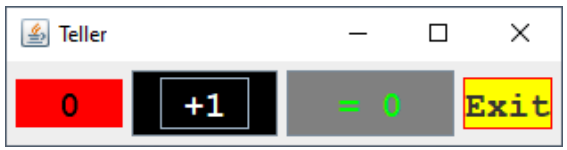
Normalt vil Swing prøve å lage GUI-utseende som harmonerer med andre vinduer på maskinen:



Linux



Mac



Windows

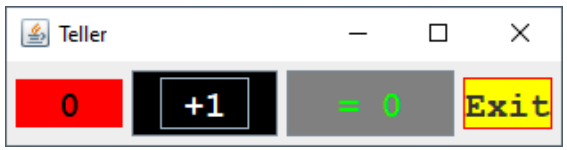
Noen ganger får vi ikke det vi ber om. Da er det bedre å gi alle vinduene et standard Swing-utseende:



Linux



Mac

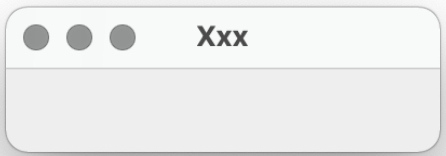


Windows

Deklarere vinduet

Nå kan vi deklarere vinduet vårt; det er en **`JFrame`**:

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



Initier stopp-knappen

Vi *må* angi at programmet vårt skal stoppe når vinduet lukkes:

```
vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Opprett tegneflaten

Nå kan vi opprette tegneflaten vår (et **JPanel**) og sette den inn i vinduet.

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

Gjør vinduet synlig

Til sist må vi huske på tre ting:

1. Pakk alt innholdet i vinduet vårt pent sammen:

```
vindu.pack();
```

2. Sørg for at vinduet plasseres midt på skjermen:

```
vindu.setLocationRelativeTo(null);
```

NB!

Dette må gjøres *etter* pakkingen!

3. Gjør vinduet med alt innhold synlig:

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

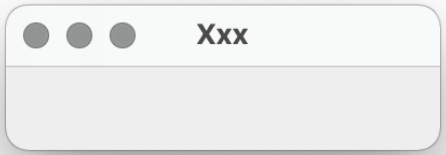
Resultatet

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

class Mini {
    public static void main (String[] args) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
        JFrame vindu = new JFrame("Xxx");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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



Kjøring

Grafiske programmer trenger mer støtte ved kjøring fra systemet enn våre tidligere tekstorienterte programmer.

Kjøring på privat maskin

Det enkleste er å installere Java på egen maskin (se nettsiden). Når man installerer vanlig JDK, får man alltid med AWT+Swing.

Kjøring på en UiO-maskin

Det er også mulig å kjøre GUI-programmene på en UiO-maskin.

- Fra Windows: Åpne **view.uio.no** i en nettleser; velg

  Ifi Workstation for å få en Linux-maskin eller
  MatNat Programmering for en Windows-maskin.
- Fra Mac: Installer **XQuartz** fra **//www.xquartz.org/** eller bruk **view.uio.no**.
- Fra Linux: Gi kommandoen **`ssh -Y login.ifi.uio.no`**

Demo 2: Bokser og slikt

På tegneflaten (`JPanel`-objektet) kan vi plassere ulike «bokser»:

- tekst (`JLabel`)
- trykknapper (`JButton`)
- redigerbart tekstfelt (`JTextField` og `JTextPane`)
- tegneflater (`JPanel`)
- filvelger (`JFileChooser`)
- ...

La oss lage et program som ønsker oss velkommen:

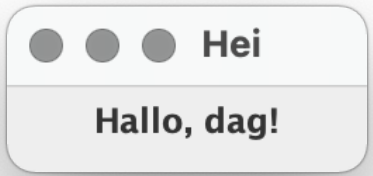
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Hallo {
    public static void main (String[] arg) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
        JFrame vindu = new JFrame("Hei");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

        String bruker = System.getProperty("user.name");
        JLabel hilsen = new JLabel("Hallo, " +  bruker + "!");
        panel.add(hilsen);

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



Det nye her er:

1. Vi ber Java om brukernavnet til den som kjører programmet:

```
String bruker = System.getProperty("user.name");
```

2. Vi lager en `JLabel` med hilsenen og legger den på vårt `JPanel`:

```
JLabel hilsen = new JLabel("Hallo, " +  bruker + "!");
panel.add(hilsen);
```

Struktur

Vi har altså en

- en `JFrame` som inneholder
  - et `JPanel` som inneholder
    - en `JLabel`.

Slik bygges et GUI-vindu opp.

Trykknapper

Den vanligste formen for interaksjon med en bruker er **trykknapper**. Når man skal ha en slik, må man:

1. opprette trykknappen (en **`JButton`**)
2. sette den på en tegneflate (en **`JPanel`**)
3. skrive kode som skal utføres ved et trykk (en **`ActionListener`**)
4. koble koden til trykknappen (med **`addActionListener`**)

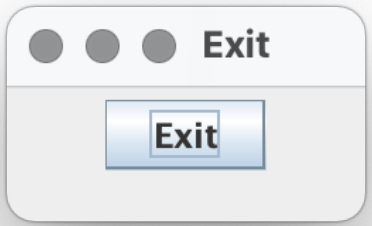
Et program som stopper seg selv

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

class Exit {
    public static void main (String[] arg) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
        JFrame vindu = new JFrame("Exit");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        vindu.add(panel);

        JButton exitKnapp = new JButton("Exit");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0);
            }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

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



1: opprette knappen

En trykknapp er en `JButton` med angitt tekst:

```
JButton exitKnapp = new JButton("Exit");
```

2: plassere på tegneflaten

Knappen må settes på en flate:

```
panel.add(exitKnapp);
```

3: skrive kode

Koden skrives ved å lage en klasse som implementerer **`ActionListener`** og definerer metoden **`actionPerformed`**:

```
class Stopper implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent e) {
        System.exit(0);
    }
}
```

4: koble knapp og kode

Til sist kan vi koble koden til knappen:

```
exitKnapp.addActionListener(new Stopper());
```



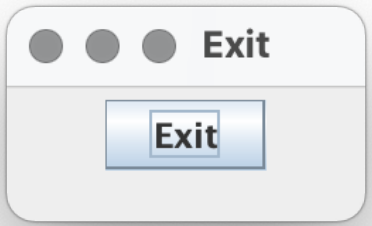
Et program som stopper seg selv

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

class Exit {
    public static void main (String[] arg) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }
        JFrame vindu = new JFrame("Exit");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        vindu.add(panel);

        JButton exitKnapp = new JButton("Exit");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.exit(0);
            }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

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



Programmeringsparadigmer

Det finnes mange **programmeringsparadigmer**,[1] for eksempel

- **Imperativ programmering** der utførelsen følger programflyten angitt av *programmereren*.
  - **Objektorientert programmering** er en undergruppe der operasjonene er knyttet til objekter.
  - I **funksjonell programmering** kan funksjoner (dvs metoder) betraktes som verdier.
- **Hendelsesdrevet programmering** («event-driven programming») der *brukerens handlinger* styrer programflyten.

[1] Et *programmeringsparadigme* er en måte å tenke på når vi programmerer.

Hendelsesdrevet programmering

Etter initieringen ligger programmet passivt og venter på at noe skal skje. Dette *noe* kan være

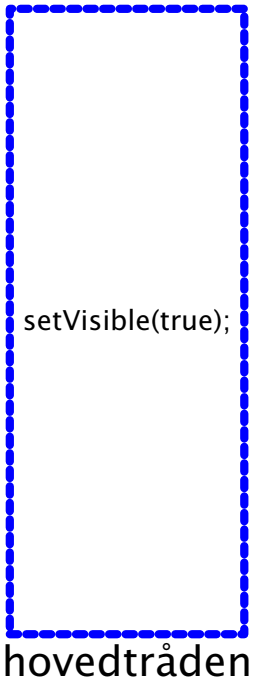
- brukeren klikker på en knapp på skjermen
- brukeren flytter musen
- brukeren trykker på en mustast
- brukeren trykker på en tast på tastaturet
- brukeren slipper opp en tast
- brukeren endrer størrelsen på vinduet
- et vindu kommer til syne fordi vinduet over fjernes

. . . og mye annet.

Oppstart av Swing

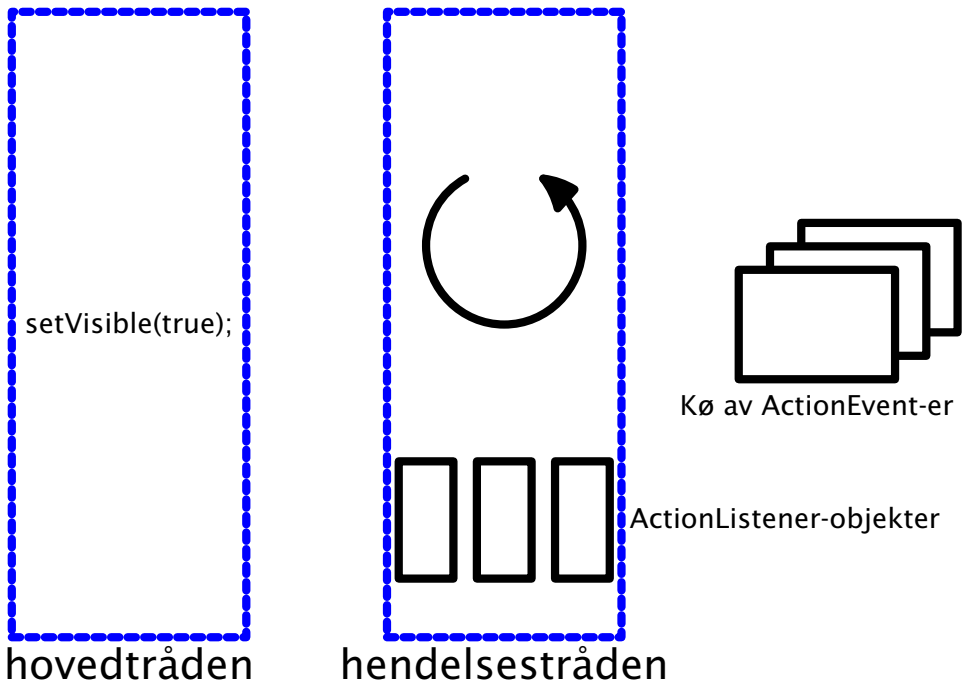
Programmet startes i metoden `main` i hovedtråden.

Etter en stund kaller programmet vårt `vindu.setVisible(true)`.



Vente på hendelser

Det opprettes en ny tråd: **hendelsestråden** («event dispatch thread») med en kø av hendelser. Så lenge køen er tom, ligger tråden passiv og venter på at noe skal skje.

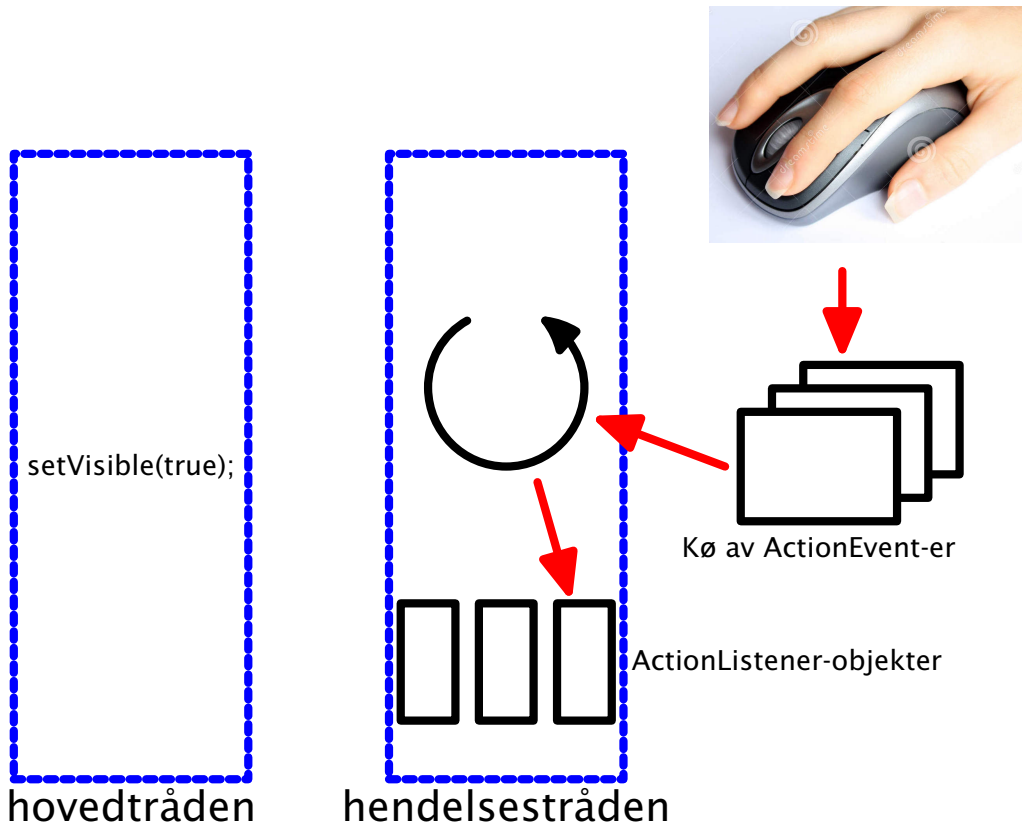


Håndtere hendelser

Hver hendelse som inntreffer resulterer i et `ActionEvent`-objekt i køen.

Hendelsesløkken tar `ActionEvent`-ene etter tur, og den korrekte `ActionListener` vil bli kalt.

Når en `ActionListener` er ferdig, vil hendelsesløkken ta neste `ActionEvent` (om det er noen).



Hvorfor trenger vi en kø av `ActionEvent`-er?

Hendelsestråden kan bare ta seg av én hendelse av gangen, men noen ganger kan flere hendelser inntreffe omtrent samtidig.

Da trenger vi køen for å ta vare på de hendelsene som venter på å bli tatt hånd om.

- Ingen hendelser må bli glemt.
- Hendelsene må håndteres i riktig rekkefølge.

Tråder

Når vi er kommet i gang, har vi altså to tråder. Dette innebærer:

- De to trådene går uavhengig av hverandre.
- Hovedtråden kan fortsette med sine egne ting uten å forstyrre eller bli forstyrret av hendelsestråden.
- Selv om den ene tråden dør, lever den andre videre.

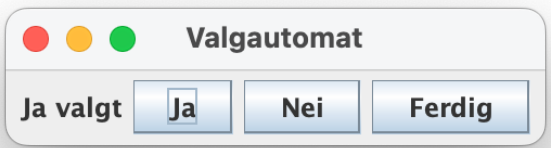
Viktig!

Det er derfor vi trenger `vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`.

- `System.exit(n)` dreper alle trådene.
- Kun én `ActionListener` kan jobbe av gangen; hvis den bruker lang tid, virker hendelseshåndteringen død.

En enkel valgautomat

Vi skal nå lage en meget enkel valgautomat der brukeren kan velge «Ja» eller «Nei».



Etterpå skriver programmet ut hva brukeren har valgt:

```
$ javac Valgautomat.java
$ java Valgautomat
Ditt valg var: Ja
```

I tillegg til det vanlige oppsettet trenger vi fire bokser:

1. En tekst med hva vi har valgt:

```
JLabel status = new JLabel("Vet ikke");
panel.add(status);
```

2. En knapp for å velge **Ja**:

```
JButton jaKnapp = new JButton("Ja");
class Javelger implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent ae) {
        valg = "Ja";  status.setText("Ja valgt");
    }
}
jaKnapp.addActionListener(new Javelger());
panel.add(jaKnapp);
```

Valget lagres i en egen variabel:

```
static String valg = "Vet ikke";
```

3. Knappen for å velge **Nei** er nesten lik:

```
JButton neiKnapp = new JButton("Nei");
class Neivelger implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent ae) {
        valg = "Nei";  status.setText("Nei valgt");
    }
}
neiKnapp.addActionListener(new Neivelger());
panel.add(neiKnapp);
```

4. Stoppknappen skal skrive ut svaret og avslutte programmet:

```
JButton exitKnapp = new JButton("Ferdig");
class Stopper implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent ae) {
        System.out.println("Ditt valg var: " + valg);
        System.exit(0);
    }
}
exitKnapp.addActionListener(new Stopper());
panel.add(exitKnapp);
```



Det ferdige programmet

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

class Valgautomat {
    static String valg = "Vet ikke";

    public static void main (String[] arg) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }

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

        JLabel status = new JLabel("Vet ikke");
        panel.add(status);

        JButton jaKnapp = new JButton("Ja");
        class Javelger implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent ae) {
                valg = "Ja";  status.setText("Ja valgt");
            }
        }
        jaKnapp.addActionListener(new Javelger());
        panel.add(jaKnapp);
```



Det ferdige programmet

```
        JButton neiKnapp = new JButton("Nei");
        class Neivelger implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent ae) {
                valg = "Nei";  status.setText("Nei valgt");
            }
        }
        neiKnapp.addActionListener(new Neivelger());
        panel.add(neiKnapp);

        JButton exitKnapp = new JButton("Ferdig");
        class Stopper implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent ae) {
                System.out.println("Ditt valg var: " + valg);
                System.exit(0);
            }
        }
        exitKnapp.addActionListener(new Stopper());
        panel.add(exitKnapp);

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

Treg hendelseshåndtering

For å demonstrere effekten av treg håndtering av brukerinteraksjon, kan vi legge inn litt søvn når brukeren trykker «Ja». Da vil vi oppleve at programmet vårt «henger» i 10 sekunder hver gang.

```
JButton jaKnapp = new JButton("Ja");
class Javelger implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent ae) {
        try {
            Thread.sleep(10_000);
        } catch (InterruptedException e) {}

        valg = "Ja";  status.setText("Ja valgt");
    }
}
jaKnapp.addActionListener(new Javelger());
panel.add(jaKnapp);
```

Oppsummering

- Hva er GUI, og hvorfor
- Oppbygging av et GUI-vindo
- Brukerinteraksjon i et GUI-vindu
- Hendelsesorientert programmering
- Parallellisering i GUI

. . . og neste gang fortsetter Eyvind.