GUI – grafiske brukergrensenitt (del 1)

IN1010, 21. april 2026

Magnus Skjegstad
(basert på materiale fra Dag Langmyhr)

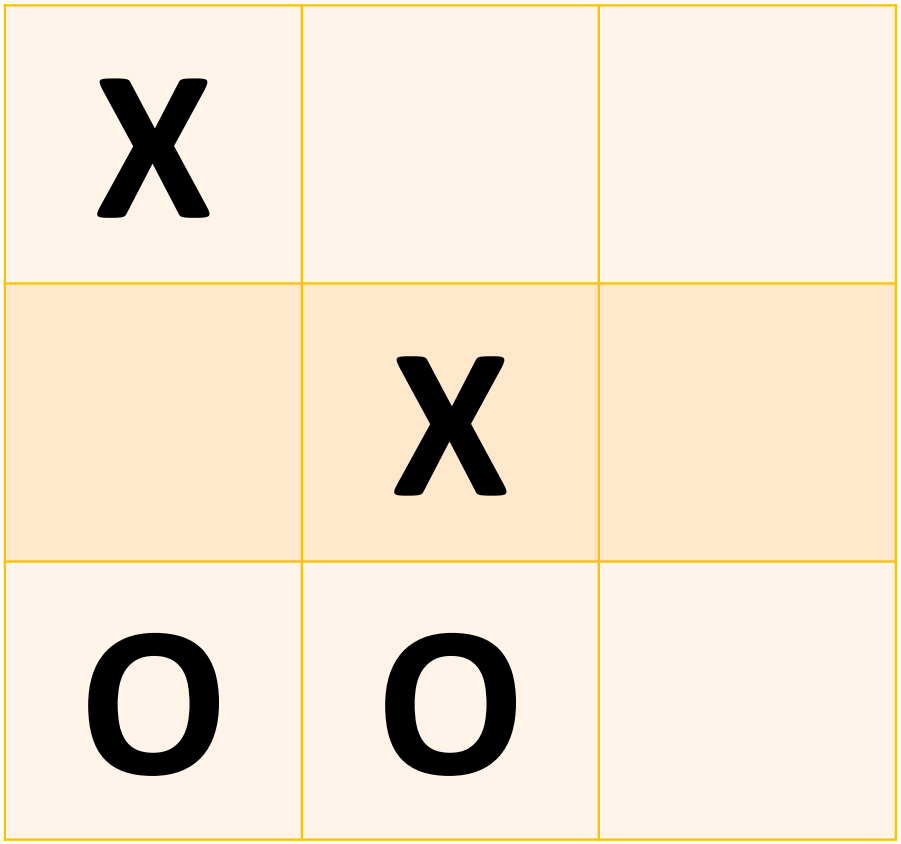
Horstmann kpt 10-11

Planen i dag

- Først litt om GUI
- Rammeverkene vi skal bruke: AWT og Swing
  - Hvordan lager vi et vindu som dukker opp på skjermen?
  - Hvordan kan vi kommunisere med brukeren?
  - Hvordan kan vi ta i mot knappetrykk?
- Hendelsesorientert programmering



Tripp-trapp-tresko



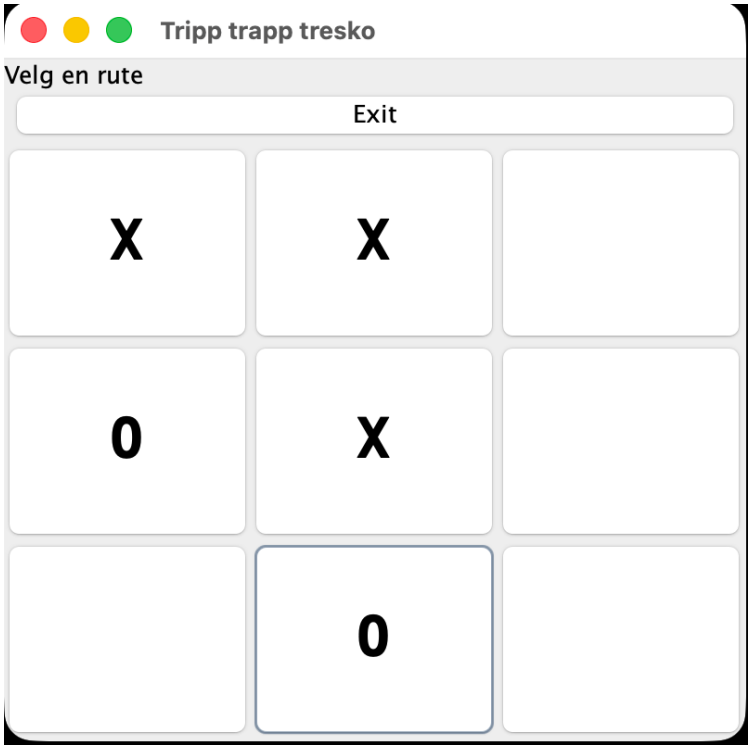
Tripp-trapp-tresko med og uten GUI

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

```
$ java TrippTrappTresko
Maskinen valgte rute 8
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   | X |   |
+---+---+---+

Ditt trekk? 1
Maskinen valgte rute 5
+---+---+---+
| O |   |   |
+---+---+---+
|   | X |   |
+---+---+---+
|   | X |   |
+---+---+---+

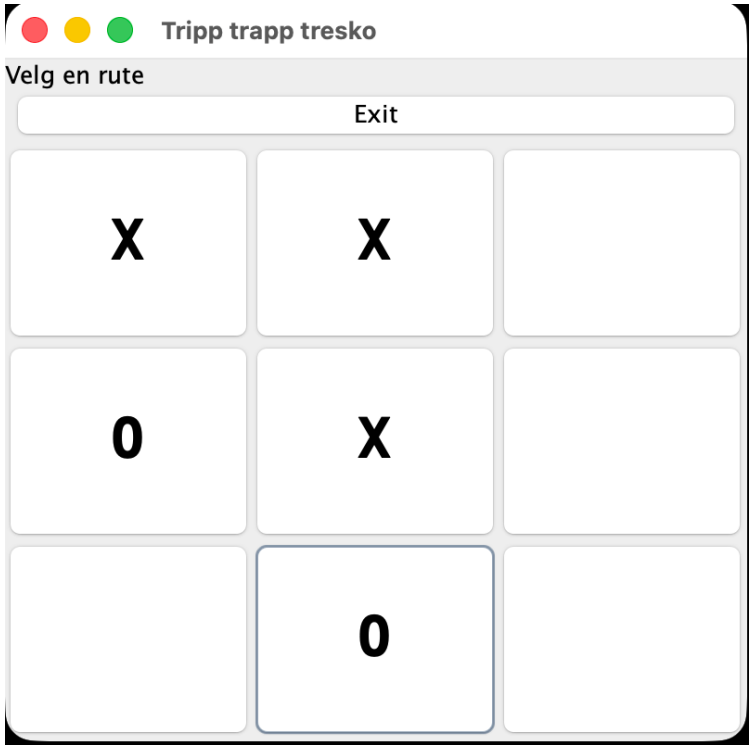
Ditt trekk? 2
```



Fordeler med GUI

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

- Hvorfor har vi ikke brukt det før?
  - Mer komplisert å programmere
  - Mange ulike GUI-biblioteker å velge blant, med ulike styrker og svakheter
  - Svært få biblioteker støtter 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 Digia og The Qt Company)
- Brukes i dag i Google Earth, Walt Disney animation studios, Tesla-biler, LG-tver (webOS), Roblox Studio, Telegram, Virtualbox, OBS...osv osv (oversikt finnes på Wikipedia)

Hvorfor GUI i IN1010?

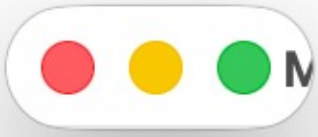
- Allmennkunnskap for programmerere – man havner ofte i situasjoner der man lager et program som trenger et lite GUI
- Et godt eksempel på bruk av objektorientering
- Viser en ny programmeringstankegang: *Hendelsesorientert programmering*
- Et godt eksempel på bruk av parallellisering og tråder
- Demonstrerer programmeringsmønsteret MVC («model-view-controller»)



Java og GUI

- Java har alltid hatt som mål at programmene skal kunne kjøres uendret på alle plattformer
- **AWT** («Abstract Window Toolkit») fra 1995:
  - Et ganske enkelt grensesnitt mot operativsystemets vindussystem
- **Swing** fra 2007:
  - Et mer avansert og generelt system for vinduer, trykknapper osv, bygget på toppen av AWT
- **JavaFX** fra 2012:
  - Et mer komplett og velstrukturert system. Fra 2018 er JavaFX ikke lenger en del av standard Java, men et frittstående produkt

Et minimalt GUI



```java
import javax.swing.*;

class Mini {
    public static void main (String[] args) {
        JFrame vindu = new JFrame("Minimalt vindu - IN1010");
        vindu.setVisible(true);
    }
}
```

```
$ javac Mini.java
$ java Mini
^C%
$
```

Et (nesten) minimalt GUI

- Vi skal lage et tomt vindu på skjermen som avslutter programmet når vi lukker det.

```
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("Nesten minimalt vindu - IN1010");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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

Samme oppsett hver gang

- Klasser som dekker det vi trenger i IN1010

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

- Ikke nødvendig, men sterkt anbefalt: angi utseendet til Swing

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

Swing «look and feel»

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



Linux



macOS



Windows

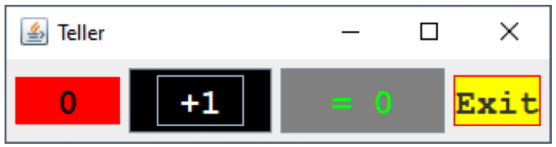
Noen ganger får vi ikke det vi ber om. Da er det bedre å gi alle vinduene et standard Swing-utseende («cross platform look and feel»):



Linux



macOS



Windows

Nå kan vi deklarere vinduet

```
JFrame vindu = new JFrame(”Nesten minimalt Vindu - IN1010");
```

Vi må også angi at trådene til Swing skal avslutte når vinduet lukkes, ellers stopper ikke programmet vårt:

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

...og opprette tegneflaten

Vi lager en tegneflate (JPanel) og setter den inn i vinduet:

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

Til slutt må vi huske på tre ting:

- Pakk alt innholdet i vinduet pent sammen:

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

- Sørg for at vinduet havner midt på skjermen:

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

- Gjør vinduet og alt innhold synlig (NB: Må skje *etter* pakking):

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

Et (nesten) minimalt GUI

- DEMO

```java
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("Nesten minimalt vindu - IN1010");
        vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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

Hva annet kan vi putte i vinduet?

- På tegneflaten (**JPanel**) kan vi plassere ulike «bokser» eller komponenter:
  - Tekst (**JLabel**)
  - Trykknapper (**JButton**)
  - Redigerbart tekstfelt (**JTextField** og **JTextPane**)
  - Tegneflater (**JPanel**)
  - Filvelger (**JFileChooser**)
  - ...

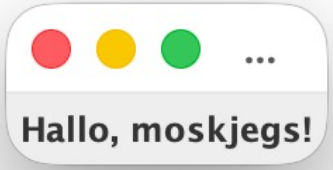
Vi lager et program som ønsker oss velkommen

```java
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

- Vi ber Java om brukernavnet til den som kjører programmet:

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

- Vi lager en **JLabel** med hilsenen og legger den på vårt **JPanel**:

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

- Vi har altså:
  - en **JFrame** som inneholder...
    - en **JPanel** som inneholder...
      - en **JLabel**

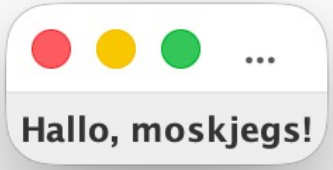
Vi lager et program som ønsker oss velkommen

```java
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);
    }
}
```



Trykknapper

- Den vanligste formen for interaksjon med en bruker er **trykknapper**.
- Når man skal lage en trykknapp må man:
  - Opprette knappen (en **JButton**)
  - Sette den på en tegneflate (en **JPanel**)
  - Skrive kode som skal utføres ved et trykk (en **ActionListener**)
  - Koble koden til knappen (med **addActionListener**)

Et program som kan avslutte

```
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);
    }
}
```



Det nye her

- Vi må opprette knappen (JButton) og legge inn tekst:

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

- Vi må plassere knappen på tegneflaten:

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

- Så må vi skrive koden for å trykke på knappen (neste slide)

Kode for trykknapp

- 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);
  }
}
```

- Til sist kan vi koble knapp og kode:

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

Et program som kan avslutte

```
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);
    }
}
```



Mer kode, men de fleste oppgavene man må utføre er ganske like – og greie når man blir vant til dem.

Programmeringsparadigmer

- Det finnes mange programmeringsparadigmer, for eksempel
  - **Imperativ programmering** der utførelsen følger programflyten angitt av programmereren.
    - **Objektorientert programmering** er en undergruppe der operasjonene er knyttet til objekter
  - **Hendelsesdrevet programmering** («event-driven programming») der brukerens handlinger styrer programflyten

Et programmeringsparadigme er en måte vi tenker 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 museknapp
  - Brukeren trykker på en knapp 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 starter i metoden *main* i hovedtråden.

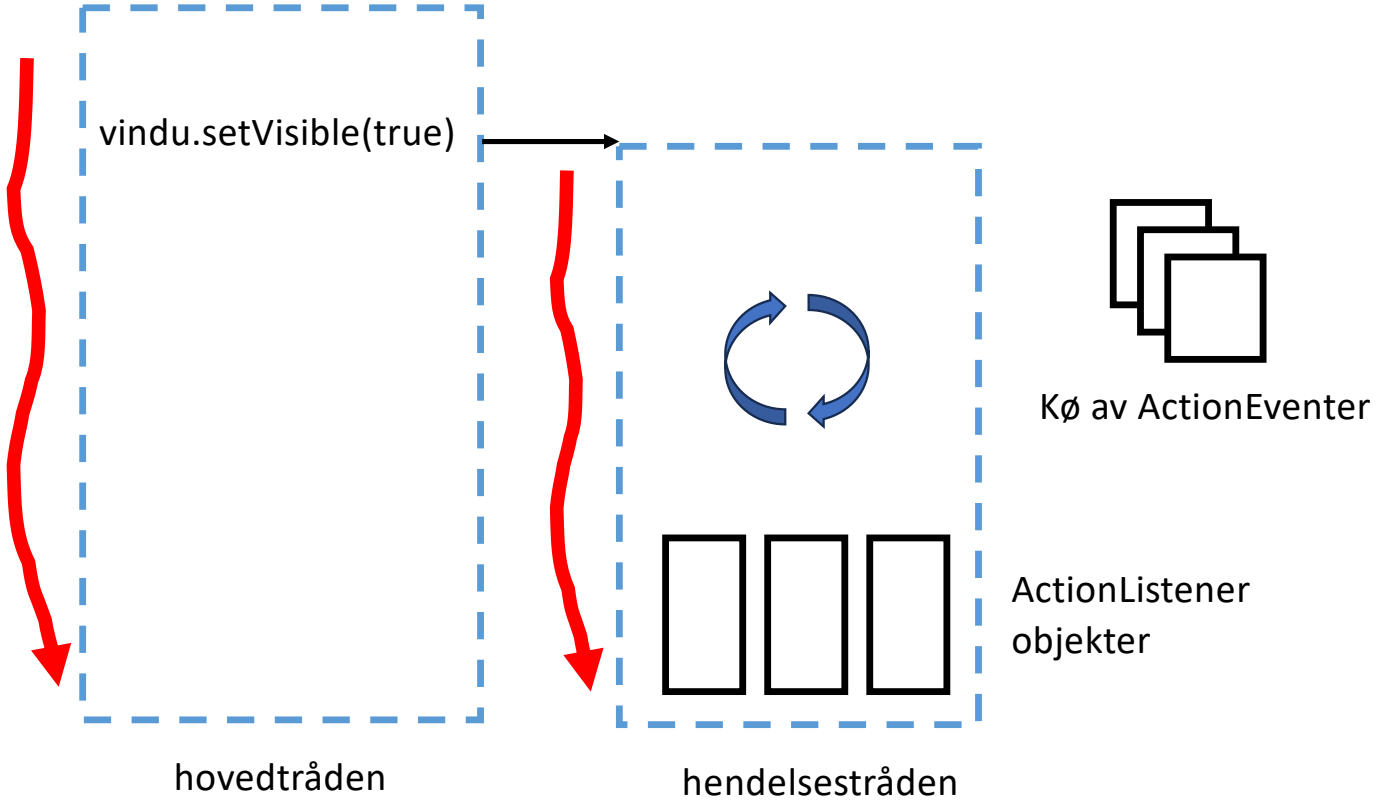
Etter en stund kaller programmet vårt
`vindu.setVisible(true)`



Oppstart av Swing

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.

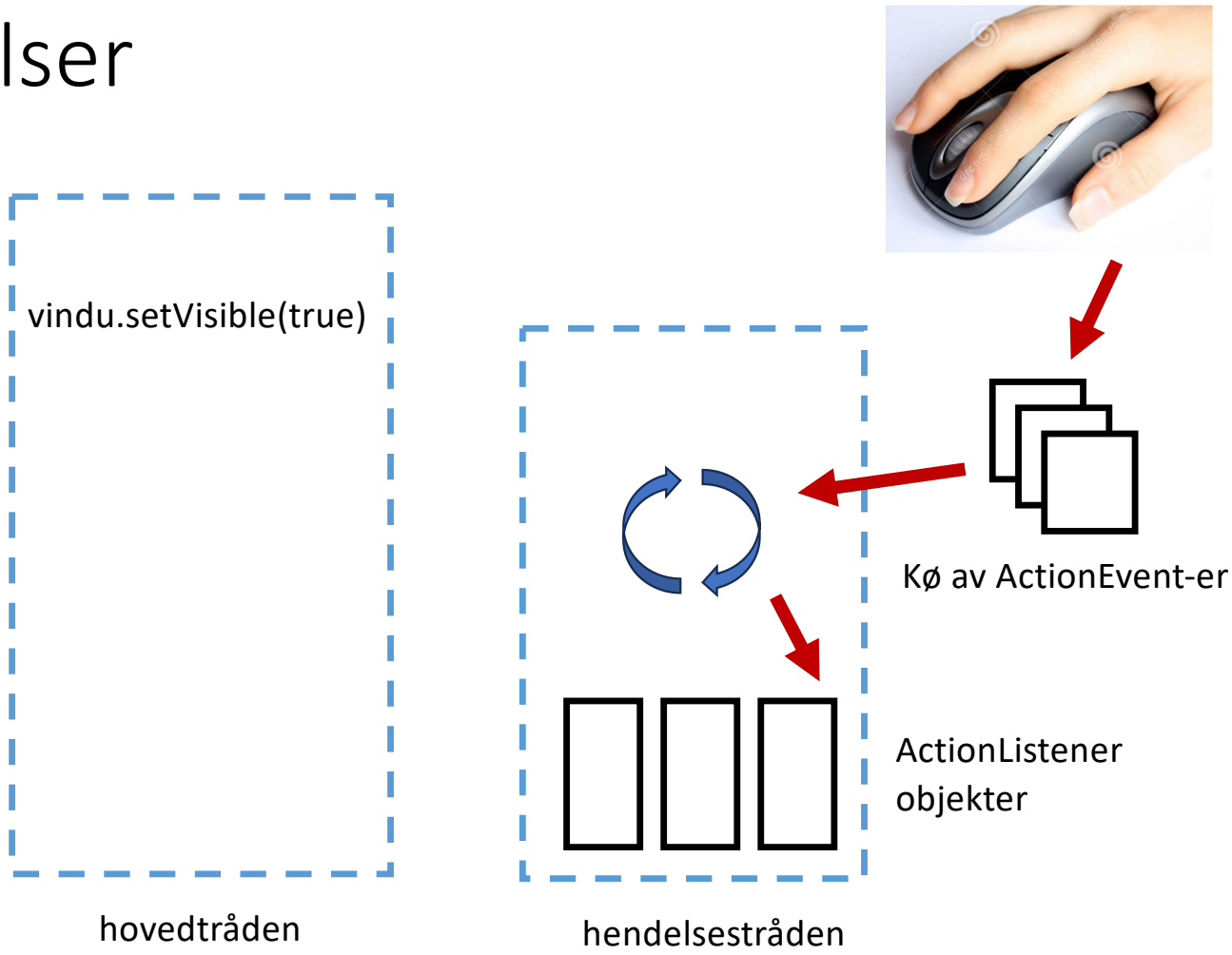


Håndtere hendelser

Hver hendelse som inntreffer legger et ActionEvent-objekt i køen.

Hendelsesløkken tar ActionEvent-er etter tur, og den korrekte ActionListener blir 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 har kommet i gang har vi 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. Derfor trenger vi:

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

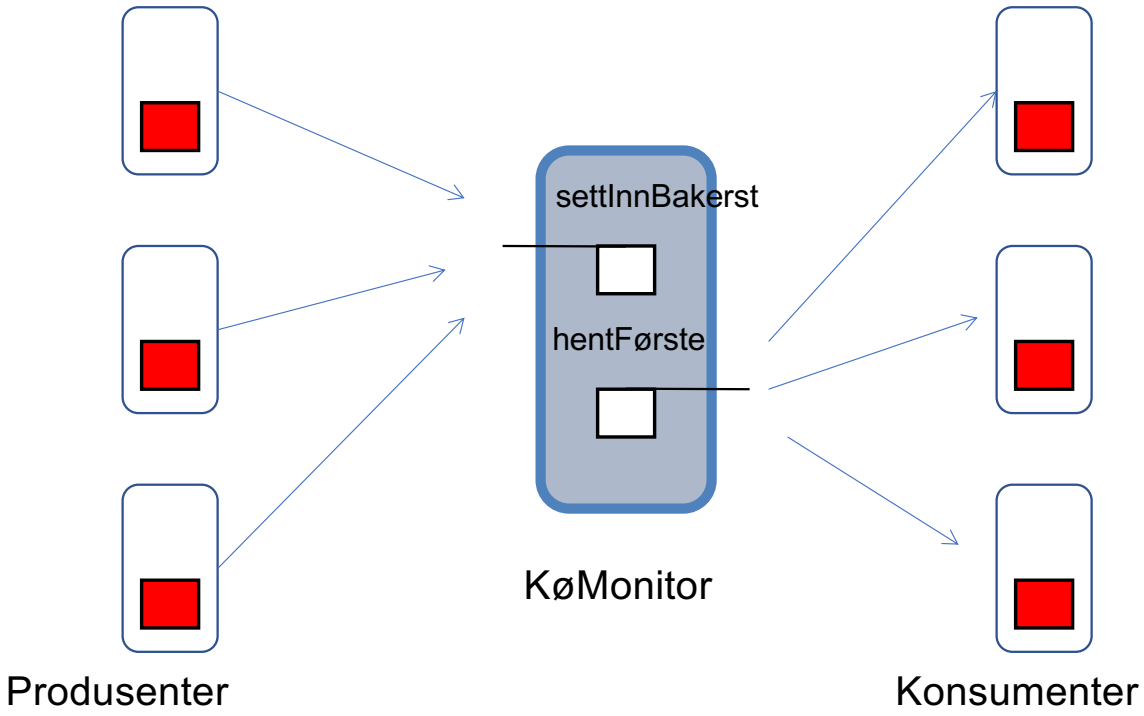
- `System.exit(`*n*`)` dreper alle trådene.
- Kun én ActionListener kan jobbe om gangen; hvis den bruker lang tid, virker GUIet tregt.

Litt mer om trådsikkerhet

- Swing er **ikke** trådsikkert – men ActionEvent-køen kan trygt ta imot hendelser fra alle andre tråder (som en **monitor**).
- Hendelsestråden henter et ActionEvent og kaller én ActionListener om gangen.
- **Bare hendelsetråden kan oppdatere GUI uten oppdateringsproblemer.**
- I IN1010 lager vi grensesnittet ferdig - så kaller vi vindu.setVisible(true) og gjør senere endringer fra ActionListener objekter.

Repetisjon fra tråd-forelesningen: «Det viktigste i IN1010 om tråder»

Kommunikasjon mellom tråder: En felles beholder = en monitor



ActionEvent-køen i Swing fungerer som en monitor, der andre tråder kan legge inn hendelser i en kø.

Sende hendelser fra hovedtråden

- *Dette er ikke nødvendig i IN1010, men kan være greit å vite om.*
- Vi kan sende hendelser fra hovedtråden til hendelsestråden etter at man har kalt setVisible(true). Da må vi bruke **SwingUtilities.invokeLater**() og **SwingUtilities.invokeAndWait()**.
- Disse legger et objekt bakerst i køen. Objektet må implementere Runnable.
- Oppgavene må være raske, ellers blokkerer de GUI.
- Nyttig hvis man skal sende GUI-oppdateringer fra en treg operasjon i en annen tråd.

En enkel valgautomat

- Vi skal nå lage en veldig 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: Nei
```

Vi trenger fire nye bokser

1. En tekst med hva vi har valgt: `JLabel status = new JLabel("Vet ikke")`
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);
```

Vi lagrer valget i en egen variabel:

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

Her endres status – det er greit, for Javelger kalles fra hendelsestråden.

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 stoppe 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);
```

```java
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);
```

```
        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

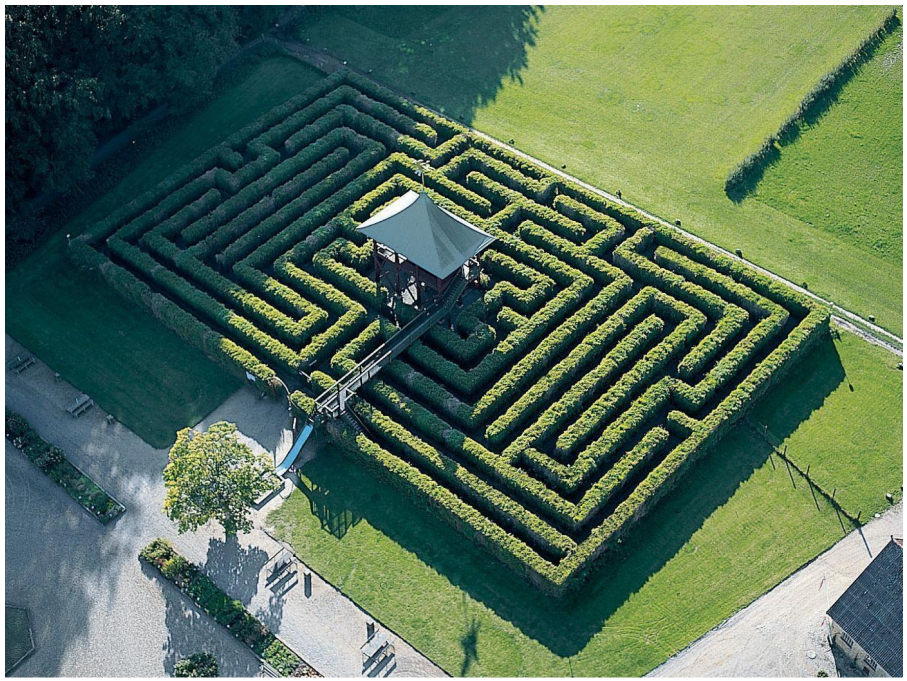
- For å demonstrere effekten av treg hånderting av brukerinteraksjon, kan vi legge inn litt søvn når brukeren trykker «Ja». Da vil vi oppleve at programmet vårt «henger» i 10 sekunder hver gang.

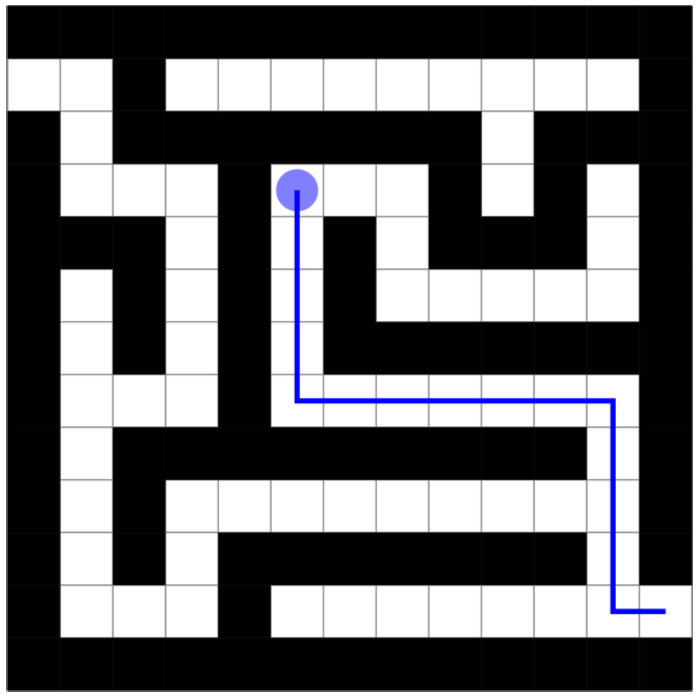
```
JButton jaKnapp = new JButton("Ja");
class Javelger implements ActionListener {
    @Override
    public void actionPerformed (ActionEvent ae) {
        try {
                Thread.sleep(10_000);
        } catch (InterruptedException e) {}

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

Innlevering 6

- Ligger ute nå, frist 4. mai
- Her skal man lage et program som finner veien gjennom en labyrint





Oppsummering

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

... neste gang ser vi mer på tripp-trapp-tresko og MVC

Alle programmer fra i dag er tilgjengelig på kurssiden:
https://www.uio.no/studier/emner/matnat/ifi/IN1010/v26/kode/