Graphical User Interfaces (Components, Layout and Event Handling)
Kapitel 13

Joakim Iversen

January 13, 2025

Disposition:

1. Motivation
2. Components
3. Layout
4. Event Handling

Noter

Main Concepts

- Constructiong GUIs
- interface Components
- GUI Leyout
- Event handling

Motivation

Tre hovedting at skulle vide

1. Hvilke elementer kan vi vise - Components
2. Hvordan arrangere vi disse elementer - Layout
3. Hvordan reagere vi på bruger input - Event Handling

Biblioteker til GUI

Der er tre biblioteker i Java til GUI. AWT - *Abstract Window Toolkit* - Der er den ældste. Swing, der bygger oven på AWT. Og den nyeste JavaFX.

Tegn "Kasserne" med hvordan de forskellige klasser ser ud

Jeg vil have fokus på Swing og være det jeg bruger i mine eksempler.

Som eksempel til at vise de forskellige ting jeg snakker om, vil jeg tage udgangspunkt i BlueJ's ImageViewer projekt, der er et program til at åbne billedfiler, lave noget billede tranformation og gemme billedet igen.

Lave en frame

Det første vi skal bruge er et *Top level Window* Når vi skal bruge vores GUI biblioteker til vi først importere dem til vores klasse. Derefter kan vi lave en simpelt klasse til at indeholde vores frame vi vil tilføje alle vores components til.

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

public class ImageViewer{
    private JFrame frame;

    public ImageViewer() {
        makeFrame();
    }

    private void makeFrame() {
        frame = new JFrame("ImageViewer");
        Contaioner contentPane = frame.getContentPane();

        JLabel label = new JLabel("I am a label. I can display some text.");
        contentPane.add(label);
```

```
        frame.pack();
        frame.setVisible(true);
    }
}
```

TEGN OUTPUTTET AF DETTE OG FORKLAR HVORDAN DE FORSKELLIGE TING HÆNGER SAMMEN

De sidste to linjer "pakker" først vores frame og arrangere tingene ordentligt. Det er altså noget vi altid skal kalde når vi har tilføjet eller ændret nogle af vores components.

Den sidste linje gør hele vores frame synlig.

Copmonents

Der er også mange andre components vi vil kunne tilføje til vores frame. For at nævne nogle af dem er der:

- Knapper
- Billede panel
- Menubar

I kan spørge ind til disse hvis i ønsker noget uddybelse.

Layout

Når vi så har tilføjet nogle ting opdager vi måske ret hurtigt at det ikke ser ud visuelt som vi regnede med, eller gerne vil have det til. For at fikse dette kan vi gøre brug af de forskellige layout metoder der er.

Swing gør brug af *layout managers* til at arrangere components i GUI'en. Hver container der indeholder elementer har en tilhørende layout manager, som så står for at arrangere de indeholdte components.

Swing giver adgang til mange forskellige layout managers men de vigtigste er

- FlowLayout
- BorderLayout
- GridLayout
- BoxLayout

Hoved forskellen mellem disse er måden hvorpå de placere forskellige elementer, og bearbejder den tilgængelige plads mellem elementerne.

FlowLayout

```
panel.setLaytout(new FlowLayout);
```

Arrangere det fra venstre til højre. Centreret horisontalt og give med den prefferet størrelse. Hvis der ikke er pldas på den igangværende linje, vil den lave linje skift.

Da elemetnerne ikke er sat til at fylde skærmen ud, vil der opstå ekstra plads hvis man resizer vinduet.

BorderLayout

```
panel.setLayout(new BorderLayout);
```

Tillader at gemme elementer i 5 forskellige placeringer

- Top - North
- Bund - South

- Venstre - West
- Højre - East
- Midten - Center

Ved at bruge disse ord kan du placere elementer i de tilsvarende placeringer.

Når man resizer et BorderLayout vindue er det midten der ændre sig mest. Højre og Venstre sidder bliver ændret i højden og top, bund i bredden.

GridLayout

```
panel.setLayout(new GridLayout(3,2));
```

Du får et "grid" men det givne antal rækker og søjler. Hvor man så fylder op fra venstre til højre.

BoxLayout

```
panel.setLayout(new BoxLayout);
```

Event Handling

Tre måder at håndetere dette på. Når man trykker på knappen bliver der rejst et **AcitonEvent**, dette vil man kunne lytte på hvis man implementere interfacet **ActionListener**.

Den har en metode som skal implementeres, hvilket er

```
public void actionPerformed(ActionEvent e) {
    if(e.getActionComment().equals("Open")) {
        open();
    }
    else if(... quit) {
        quit();
    }
}
```

For at vores open metode ved at der skal lyttes efter et tryk, skal vi huske at tilføje

```
open.addActionListener(this);
```

Dette er en grim måde at gøre det på, da vi kan have vildt mange items der går hen til den samme metode, hvor metoden så skal finde ud af hvor kaldet kom fra og så handle derefter.

Da der kune er en metode i interfacet, vil vi nemt kunne implementere det via en lambda. Vi vil derfor i stedet kunne skrive følgende:

```
open.addActionListener(e -> open());
```

Grunden til vi ikke behøver skrive *ActionEvent* i vores lambda er at compileren selv kan regne det ud.

Dette er meget pænere. men det kan ske at et listener interface ikke er funktionelt og har flere metoder, og så ville dette ikke være en mulig løsning. Der vil kunne gøre følgende, ved hjælp af en anonym indre klasse:

```
open.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e){
        open();
    }
})
```

Her laver deklarer vi en ny indre klasse