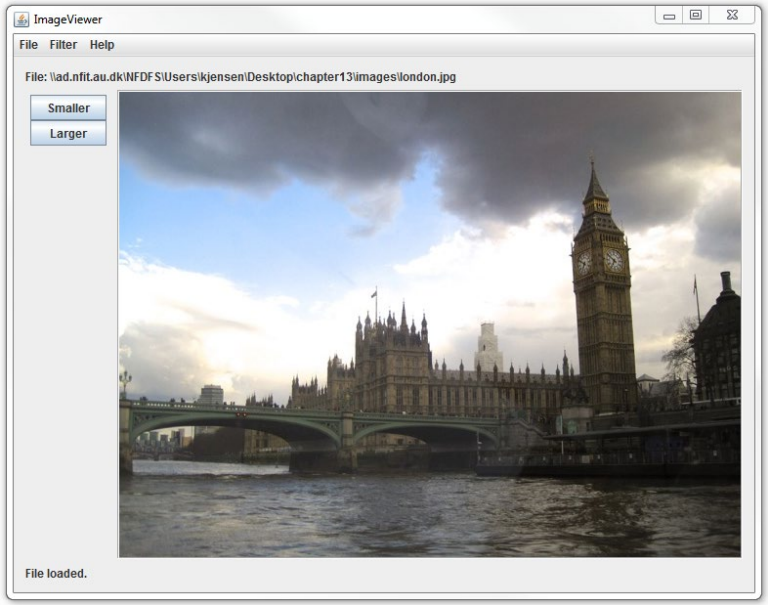
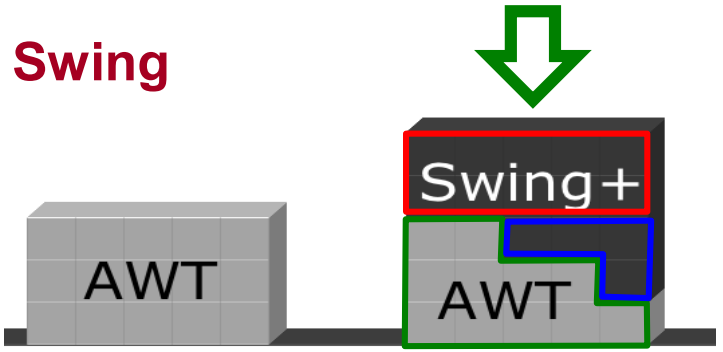
Forelæsning Uge 13

- **Konstruktion af grafiske brugergrænseflader (GUI'er)**
  - Definition af de elementer, der vises på skærmen (vinduer, knapper, menuer, scrollbarer, tekster, osv.)
  - Hvordan reagerer de på input (via mus og tastatur)?
  - Hvordan placeres de i forhold til hinanden (layout)?
- **Anonyme indre klasser**
  - Sprogkonstruktion, der bl.a. er nyttig i forbindelse med visse GUI events
- **Afleveringsopgave: Computerspil 3**
  - Brug af nedarvning og dynamic method lookup



AWT og Swing

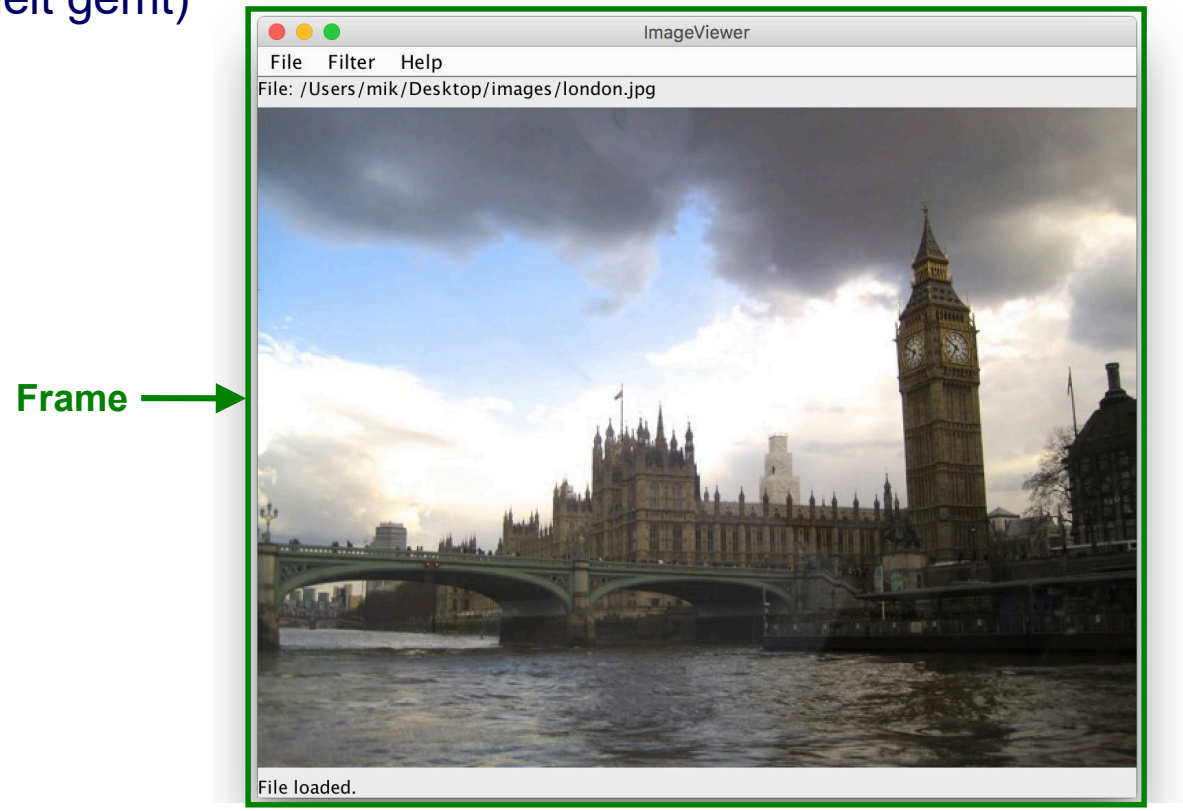
- **Java indeholder tre forskellige biblioteker til konstruktion af GUI'er**
  - Ældste (1995): **AWT** (Abstract Window Toolkit)
  - Mellemste (2008): **Swing** (langt bedre på mange punkter)
  - Nyeste (2015): **JavaFX**
- **Vi vil koncentrere os om brugen af Swing**
  - Mange Swing klasser er helt nye ■
  - Andre erstatter AWT klasser ■
  - Endelig bruger Swing også klasser fra AWT (uden at ændre dem) ■



- **Når der er ækvivalente klasser i AWT og Swing, tilføjer Swing et J foran navnet**
  - Button, Frame og Menu er klasser i AWT
  - JButton, JFrame og JMenu er klasser i Swing

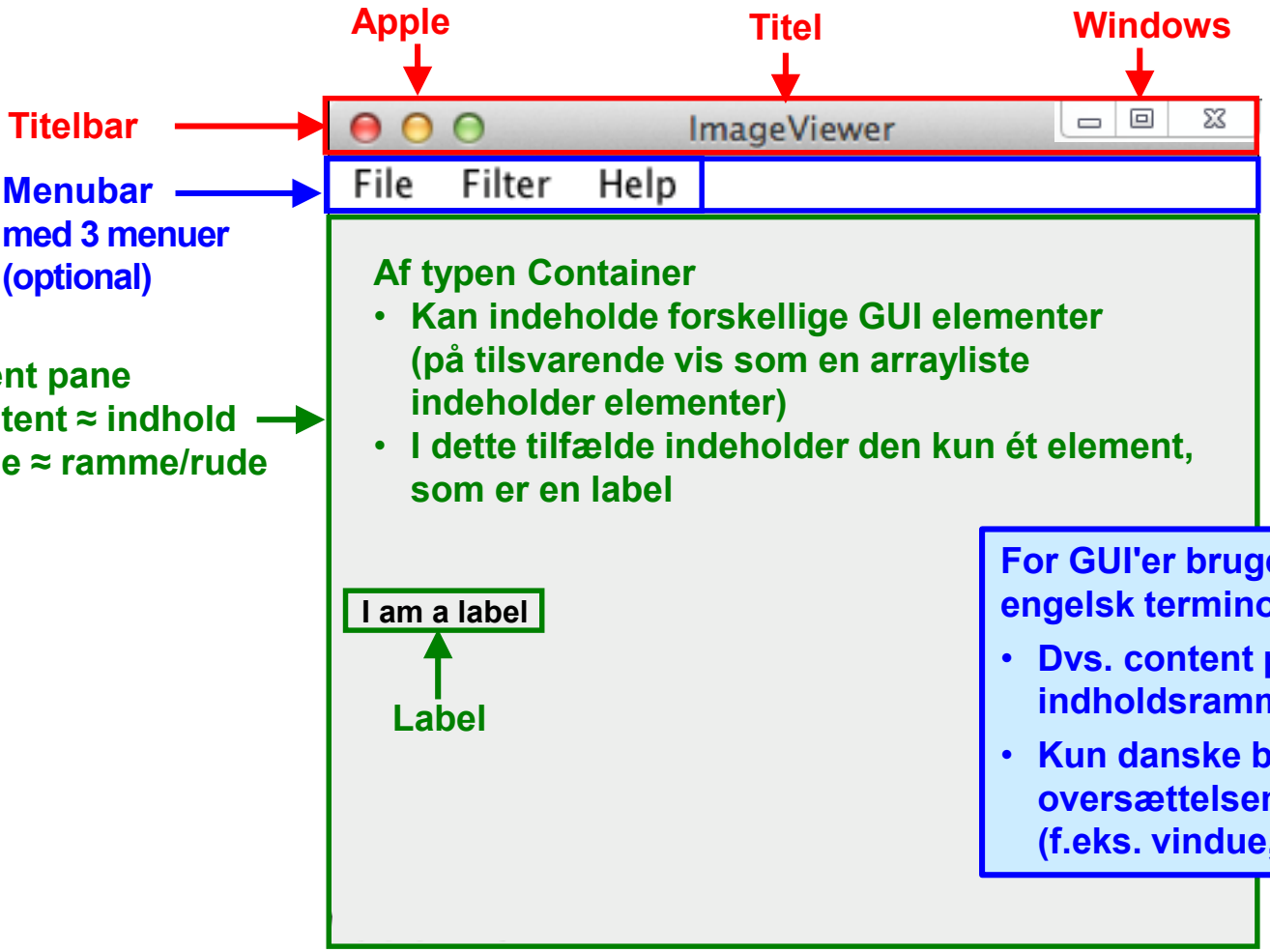
Vinduer (frames)

- **Lad os starte med at se, hvordan vi kan opbygge et vindue med nedenstående indhold**
  - Dette gøres ved hjælp af en **frame** (ramme)
  - Det er **operativsystemet**, der bestemmer, hvordan vinduet vises på skærmen (dvs. om den er øverst, delvist skjult af andre vinduer, eller helt gemt)



Terminologi for frames (vinduer)

Knapper til kontrol af vinduet (minimer, maksimer, luk)
- Udseendet af kontrolknapperne afhænger af operativsystemet



For GUI'er bruger jeg primært engelsk terminologi
- Dvs. content pane (i stedet for indholdsramme).
- Kun danske betegnelser, hvor oversættelsen er helt oplagt (f.eks. vindue, menu og knap)

Java kode for simpel ramme (frame)

Importer relevante pakker fra AWT og Swing (bemærk x'et)

Feltvariabel af type JFrame

Konstruktør

Initialisering af feltvariablen

Erklæring af lokal variabel af type Container

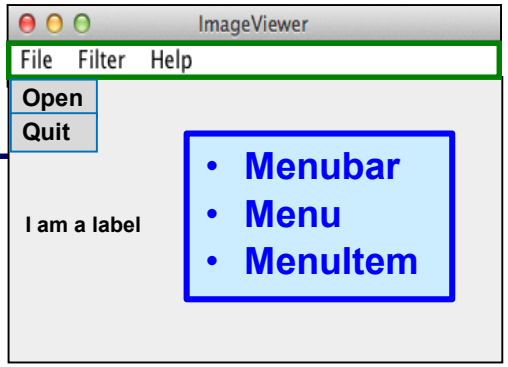
Skab en label og tilføj den til contentPane

Rammen pakkes (størrelser og layout) og gøres synlig

```
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");
      Container contentPane = frame.getContentPane();
      JLabel label = new JLabel("I am a label");
      contentPane.add(label);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Privat metode
- Indeholder al kode til konstruktion af rammen
- Eksempel på god "cohesion"

Menuer




```
private void makeFrame() {
   frame = new JFrame("ImageViewer");
   ...
   makeMenuBar(frame);
   ...
}
```

Privat metode
- Indeholder al kode til konstruktion af menubaren
- Har rammen (frame) som parameter

```
private void makeMenuBar(JFrame frame) {
   JMenuBar menubar = new JMenuBar();
   frame.setJMenuBar(menubar);

   // Create the File menu
   JMenu fileMenu = new JMenu("File");
   menubar.add(fileMenu);

   JMenuItem openItem = new JMenuItem("Open");
   fileMenu.add(openItem);

   ...

   JMenuItem quitItem = new JMenuItem("Quit");
   fileMenu.add(quitItem);

   ...
}
```

Skab en menubar og lad den være menubar for rammen

Skab en menu med navnet File og tilføj den til menubaren

Skab en menuindgang med navnet Open og tilføj den til File menuen

Skab en menuindgang med navnet Quit og tilføj den til File menuen

Håndtering af events (actions)

- **Brugerne aktiverer objekterne i GUI'en ved hjælp af mus og tastatur**
  - Man kan trykke på knapper og menuindgange, indtaste tekst i tekstbokse, osv.
- **Når et GUI objekt aktiveres af brugeren genereres et ActionEvent**
  - Dette sendes til alle de objekter, som **abonnerer** på ActionEvents fra det pågældende GUI objekt
- **Man registrerer sig som abonnent via addActionListener metoden**
  - Parameteren fortæller, hvad der skal gøres, når et **ActionEvent e** modtages
  - I dette tilfælde kaldes den private metode **quit**

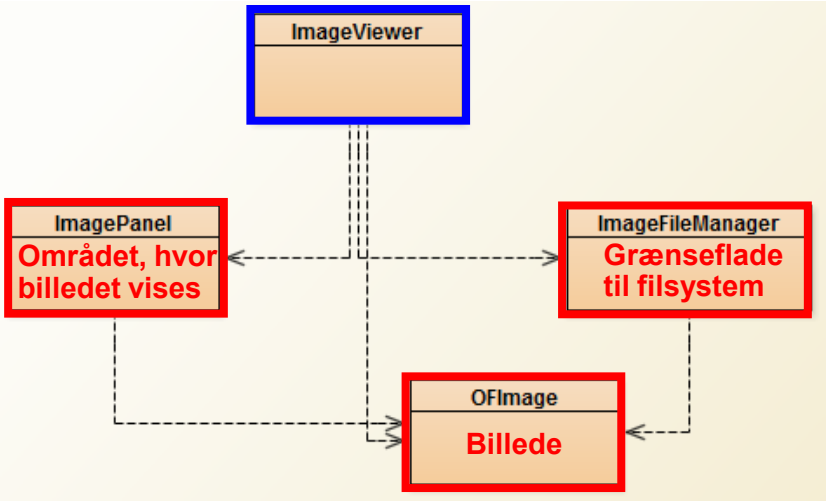
```
private void makeMenuBar(JFrame frame) {
   ...
   JMenuItem quitItem = new JMenuItem("Quit");
   fileMenu.add(quitItem);
   quitItem.addActionListener( e -> quit());
   ...
}
```

Vi kan bruge en lambda, fordi parameteren er af typen ActionListener, som er et funktionelt interface

```
private void quit() {
   System.exit(0);
}
```

exit metoden i System klassen standser udførelsen af programmet
- Parameterværdien 0 indikerer, at det er en normal terminering

Håndtering af billeder



- **Vi introducerer tre nye klasser**
- **OFImage repræsenterer et billede**
  - OFIMage modellerer vores interne billedformat (OF ≈ "Objects First")
  - Bruger et 2-dimensionalt array, hvor hver element angiver en farve fra klassen Color
- **ImageFileManger er grænsefladen til filsystemet**
  - Indeholder klassemetoder til at konvertere billeder på en fil til et OFIMage objekt og tilbage igen
- **ImagePanel implementerer en Swing komponent**
  - Den er en subklasse af JComponent, og er det område i vinduet, hvori billeder kan vises
  - Indeholder en metode **setImage**, hvor parameterværdien er det OFImage objekt, der skal vises i vinduet (rammen)

openFile metoden

- **I makeFrame metoden skabes et ImagePanel objekt**
  - Objektet assignes til feltvariablen imagePanel og tilføjes til contentPane

```
private void makeFrame() {
   ...
   imagePanel = new ImagePanel();
   contentPane.add(imagePanel);
   ...
}
```

Klassemetode i ImageFileManager
- Åbner en dialogboks, hvori brugeren vælger en fil
- Filens billede returneres som et OFImage objekt

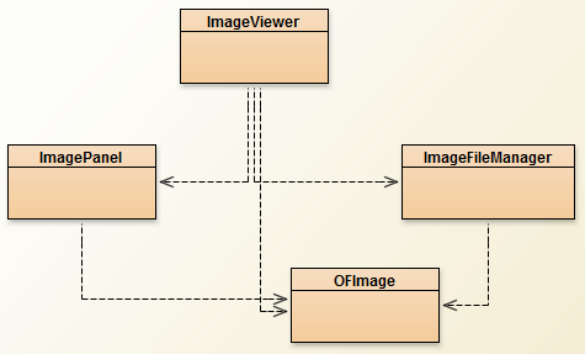
- **Når brugeren vælger Open i File menuen kaldes openFile metoden**

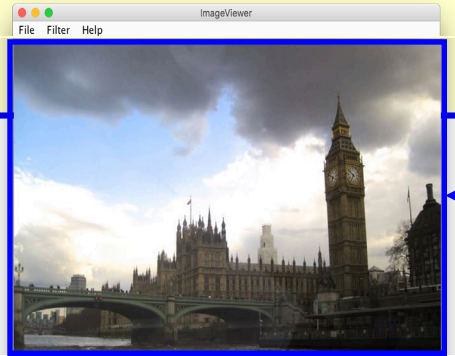
```
openItem.addActionListener( e -> openFile());
```

```
private void openFile() {
   OFImage image = ImageFileManager.getImage();
   imagePanel.setImage(image);
   frame.pack();
}
```

Det indlæste OFImage objekt vises i imagePanel objektet

Rammen pakkes, idet imagePanel objektet har skiftet indhold og dermed størrelse

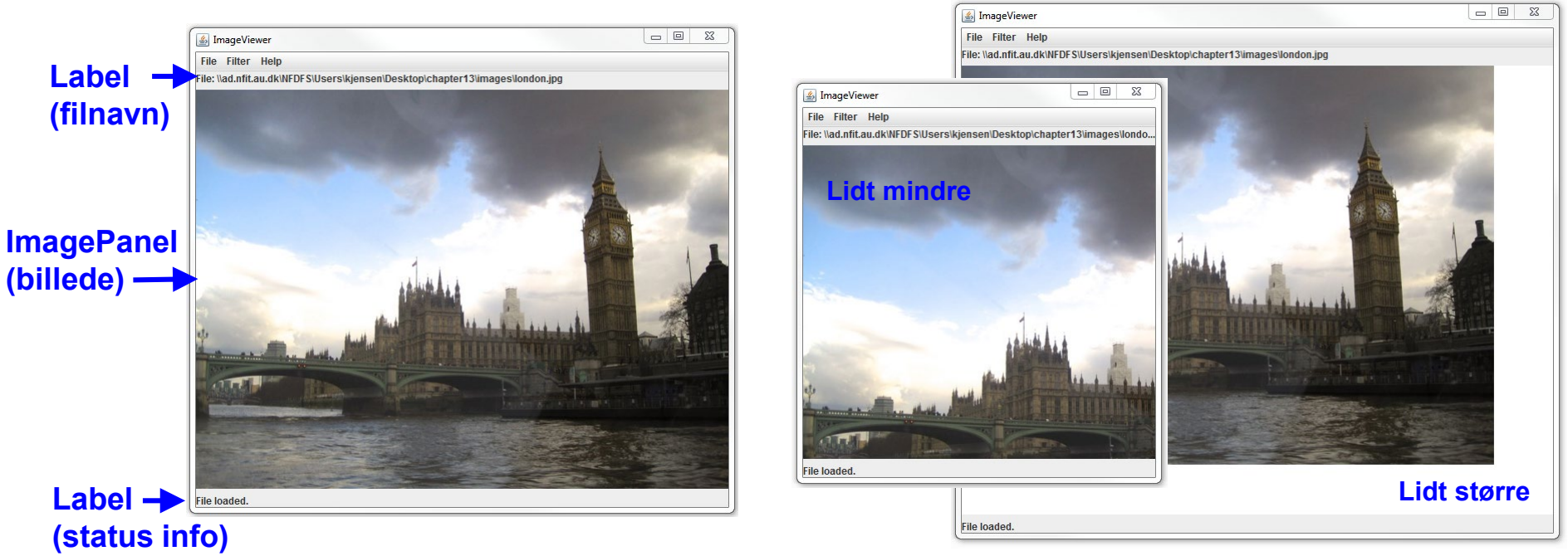




imagePanel

Layout managers

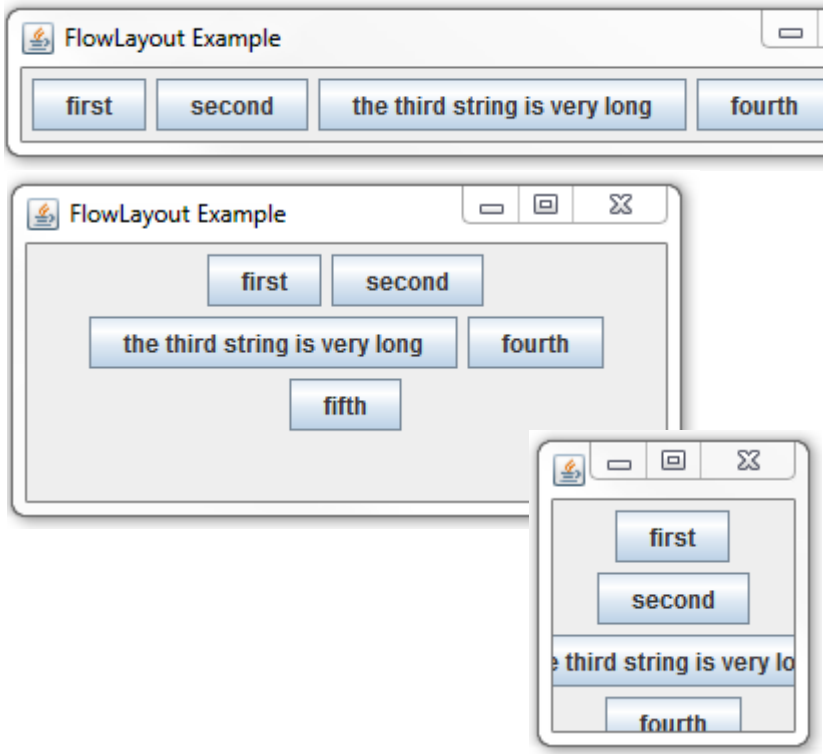
- **Swing bruger layout managers til at bestemme, hvordan de enkelte elementer i en frame placeres i forhold til hinanden**
  - Det er en layout manager, der sørger for, at de to labels i nedenstående vindue placeres hhv. over og under billedet, og at de er venstrejusteret



  - Det er også layout manageren, der bestemmer, hvad der sker med de tre elementer, når billedet gøres mindre eller større
  - Der er mange forskellige layout managers, som vi nu vil studere (nogle af)

Flow layout

- **Elementerne placeres efter hinanden fra venstre mod højre**
  - Om nødvendigt begyndes på en eller flere nye linjer, der alle centreres horisontalt
  - Elementernes størrelse ændres ikke, når vinduet skaleres
  - Den horisontale og vertikale afstand mellem elementerne er fast



Her vil vi kun se på layout managernes "standard" opførsel

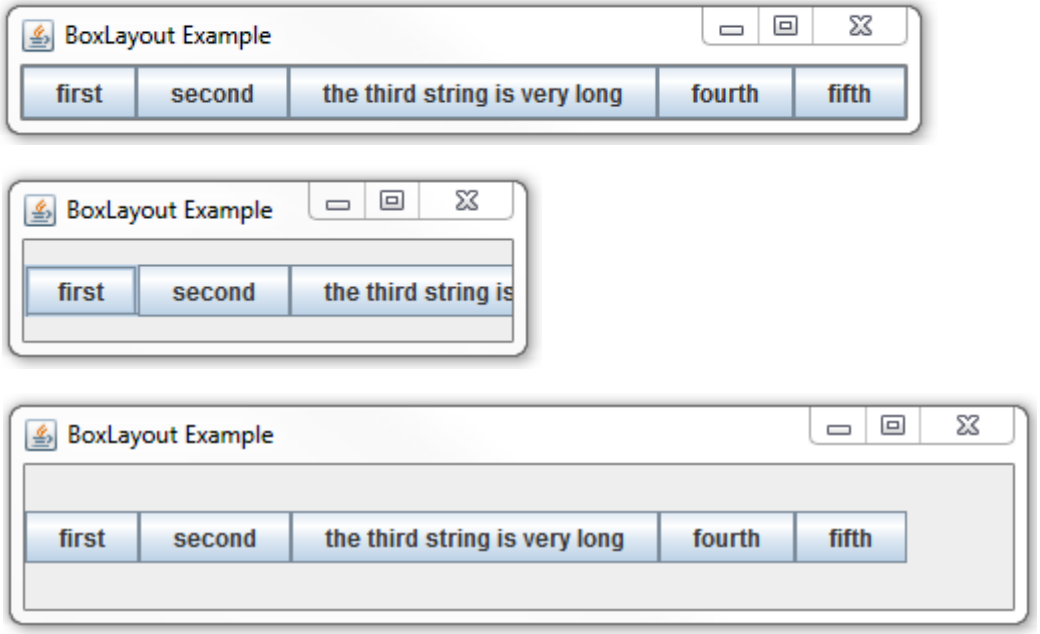
Alle layout managers har parametre, der bestemmer deres detaljerede opførsel

- **Venstre mod højre / højre mod venstre**
- **Vertikalt / horisontalt**
- **Afstand mellem elementerne**
- **Alignment, osv.**

Se Java API'en for detaljer

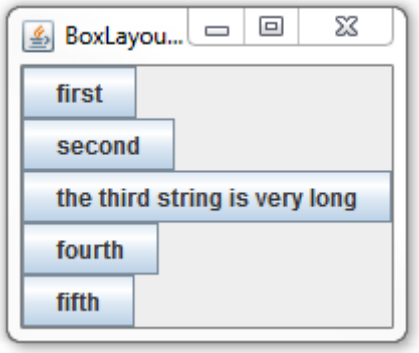
Horisontalt box layout

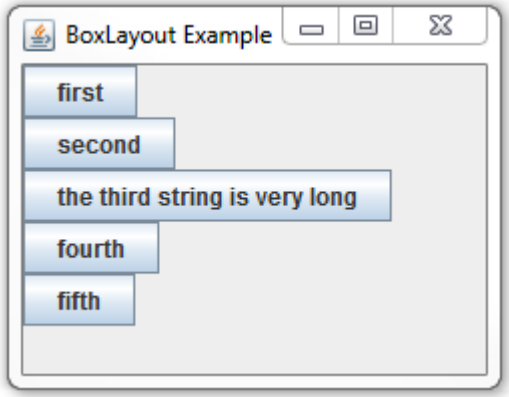
- **Ligner flow design, men opfører sig anderledes, når vinduet skaleres**
  - Elementerne placeres fra venstre mod højre på **én enkelt linje**, der er **venstrejusteret og centreres vertikalt**
  - Elementernes størrelse ændres ikke, når vinduet skaleres
  - Den horisontale afstand mellem elementerne er fast



Vertikalt box layout

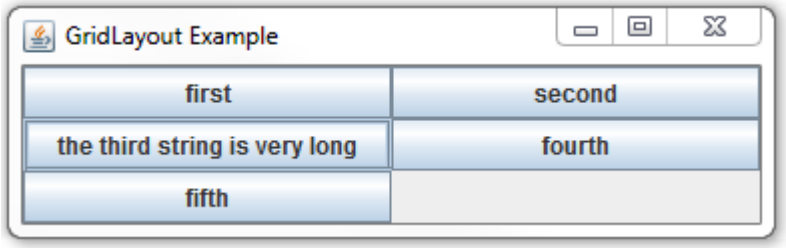
- **Nu placeres elementerne vertikalt under hinanden**
  - Elementerne er venstrejusteret
  - Elementernes størrelse ændres ikke
  - Om nødvendigt forkortes nogle af teksterne
  - Den vertikale afstand mellem elementerne er fast

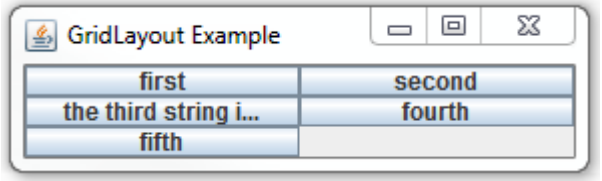




Grid layout

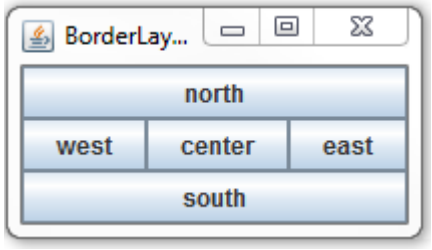
- **Elementer placeres i et gitter (grid)**
  - Elementernes har ens størrelse, og denne tilpasses, så vinduet fyldes ud
  - Der kan dog være ubrugte pladser i gitteret
  - Om nødvendigt forkortes nogle af teksterne

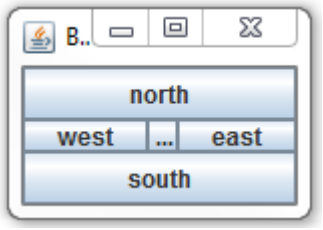


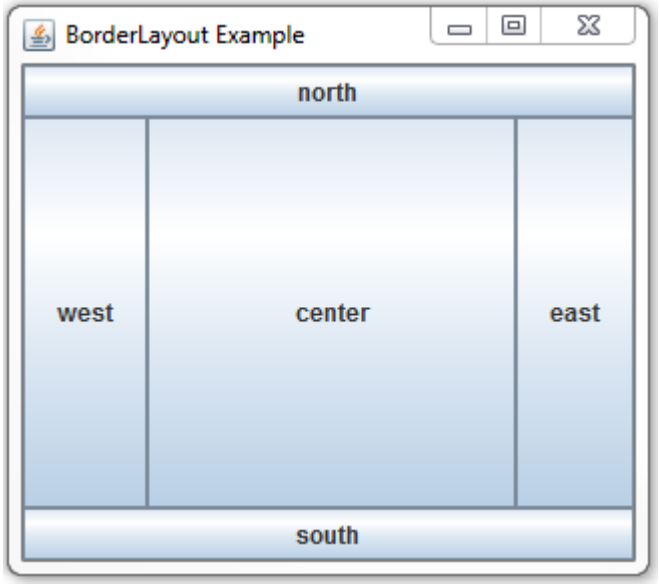


Border layout

- **Der er fem elementer (hvoraf et eller flere kan udelades)**
  - Når vinduet skaleres er det primært størrelsen på center elementet, der ændres
  - Vestlige og østlige element har fast bredde
  - Nordlige og sydlige element har fast højde



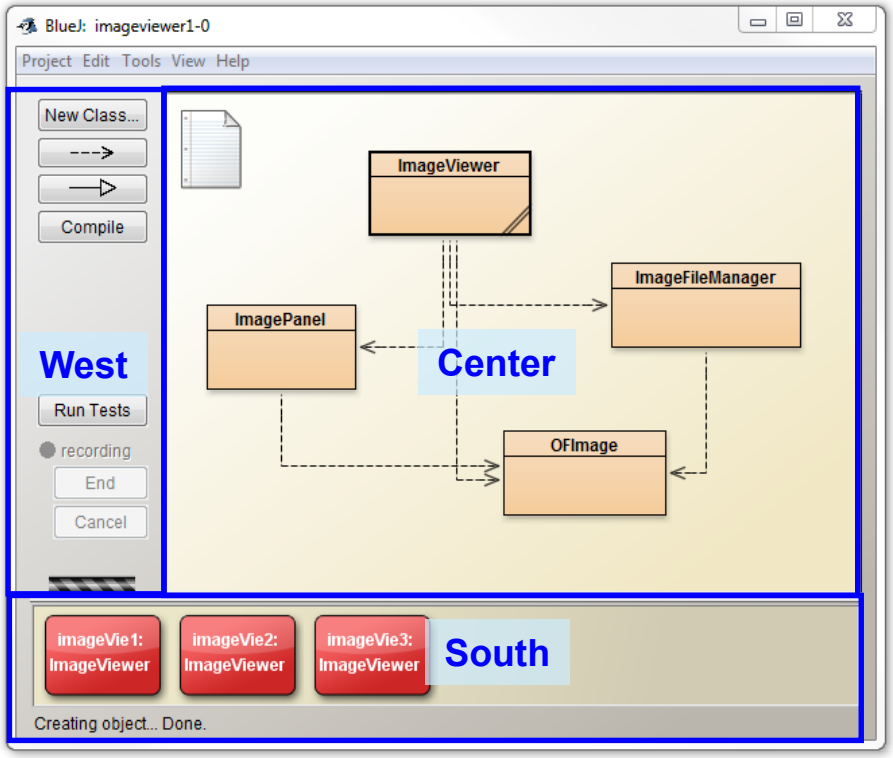




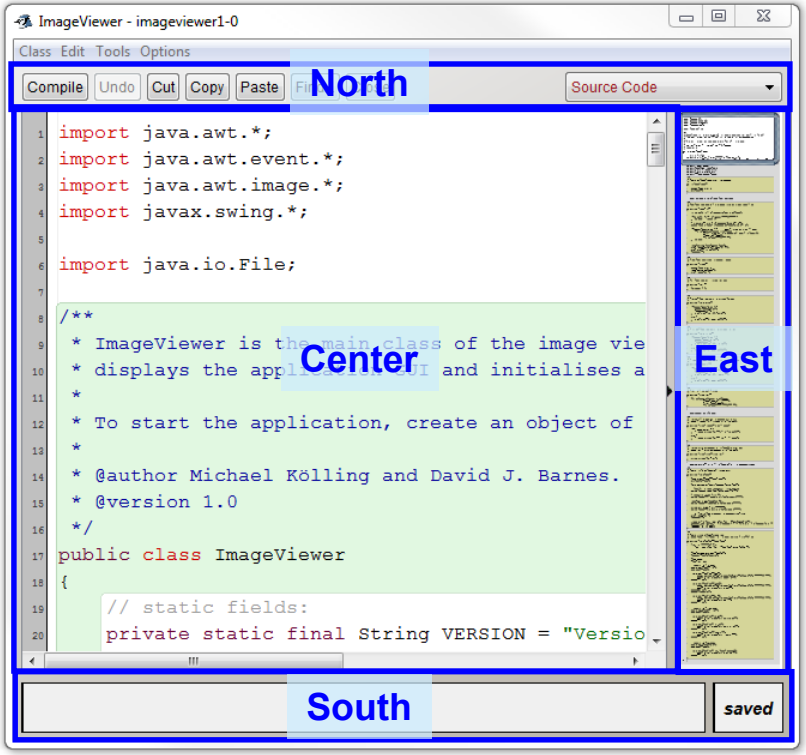
- **Umiddelbart kan man tro, at border layoutet er for specielt til at være nyttigt, men det er ingenlunde tilfældet**

Border layout (fortsat)

- **BlueJ's vinduer er Border layouts**



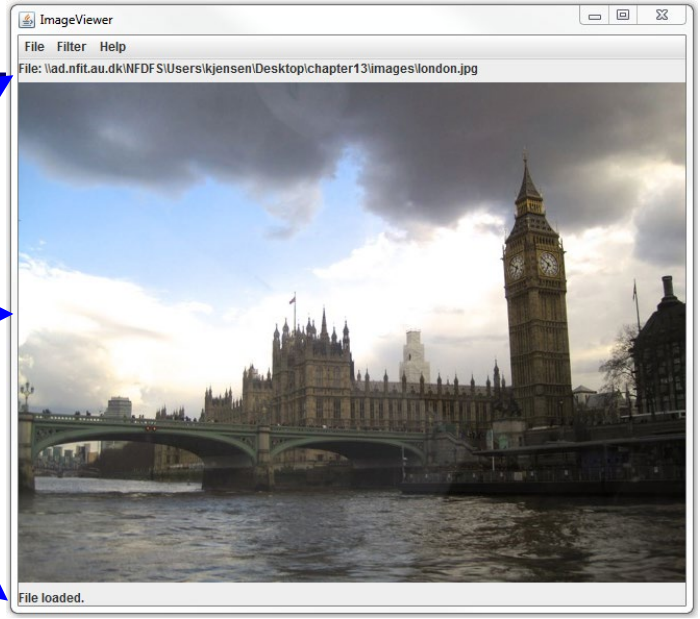
North og East mangler



West mangler

Border layout (fortsat)

- **Vores vindue med billedet er også et border layout**
  - De to labels er placeret i henholdsvis North og South, mens billedet er placeret i Center
  - West og East mangler



Feltvariablen contentPane sættes til at pege på rammens content pane

Sæt layoutet til Border

Skab første label og placér den i NORTH

Skab et ImagePanel og placér det i CENTER

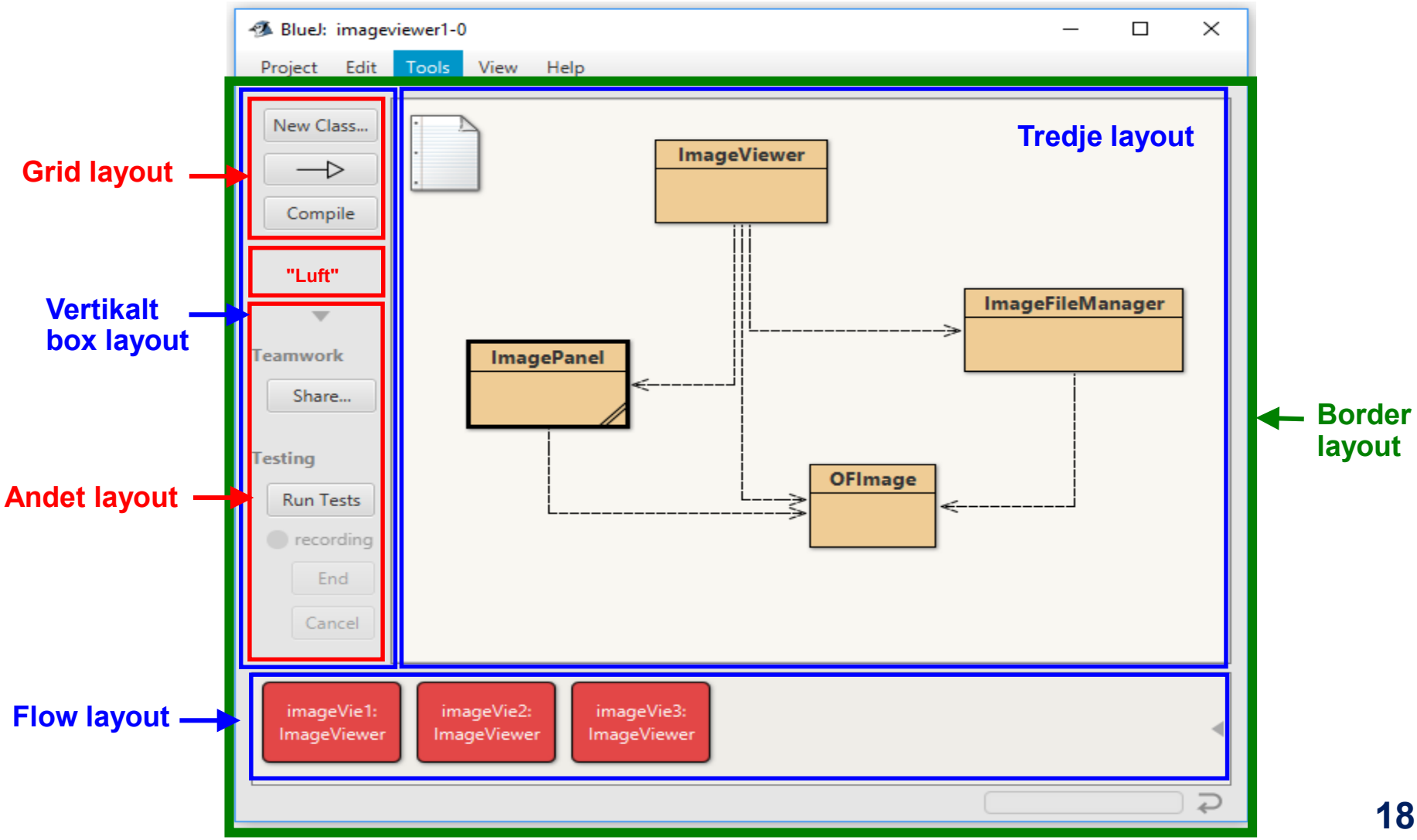
Skab anden label og placér den i SOUTH

```
private void makeFrame() {
  ...
  Container contentPane = frame.getContentPane();
  contentPane.setLayout(new BorderLayout(6, 6));
  filenameLabel = new JLabel();
  contentPane.add(filenameLabel, BorderLayout.NORTH);
  imagePanel = new ImagePanel();
  contentPane.add(imagePanel, BorderLayout.CENTER);
  statusLabel = new JLabel();
  contentPane.add(statusLabel, BorderLayout.SOUTH);
  ...
}
```

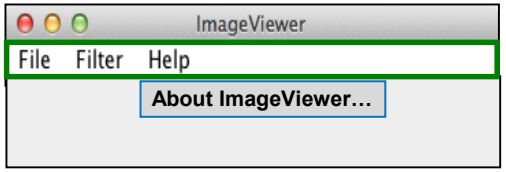
filenameLabel, imagePanel og statusLabel er feltvariabler

Indlejring af layout managers

- De forskellige layout managers kan bruges inde i hinanden



Dialogbokse og knapper

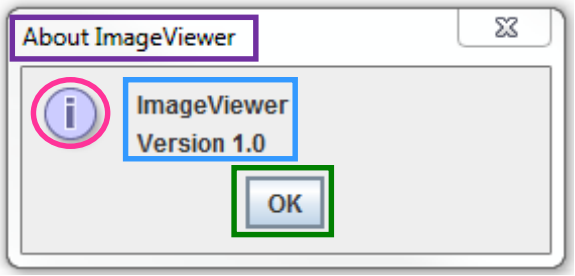


- Vi vil nu lave en menuindgang der åbner en dialogboks

Skab Help menu og tilføj den til menubaren

Skab menuindgang, tilføj den til Help menuen og fortæl, at den skal aktivere showAbout metoden

```
private void makeMenuBar() {
   ...
   Jmenu helpMenu = new JMenu("Help");
   menubar.add(helpMenu);
   JMenuItem aboutItem =
      new JMenuItem("About ImageViewer...");
   helpMenu.add(aboutItem);
   aboutItem.addActionListener( e -> showAbout() );
   ...
}
```



Klassemetode i JOptionPane, hvor parametrene angiver

- Rammen som den tilknyttes
- Teksten, der skal vises
- Titlen, der skal vises øverst
- Message typen
  - INFORMATION_MESSAGE
  - ERROR_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE

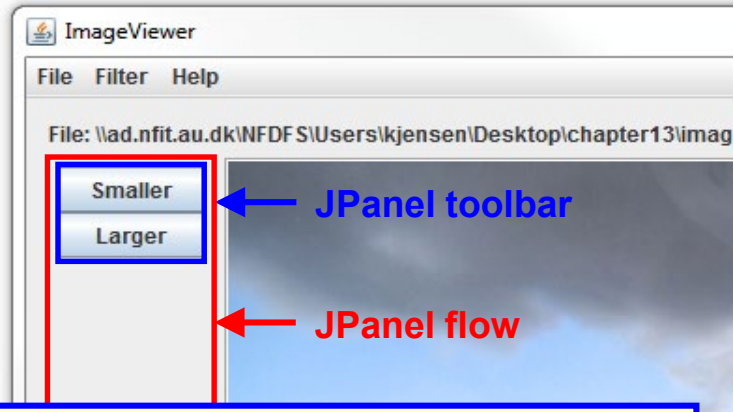
```
private void showAbout() {
   JOptionPane.showMessageDialog( frame,
         "ImageViewer\n" + VERSION,
         "About ImageViewer",
         JOptionPane.INFORMATION_MESSAGE);
}
```

Forskellige slags dialogbokse

- MessageDialog: OK button
- ConfirmDialog: Yes, No, Cancel button
- InputDialog: Tekstfelt til input + nogle knapper

Knapper

- Vi vil nu tilføje et par knapper, der kan ændre billedets størrelse



Skab nyt panel og sæt dets layout manager til GridLayout

Skab den første knap og tilføj den til panelet

Skab den anden knap og tilføj den til panelet

Skab et nyt panel og læg det første panel derind i

Det yderste panel placeres i border layoutets vestlige del

```
private void makeFrame() {
  ...
  JPanel toolbar = new JPanel();
  toolbar.setLayout(new GridLayout(0, 1));
  smallerButton = new JButton("Smaller");
  toolbar.add(smallerButton);
  smallerButton.addActionListener(e -> makeSmaller());
  largerButton = new JButton("Larger");
  toolbar.add(largerButton);
  largerButton.addActionListener(e -> makeLarger());
  JPanel flow = new JPanel();
  flow.add(toolbar);
  contentPane.add(flow, BorderLayout.WEST);
  ...
}
```

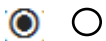
0 ≈ nødvendige antal rækker

antal søjler

smallerButton og largerButton er feltvariabler

- JPanels har FlowLayout som default layout manager
- Tilføjelsen af det yderste panel sikrer at knapperne ikke skaleres i højden (så de fylder hele West)

Andre GUI elementer

- **I denne forelæsning har vi kun set på nogle af de vigtigste elementer, der kan indgå i en grafisk brugergrænseflade**
- **Der er masser af andre**
  - Scrollbarer (klassen Scrollbar) 
  - Checkbokse (klassen Checkbox) 
  - Radiobuttons (klassen JRadioButton) 
  - Lister hvor en/flere indgange kan være selekteret (klassen JList)
  - Dropdown lister, hvor én indgang er selekteret (klassen JComboBox)
  - Billeder (klassen ImageIcon)
  - Kanter/rammer (interfacet Border og dets implementerende klasser)
  - Usynlige elementer som påvirker layoutet (Box klassen)
- **Fremgangsmåden er hele tiden den samme**
  - Skab GUI objekterne og tilføj dem til rammer, paneler og andre Container objekter
  - Tilknyt en passende LayoutManager til containeren (eller brug default)
  - Abonnér på de ActionEvents, der sendes fra GUI objektet og angiv, hvilken metode, der skal udføres, når GUI objektet aktiveres af brugeren

Gode råd omkring GUI konstruktion

- **Cohesion og læsbarhed**
  - Placer GUI elementerne samlet (i en enkelt eller nogle få klasser) og adskilt fra de ting, der beskriver programmets øvrige funktionalitet
  - Opdel i et antal private metoder (f.eks. makeFrame og makeMenubar)
  - De lambda'erne, man bruger som parametre til addActionListener metoden bør være korte og letlæselige (f.eks. et metodekald til en privat metode, hvori den egentlige kode så placeres)
- **Lad andre gøre arbejdet**

  - Brug de predefinerede GUI objekter i Swing og AWT
  - Mange af disse kan identificere brugerevents og videregiver dem til lyttere (event listeners)
- **Der findes værktøjer, hvor man kan lave en GUI via "plug and play"**
  - Elementerne i vinduer, dialogbokse, menuer og lignende skabes via byggeklodser, der tilpasses og placeres på rette position
  - Herefter kan værktøjet selv generere den nødvendige Java kode – med "huller" til den kode, der skal udføres ved modtagelsen af de forskellige GUI events
  - Anvendelsen af sådanne værktøjer falder uden for rammerne af dette kursus

Anonyme indre klasser

- **Indtil nu har vi specificeret al event håndtering via lambda'er, fx:**

```
largerButton.addActionListener(e -> makeLarger());
```

  - Dette har været muligt, fordi alle vores events har været af typen **ActionEvent**, som man abonnerer på ved at kalde **addActionListener** metoden, hvor parameteren er af typen **ActionListener**, som er et funktionelt interface

- **Desværre har Java også en del ældre "lyttere" fra AWT, der ikke er funktionelle**
  - Det gælder bl.a. KeyListener, MouseListener og MouseMotionListener
  - Håndtering af sådanne events sker typisk ved, at man for hvert event, der kan modtages, definerer en **ny klasse**, der implementerer det pågældende Listener interface (og udfører de operationer, der skal foretages)
  - Vi får derfor en masse små klasser, hvor vi kun har behov for at skabe **ét enkelt objekt** af hver klasse
  - Denne situation kan håndteres ved brug af **anonyme indre klasser**

Erklæring af anonym indre klasse

- **Vi vil se på, hvordan muse-events kan håndteres**
  - Sådanne events genereres, når brugeren trykker på en museknap (udenfor specifikke kontroller såsom knapper, menuindgange, scrollbarer, osv.)

Tilknyt en MouseListener til imagePanel (det område af vores vindue, der ineholder billedet)

```
private void makeFrame() {
   ...
   imagePanel.addMouseListener(new MouseAdapter()
     {
        public void mousePressed(MouseEvent e) {
           handleMousePressed(e);
        }
     });
   ...
}
```

Start på klasseerklæring

Slut på klasseerklæring

- **Interfacet er ikke funktionelt, og vi kan derfor ikke bruge en lambda som parameter**
  - Parameteren skal være et objekt, der implementer **MouseListener** interfacet
  - Objektet skabes på det sted, hvor det bruges (mellem de to røde parenteser)
  - Klassen er en subklasse af **MouseAdapter** klassen, der implementerer MouseListener interfacet
- **Den ny klasse har intet navn og er erklæret inde i ImageViewer klassen, hvorfor den siges at være en anonym indre klasse**

MouseAdapter klassen

- **MouseAdapter klassens implementation af MouseListener interfacet er helt triviel**
  - Alle otte metoder i interfacet har tomme kroppe i MouseAdaptor klassen
- **Vi skal kun bruge én af metoderne i MouseListener interfacet**
  - Den erklærer vi i den anonyme indre klasse
  - De øvrige syv metoder nedarver vi fra MouseAdapter klassen
  - De har tomme kroppe, men det betyder ikke noget, da vi ikke skal bruge dem
  - På den måde slipper vi for at skulle lave de syv metoder vi ikke bruger

```
private void makeFrame() {
   ...
   imagePanel.addMouseListener(new MouseAdapter()
     {
       public void mousePressed(MouseEvent e) {
         handleMousePressed(e);
       }
     });
   ...
}
```

Start på klasseerklæring

Slut på klasseerklæring

Gentag

- Vi overskriver den "tomme" mousePressed metoden fra MouseAdapter klassen
- Den overskrivende metode kalder blot den private metode handleMousePressed

- Parameteren for addMouseListener skal være et objekt, der implementerer MouseListener interfacet
- Det opnår vi ved at bruge en subklasse af MouseAdapter klassen (der implementerer interfacet)

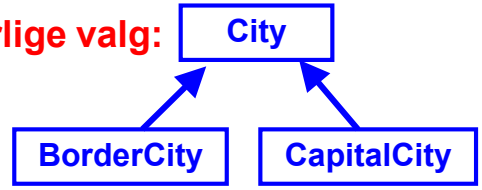
Indre klasser

- **Indre klasser behøver ikke være anonyme**
  - Man kan have en helt almindelig (ikke-anonym) klasse inden i en anden klasse (den indre klasse placeres normalt sidst i den ydre klasse)
  - På den måde kan man opdele en stor og kompleks klasse i flere (nært forbundne) klasser og dermed forbedre læsbarheden (øget cohesion)
  - Ved at bruge indre klasser (i stedet for almindelige klasser) har man adgang til feltvariablerne og metoderne i den omgivende klasse
  - I computerspilsopgaven har GUI klasen en indre klasse, WorldPanel, som implementerer den del af vinduet, der indeholder landkortet
- **Objekter af en indre klasse kan kun eksistere "inde i" objekter af den omgivende klasse**
  - Det indre objekt skabes sammen med det omgivende objekt og dør sammen med det
- **Før introduktionen af funktionelle interfaces i Java 8 skulle man også lave en anonym indre klasse for hvert ActionEvent**
  - Nu kan man (som vi har set) i stedet bruge en lambda som parameter
  - Det er meget lettere – både at skrive og forstå

Computerspil 3

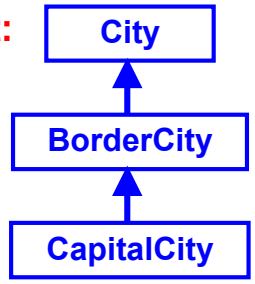
- **I den tredje delaflevering skal I bruge nogle af de ting, som I har lært om nedarvning og dynamic method lookup til at strukturere jeres kode**
  - I skal indføre flere forskellige slags byer/lande:
    - **BorderCity** repræsenterer en grænseby, hvor man skal betale told, når man ankommer fra udlandet
    - **CapitalCity** repræsenterer en hovedstad, hvor der er mange fristelser, så man (udover at modtage bonus) bruger af sin formue
    - **MafiaCountry** repræsenterer et land (Sverige!), hvor man risikerer at blive overfaldet og frarøvet dele af sin formue

Naturlige valg:



Men vi har i stedet valgt:

Hvorfor mon det?



- **Herudover skal I**
  - rette gamle fejl og mangler
  - holde jeres dokumentation og regression tests opdaterede
  - herunder tilføje dokumentation og regression tests for nye programdele

Regression tests for BorderCity / CapitalCity

- **Testmetoden for arrive metoden i BorderCity kan være næsten identisk med den tilsvarende testmetode i City klassen**
  - Den væsentlige forskel er, at der skal betales told, hvis spilleren kommer fra et andet land, f.eks. fra City E til City C

Skaber en spiller, der ankommer fra City E til City C med en formue på 250 €

Beregn 20% told

Tag hensyn til tolden

Disse to byer ligger i forskellige lander

```
@Test
public void arriveFromOtherCountry() {
  for(int seed = 0; seed < 1000; seed++) {
    Player player = new GUIPlayer(new Position(cityE, cityC, 0), 250);
    game.getRandom().setSeed(seed);              // Set seed
    int bonus = country1.bonus(40);              // Remember bonus
    int toll = 250 / 5;                          // 20% of 250
    game.getRandom().setSeed(seed);              // Reset seed
    assertEquals(..., cityC.arrive(player));     // Same bonus
    assertEquals(..., cityC.getValue());
    cityC.reset();
  }
}
```

I skal også lave en testmetode, der tjekker, at der ikke betales told, når spilleren kommer fra samme land

- **For CapitalCity klassen laves tilsvarende testmetoder**
  - Nu skal man også tage hensyn til de penge, som spilleren bruger i hovedstaden

Regression test for MafiaCountry

- **Testmetoden for bonus metoden i MafiaCountry er analog til den tilsvarende testmetode i Country klassen**

I skal tjekke, at

- tabet ved røveriet ligger i intervallet [10,50]
- man bliver røvet ca. 20% af gangene
- tabet i gennemsnit udgør ca. 30 €
- tabet kan antage alle værdier i intervallet [10,50]

Husk også at tjekke, at bonussen udregnes korrekt, når man ikke bliver røvet

```
@Test
public void bonus() {
  for(int seed = 0; seed < 1000; seed++) {
    game.getRandom().setSeed(seed);
    int robs = 0;
    int loss = 0;
    Set<Integer> values = new HashSet<>();
    for(int i = 0; i<50000; i++) {
      int bonus = country2.bonus(80);
      if(bonus < 0) {  // Robbery
        robs++;
        assertTrue( ...);
        loss -= bonus;
        values.add(-bonus);
      }
      else {  // No Robbery
        ...
        ...
      }
    }
    assertTrue(...);
    assertTrue(...);
    assertEquals(...);
    ...
  }
}
```

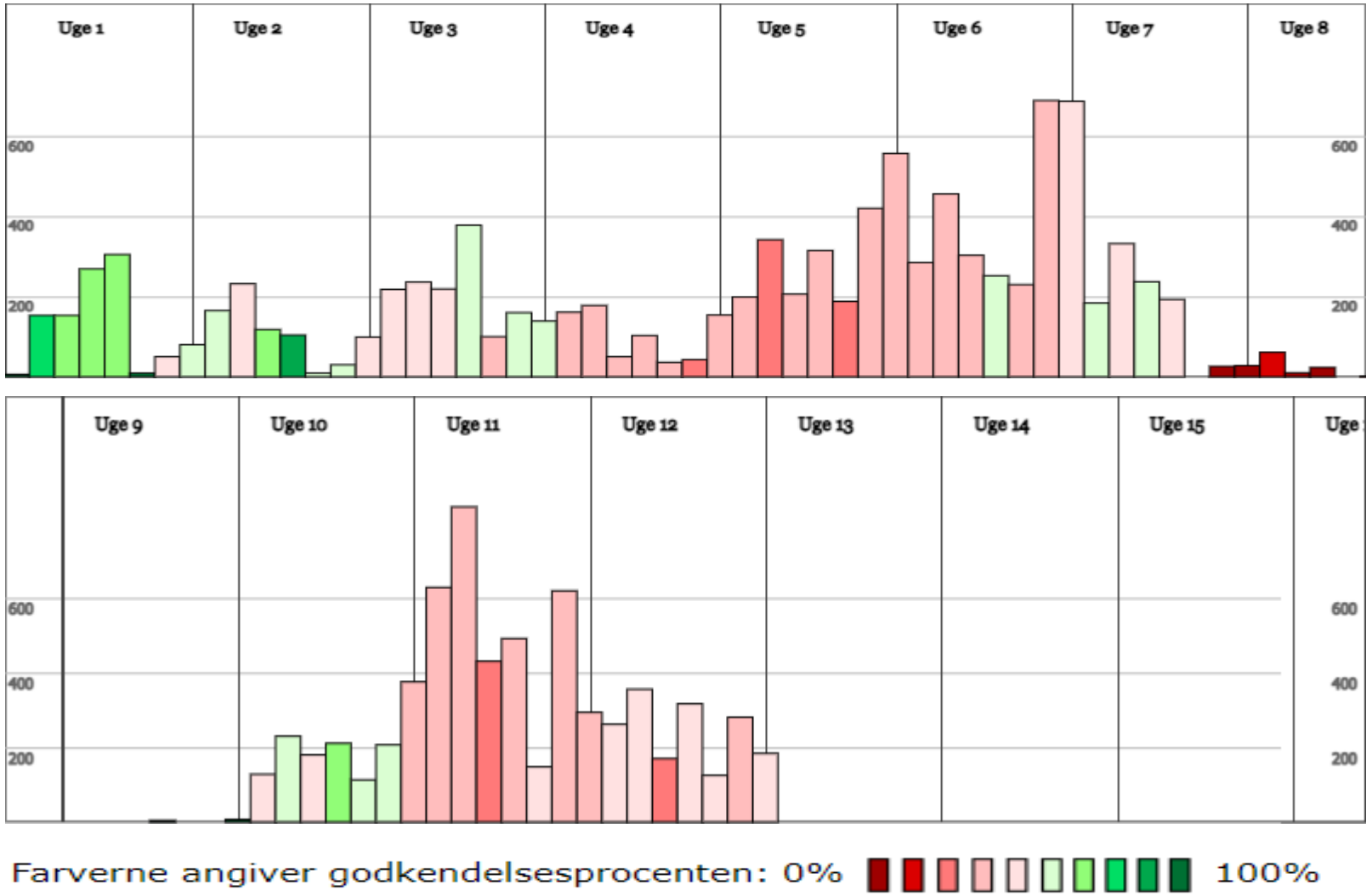
Mafialand

Testserveren

- **Testserveren skal også anvendes for Computerspil 3**
  - Her testes kun de **nye klasser**, som I har skrevet i CG3
  - For hver klasse udføres en række **regression tests** for konstruktørerne og metoderne (på tilsvarende vis som i Computerspil 1)
  - Derudover testes det, at jeres **regression tests** er fornuftige (på tilsvarende vis som i Computerspil 2)
- **Brug Testserveren med omtanke**
  - Når I får en fejlrapport, bør I rette alle de fejl, der rapporteres og kontrollere, at rettelserne er korrekte, **før** I atter forsøger at køre TestServeren
- **Testserveren er et stort og komplekst stykke kode (50.000+ linjer)**
  - Det er derfor ikke underligt, at den sommetider indeholder fejl og går ned
  - Nogle nedbrud skyldes upload af kørsler med en uendelig løkke/rekursion
- **Inden man bliver irriteret på testserveren, skal man huske, hvordan situationen ville være, hvis I ikke havde den**
  - Så skulle I helt på egen hånd finde frem til, hvor fejlene er i jeres kode
  - Nu får I at vide, hvilke klasser og metoder, I skal søge fejlene i

Statistik for brug af testserveren

- **Ca. 17.000 kørsler indtil nu, dvs. knap 100 kørsler pr studerende**



Opsummering

- **Konstruktion af grafiske brugergrænseflader (GUI'er)**
  - Definition af de elementer, der vises på skærmen (vinduer, knapper, menuer, scrollbarer, tekster, osv.)
  - Hvordan reagerer de på input (via mus og tastatur)?
  - Hvordan placeres de i forhold til hinanden (layout)?
- **Anonyme indre klasser**
  - Sprogkonstruktion, der er nyttig i forbindelse med visse GUI events
- **Afleveringsopgave: Computerspil 3**
  - Brug af nedarvning og dynamic method lookup

Computerspillets GUI klasse indeholder næsten 1000 linjer kode

- **Kig evt. lidt på den og find eksempler på nogle af de ting, som jeg har gennemgået i denne forelæsning**
- **I Computerspil 4 skal i lave nogle simple modifikationer/udvidelser af GUI klassen**

Mundtlige præsentationer

- **Husk at se den sidste video om den "perfekte" mundtlige præsentation**
- **Den handler om grafiske brugergrænseflader**
- **Findes under Uge 13**

Det var alt for nu….. … spørgsmål