# Programmierung 2

DHBW Mannheim - WWI17SCB - May 2018

# Slides

https://goo.gl/U45wY3

# Morgen: Beginn 10:30

# Geschwindigkeit / Fragen

# Etwas Wiederholung...

# Collections - Wiederholung

1. Was sind Collections und wozu dienen sie?

2. Welche wesentlichen Collection-Interfaces gibt es?

# Collections - Wiederholung

3. Was sind die wesentlichen Eigenschaften von...

    * List

    * Set

    * Map

# Collections - Wiederholung

4. Mithilfe welcher Klasse kann ich über Collections itieren?

   Wie bekomme ich ein Objekt dieser Klasse?

   Welche wesentlichen Methoden hat es?

5. Wie iteriere ich über die Werte einer Map, wenn ich die Schlüssel nicht kenne?

# Collections - Wiederholung

6. Was ist ein "generischer Typ" und wozu dient er?

7. Was sind die zwei wesentlichen Vorteile der Verwendung einer Typ-parametrisierten Collection gegenüber dem "raw type"?

# Collections - Wiederholung

8. Welches Interface müssen Objekte implementieren, die in eine Collection des Typs "SortedSet" eingefügt werden?

9. Was ist die "natural ordering" bezogen auf Datentypen?

10. Was ist ein Comparator? Wie unterscheidet er sich von Comparable?

# Collections - Wiederholung

11. Wie kann ich eigene Logik zur Überprüfung der Gleichheit von Objekten in Collections implementieren?

12. Welche Prüfungen muss die "equals"-Methode zwingend durchführen?

# Collections - Wiederholung

13. Wozu brauche ich die "hashCode"-Methode?

14. Was ist die Beziehung zwischen "hashCode" und "equals"?

15. Was sind "Wrapper-Klassen"?

16. Was ist Autoboxing bzw. -unboxing?

# 4. GUIs mit Swing

# Swing - Lernziele

* Die wesentlichen GUI-Toolkits in Java kennen
* Mit Swing einfache Fenster erzeugen und schließen
* Layouts in Verbindung mit Panels einsetzen
* Einfache Benutzerdialoge mit Swing- Komponenten erstellen
* Validierende Textfelder erstellen
* Die Interfaces Action- und ItemListener einsetzen
* Eigene Menüs implementieren
* Benutzeroberfläche mit Rahmen und Tooltips ergänzen

# GUIs in Java

Bestandteil der Java API:

* AWT
* Swing
* JavaFX

3rd-Party-Lösungen:

* SWT / JFace

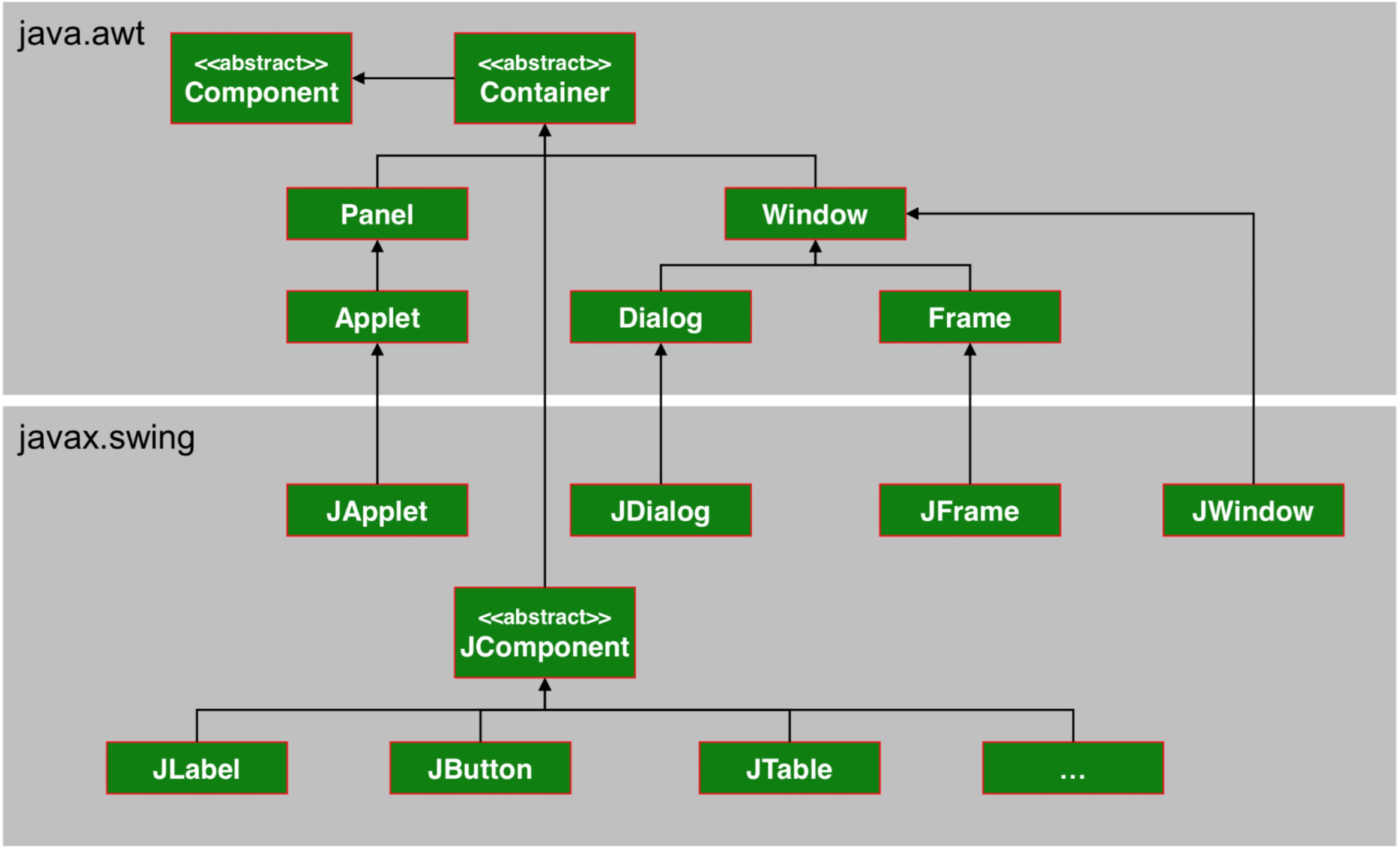
...

# AWT - "Heavyweight Components"

* Verwendung von plattformspezifischen Implementierungen der AWT-Klassen (nicht in Java implementiert !)
* AWT-Komponenten besitzen einen Partner auf Betriebssystem-seite (Peer), der Darstellung und Funktionalität steuert
* Vorteil: sehr schnell, da die Peer-Klassen im Code der Ausführungsplattform geschrieben sind
* Nachteil: unhandliche API

# Swing - "Lightweight Components"

* "Nachfolger" von AWT
* Nur sehr wenige plattformspezifische GUI-Ressourcen
* Lightweight Components besitzen keinen Peer auf Betriebs-systemseite
* Swing besitzt zahlreiche zusätzliche GUI-Komponenten
* Vorteil: „bessere“ Plattformunabhängigkeit
* Nachteil: im Vergleich zu AWT weniger performant


java.awt
<<abstract>>
Component
<<abstract>>
Container
Panel
Window
Applet
Dialog
Frame
javax.swing
JApplet
JDialog
JFrame
JWindow
<<abstract>>
JComponent
JLabel
JButton
JTable
...

# JavaFX - The new kid

* "Nachfolger" von Swing seit ca. 2014
* Neuerungen v.a. im Bereich Media / Animation
* GUI-Lösung für alle Plattformen
* Verfügbarkeit einer WebView-Komponente & Interaktion mit Java
* Deklarative UI-Modellierung mit XML-Dialekt FXML
* Styling der Komponenten per CSS
* Aber: Supportende vermutlich schon 2022

# Bestandteile von JFC

JFC = Java Foundation Classes

* Swing GUI Components
* Pluggable Look-and-Feel Support (PLAF)
* Accessibility API
* Java 2D API
* Internationalization

https://docs.oracle.com/javase/tutorial/uiswing/start/index.html

# Bestandteile von Swing

## 18 Core Packages...

| | | |
|---|---|---|
| javax.accessibility | javax.swing.plaf | javax.swing.text |
| javax.swing | javax.swing.plaf.basic | javax.swing.text.html |
| javax.swing.border | javax.swing.plaf.metal | javax.swing.text.html.parser |
| javax.swing.colorchooser | javax.swing.plaf.multi | javax.swing.text.rtf |
| javax.swing.event | javax.swing.plaf.synth | javax.swing.tree |
| javax.swing.filechooser | javax.swing.table | javax.swing.undo |

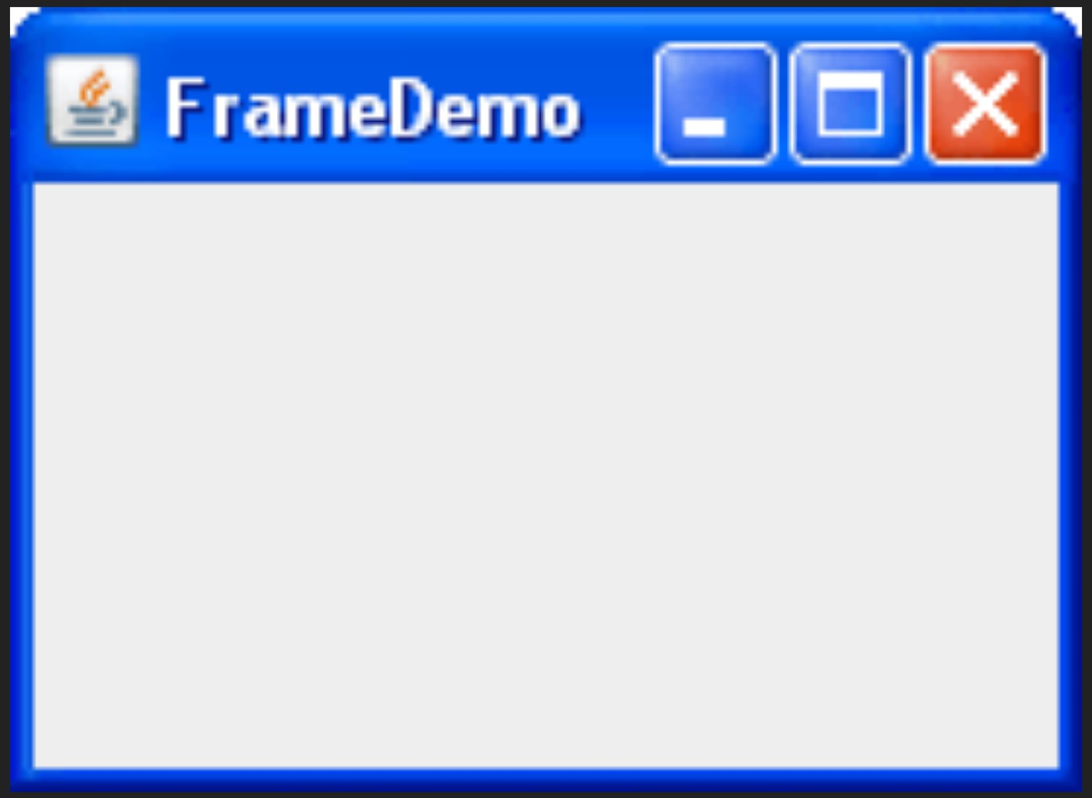
# Top-Level Container

# Top-Level Container

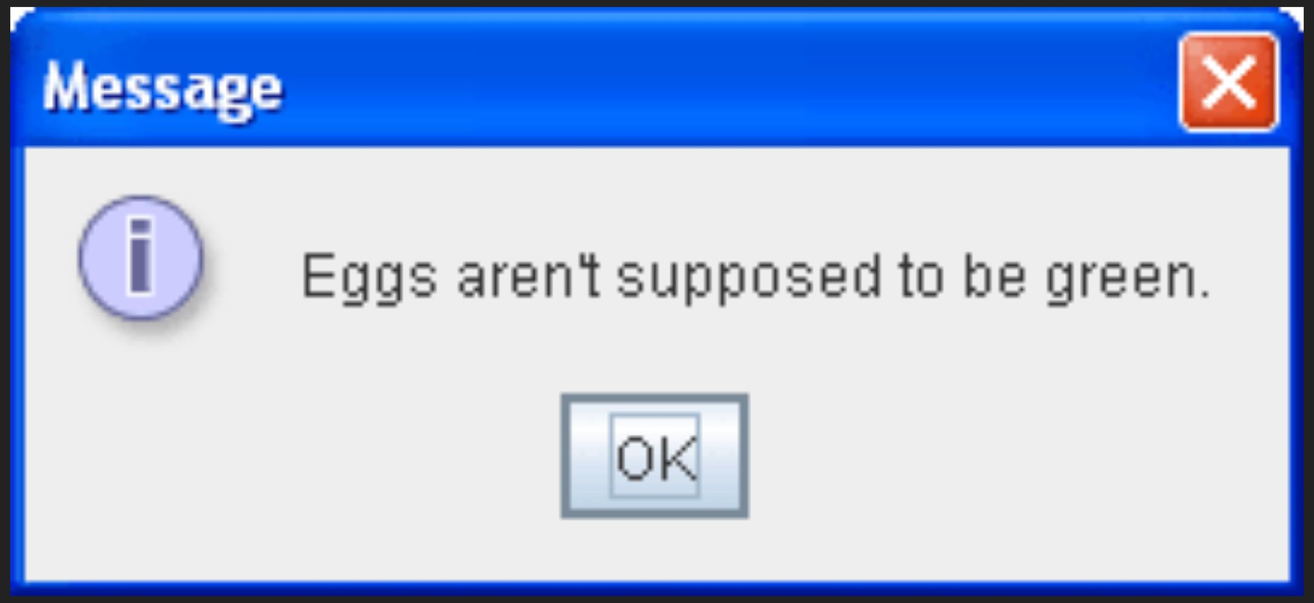
* "Wurzel" des GUI-Komponentenbaums
* JFrame, JDialog, JApplet

https://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html

# JFrame



# JDialog



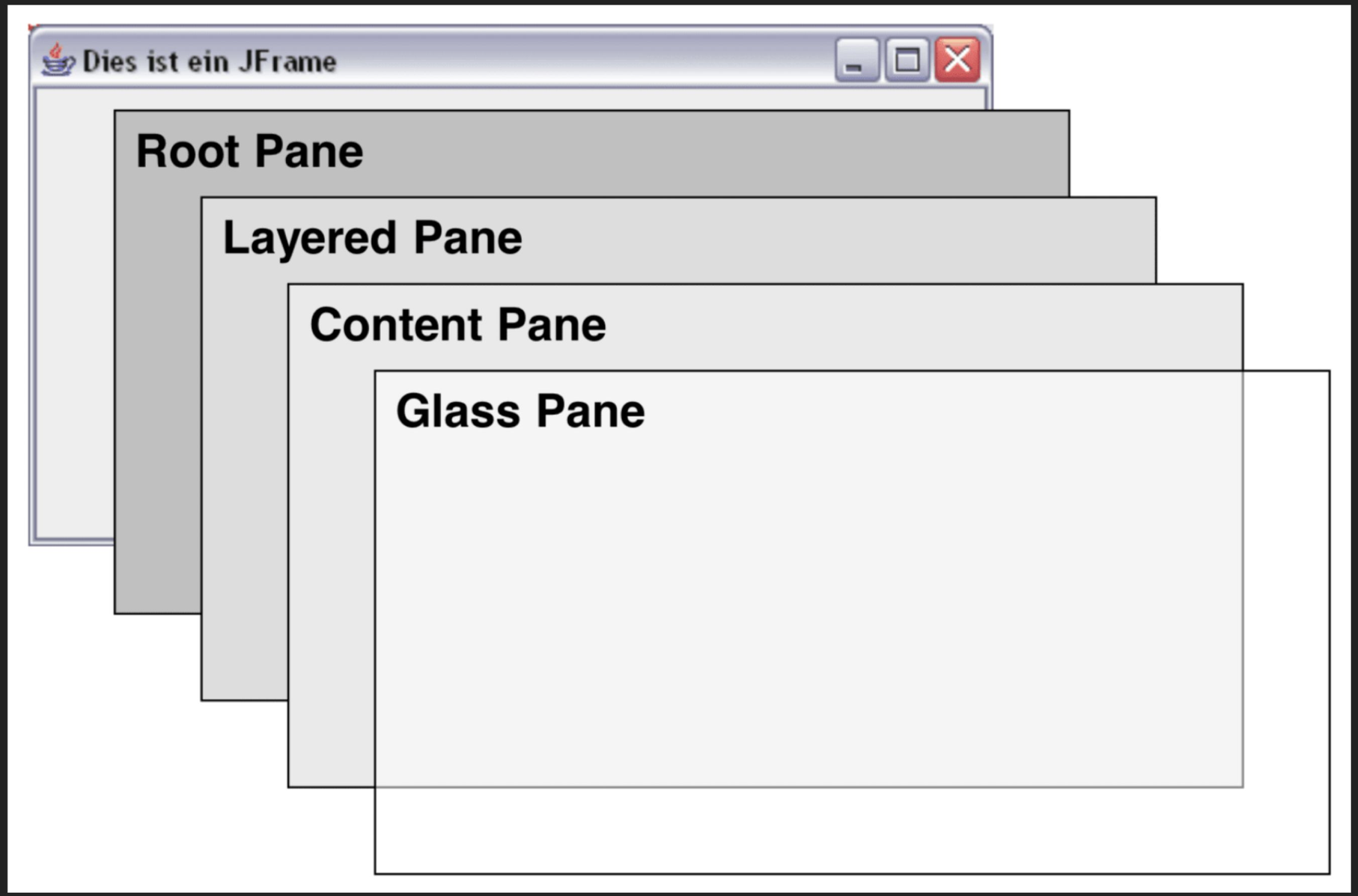
# ~~JApplet~~

# Top-Level Container

* Jede GUI-Komponente außer Top-Level-Containern muss in einer Container-Hierarchie stecken
* Container-Hierarchie = Baum von GUI-Komponenten
* Wurzel des Baumes = Top-Level-Container
* GUI-Komponenten haben immer nur eine Elternkomponente
* TLCs haben ein "ContentPane", das die weiteren UI-Komponenten enthält

https://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html

# Top-Level Container



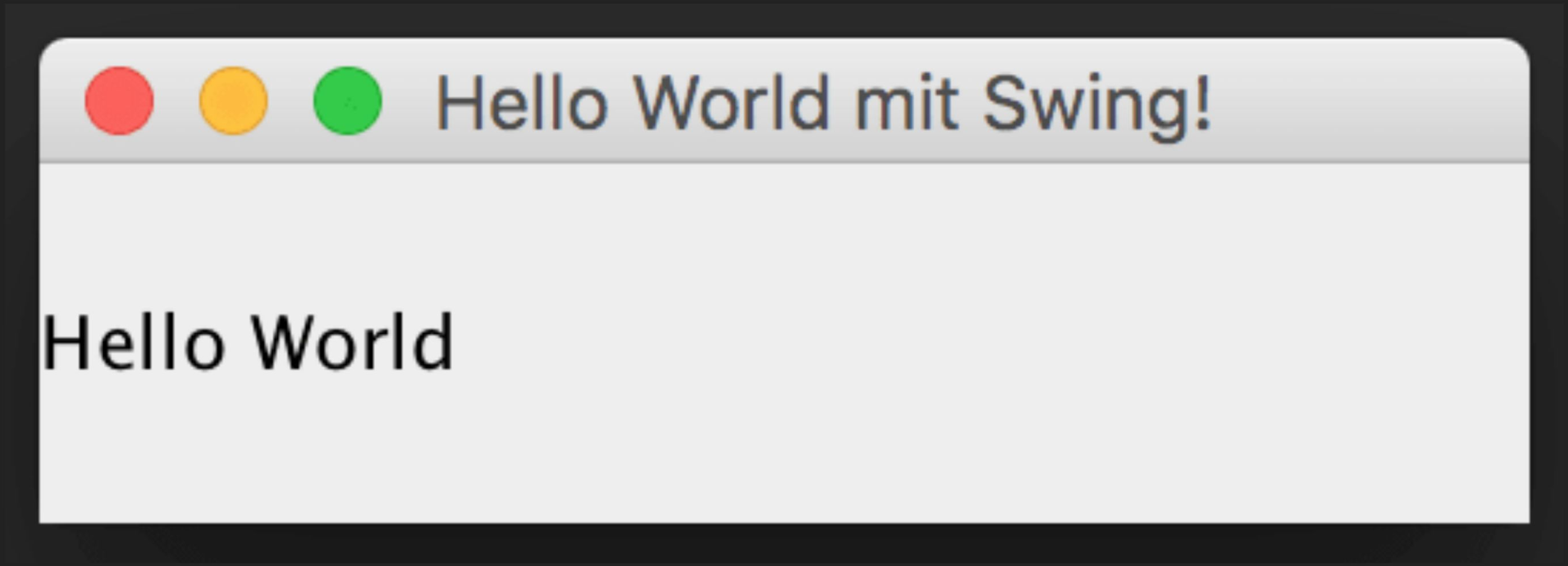
# Swing - Codebeispiel 1

```
JFrame frame = new JFrame("Hello World mit Swing!");

// Füge ein "Hello World" zum ContentPane hinzu
JLabel label = new JLabel("Hello World");
frame.getContentPane().add(label);

// Zeige das Fenster an.
frame.pack();
frame.setVisible(true);
```

# Swing - Codebeispiel 1



# Close Operations

* Festlegen, was beim Schließen des Fensters passieren soll:

```
public void setDefaultCloseOperation(int operation)
```

* WindowConstants.DO_NOTHING_ON_CLOSE löst nur Event aus
* WindowConstants.HIDE_ON_CLOSE versteckt das Fenster
* WindowConstants.DISPOSE_ON_CLOSE zerstört den Frame
* WindowConstants.EXIT_ON_CLOSE beendet die Applikation

# Swing - Codebeispiel 2

```
JFrame frame = new JFrame("Hello World mit Swing!");
// Beim Schließen Programm verlassen
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

// ...
```

# Fenstergröße bestimmen

## Größe und Position festlegen

```
public void setBounds(int x, int y, int width, int height)
```

## Nur Größe festlegen

```
public void setSize(int width, int height)
```

## Fenster auf Inhaltsgröße abstimmen

```
public void pack()
```

# Swing - Codebeispiel 3

```
JFrame frame = new JFrame("Dieses Fenster ist 500*300 Pixel groß!");
frame.setSize(500, 300);
```

# Fenstergröße bestimmen

## Größe und Position festlegen

```
public void setBounds(int x, int y, int width, int height)
```

## Nur Größe festlegen

```
public void setSize(int width, int height)
```

## Fenster auf Inhaltsgröße abstimmen

```
public void pack()
```

# Sichtbarkeit bestimmen

Fenster zeigen/verstecken

```
public void setVisible(boolean b)
```

# Bestandteile des JFrame holen

* `public JLayeredPane getLayeredPane()`
* `public Container getContentPane()`
* `public Component getGlassPane()`
* `public JMenuBar getJMenuBar()`

# Komponenten hinzufügen

```
JFrame frame = new JFrame("Komponenten hinzufügen");

JLabel label = new JLabel("Hello World");
// JFrame.add leitet an ContentPane weiter
// frame.getContentPane().add(label);
frame.add(label);

// Zweites Label hinzufügen
frame.add(new JLabel("Zweites Label!"));

// Erstes Label wieder entfernen
frame.remove(label);
```

# UI-Komponenten

# Komponenten - JComponent

* Alle GUI-Komponenten, die in einem TLC platziert werden, erben von JComponent
* Stellt Basisfunktionen bereit für (u.a.):
  - Tooltips
  - Painting and borders
  - Pluggable look and feel, Accessibility
  - Layout
  - Hotkeys, Drag&Drop

https://docs.oracle.com/javase/tutorial/uiswing/components/jcomponent.html

# JLabel - Anzeige von Text

* Anzeige von (nicht-selektierbarem) Text
* Oft zur Beschriftung anderer UI-Elemente
* Kann HTML enthalten (please don't)
* Text- und Hintergrundfarbe änderbar
* Ausrichtung des Texts änderbar (Links, Rechts, Zentriert)

https://docs.oracle.com/javase/tutorial/uiswing/components/label.html

# JLabel - Anzeige von Text

## Text nachträglich ändern

```
public void setText(String text)
```

## Vordergrundfarbe ändern

```
public void setForeground(Color fg)
```

https://docs.oracle.com/javase/tutorial/uiswing/components/label.html

# Farben setzen

`public Color(int rgb)`

* "int rgb" setzt einen RGB (Rot/Grün/Blau) Farbcode

* z.B. "0x00ff00" entspricht Rot 0, Grün 255, Blau 0 -->

```
label.setForeground(new Color(0x00ff00));
```

# JLabel - Anzeige von Text

## Hintergrundfarbe ändern

```
public void setBackground(Color bg)
```

und

```
public void setOpaque(boolean b)
```

```
label.setBackground(new Color(0x000000));
label.setOpaque(true);
```

# JLabel - Textausrichtung

Ausrichtung des Texts

```
public void setHorizontalAlignment(int alignment)
```

oder direkt im Konstruktur:

```
JLabel myRightAlignedLabel = new JLabel("Aligned right", SwingConstants.RIGHT);
```

# Coding Time!

# Swing - Codingaufgabe 1

30m

* Erstellt eine Anwendung, die ein Fenster mit dem Titel "Hallo WIWI17SCB" öffnet
* Platziert ein Label mit Text in dem Fenster
* Verändert Farbe und Hintergrundfarbe des Labels

# Layouts

# Layout Manager

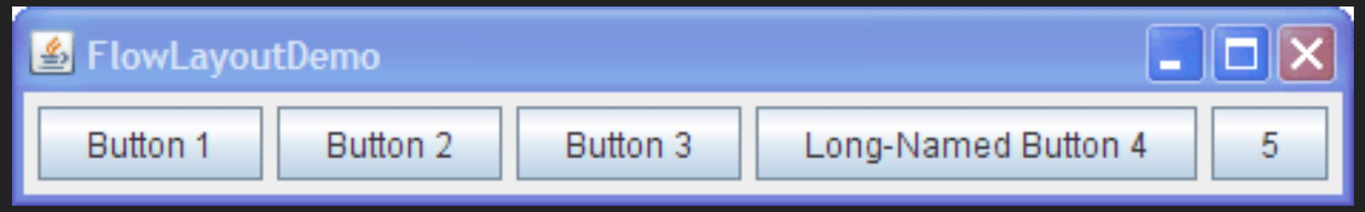
* Anordnung der Elemente eines Containers nach bestimmten Verfahren über Layout-Manager
* Setzen eines Layout-Managers:

```
public void setLayout(LayoutManager manager)
```

```
frame.setLayout(new FlowLayout());
```

# Layout Manager - FlowLayout

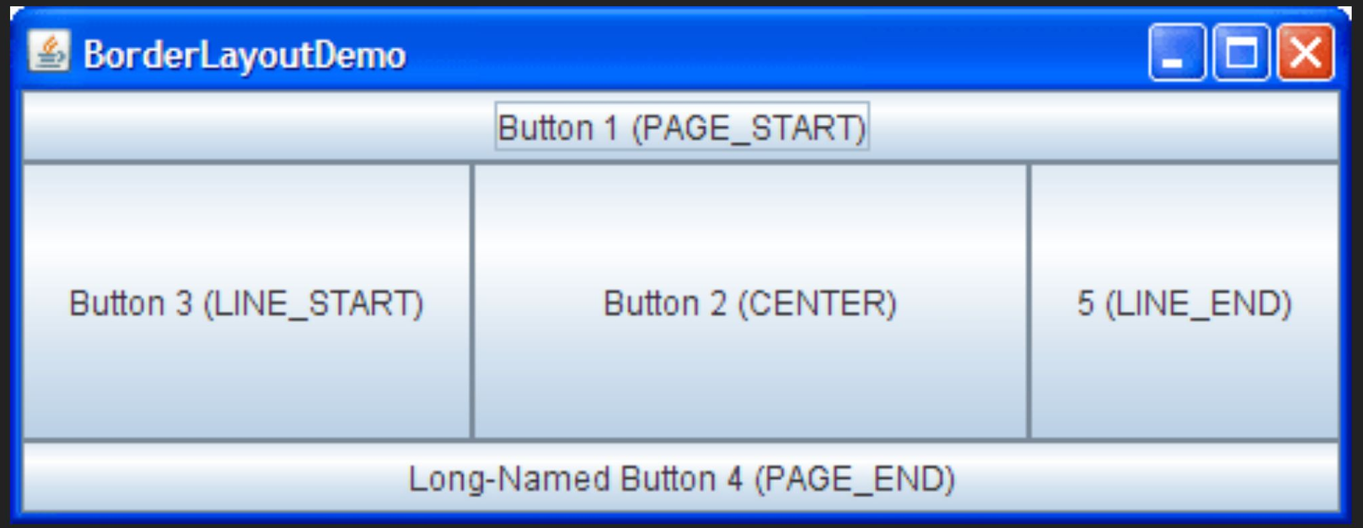
* Anordnung von links nach rechts, ggf. mit Umbruch




https://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html

# Layout Manager - BorderLayout

* 5 Bereiche: oben, unten, links, rechts, mitte
* Beim Vergrößern wird die Mitte gestreckt
* Es ist möglich, nur einige Bereiche zu nutzen




https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html

# Layout Manager - BorderLayout

* Beim Hinzufügen von Komponenten wird festgelegt, wo sie platziert werden

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

frame.add(new JLabel("Oben"), BorderLayout.PAGE_START);
frame.add(new JLabel("Unten"), BorderLayout.PAGE_END);
frame.add(new JLabel("Links"), BorderLayout.LINE_START);
frame.add(new JLabel("Rechts"), BorderLayout.LINE_END);
frame.add(new JLabel("Ich bin in der Mitte!"), BorderLayout.CENTER);
```

https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html

# Layout Manager - GridLayout

* Gitter mit n*m Zellen
* Eine Komponente pro Zelle
* Komponente nimmt immer gesamte Zelle ein
* Gleichmäßiges Strecken aller Komponenten

https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html

# Layout Manager - GridLayout

* Komponenten werden in der Einfügereihenfolge im Gitter platziert

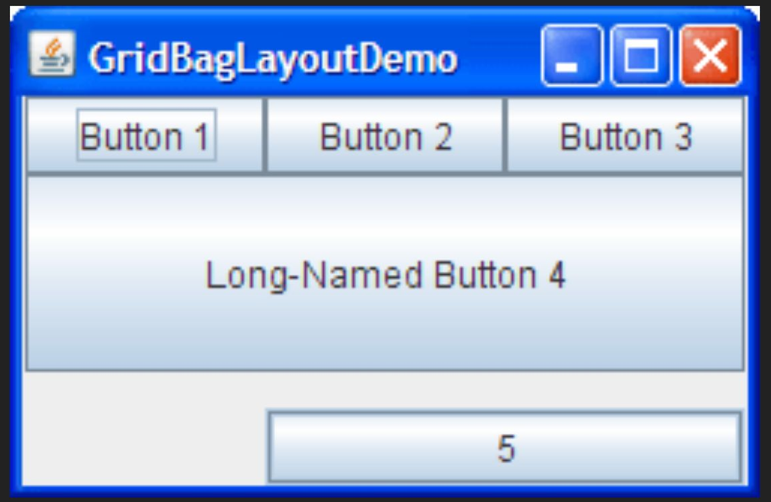
```
// 2*2 Gitter
frame.setLayout(new GridLayout(2,2));

frame.add(new JLabel("Hello World"));
frame.add(new JLabel("Grid 2"));
frame.add(new JLabel("Nochn Grid-Item"));
frame.add(new JLabel("Weiteres Grid"));
```

https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html

# Layout Manager - GridBagLayout

* Mächtigster, flexibelster LayoutManager
* Gitter mit n*m Elementen
* UI-Komponenten können mehr als eine Zelle besetzen
* Granulare Kontrolle über Skalierverhalten



https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

# Verschachtelte Container

# Verschachtelte Container

* Es ist sehr unpraktisch, alle UI-Elemente auf einer Ebene in einen Container zu packen
* Sinnvoll: Kleine, wiederverwendbare UI-Einheiten
* Die Klasse JPanel dient zur Gruppierung von Elementen
* Layoutmanager und Komponenten werden dem Panel zugewiesen
* Layoutmanager wird direkt im Konstruktor übergeben
* über die add()-Methode werden Komponenten zugeordnet

# Verschachtelte Container

* Beispiel: 2 Container einfügen, die restliche Elemente beinhalten

```
JFrame frame = new JFrame("Using two panels");
frame.setLayout(new BorderLayout());

JPanel panel1 = new JPanel();
JPanel panel2 = new JPanel();

panel1.add(new JLabel("links"));
panel2.add(new JLabel("rechts"));

frame.add(panel1, BorderLayout.LINE_START);
frame.add(panel2, BorderLayout.LINE_END);

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

# Verschachtelte Container

* Eigene UI-Komponenten als Kinder von JPanel

```
public class TwoByTwoLabelPanel extends JPanel {
    public TwoByTwoLabelPanel(String text1, String text2, String text3, String text4){
        super(new GridLayout(2,2));
        this.add(new JLabel(text1, SwingConstants.CENTER));
        this.add(new JLabel(text2, SwingConstants.CENTER));
        this.add(new JLabel(text3, SwingConstants.CENTER));
        this.add(new JLabel(text4, SwingConstants.CENTER));
    }
}
```

# Verschachtelte Container

* Eigene UI-Komponenten als Kinder von JPanel

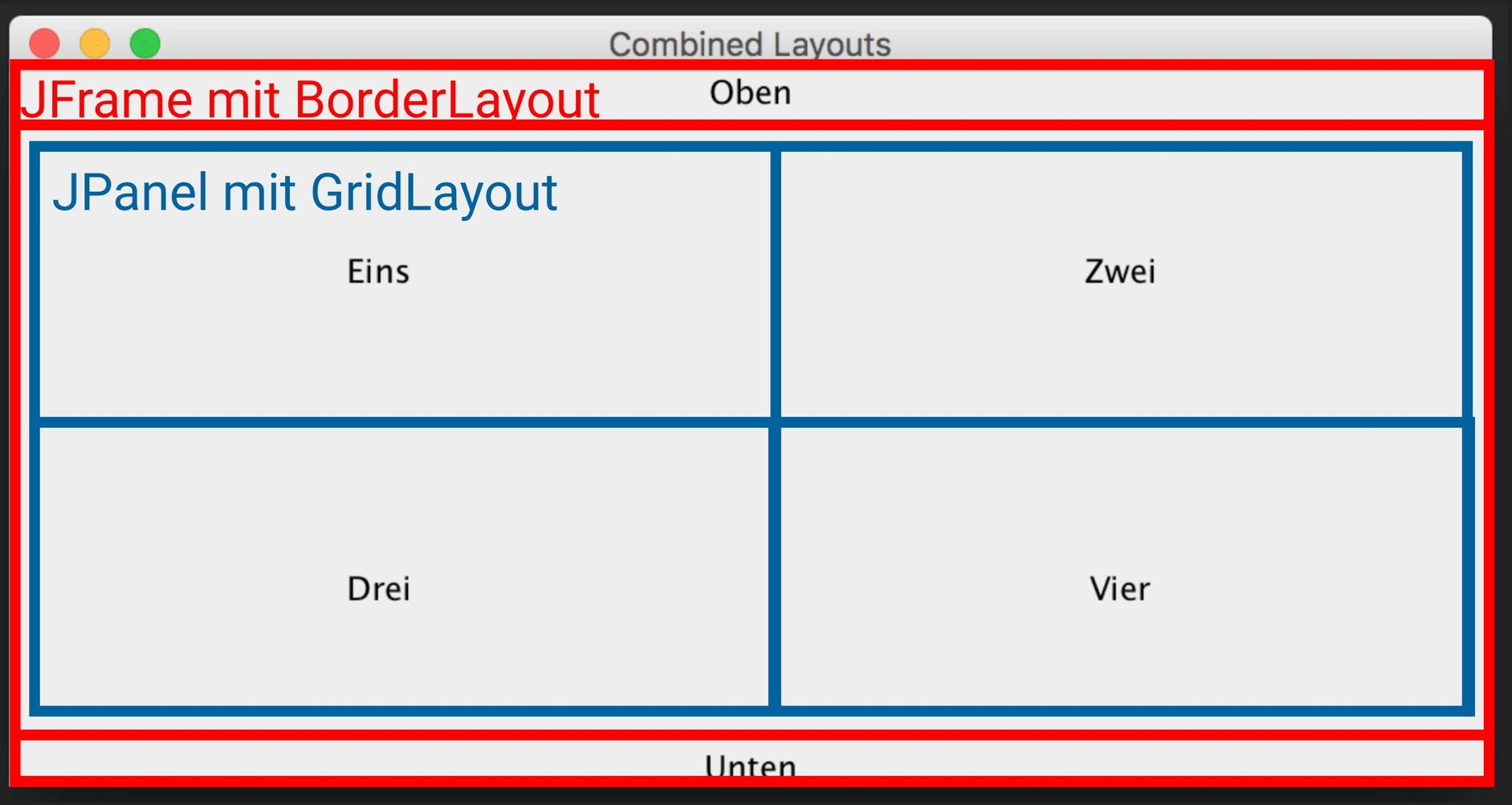
```
JFrame frame = new JFrame("Combined Layouts");
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
frame.setLayout(new BorderLayout());

frame.add(new JLabel("Oben", SwingConstants.CENTER), BorderLayout.PAGE_START);
frame.add(new JLabel("Unten", SwingConstants.CENTER), BorderLayout.PAGE_END);
frame.add(new TwoByTwoLabelPanel("Eins", "Zwei", "Drei", "Vier"), BorderLayout.CENTER);

// Zeige das Fenster an.
frame.pack();
frame.setVisible(true);
```

# Verschachtelte Container

* Eigene UI-Komponenten als Kinder von JPanel



# Coding Time!

# Swing - Codingaufgabe 2

30m

* Erstellt eine eigene Klasse, die von JPanel erbt und ihr eigenes Layout definiert (z.B. GridLayout)
* Gebt der Klasse einen Konstruktor, der mehrere JLabels instantiiert und hinzufügt
* Verwendet die Klasse in einem Programm
* Experimentiert ggf. mit mehreren JPanels innerhalb des Top-Level-Contaners - gerne auch verschachtelt

# Bilder einbinden

# Klasse ImageIcon

* Einfachster Weg um Bilder im GUI anzuzeigen
* Sind **keine** Components - müssen in eine Component platziert werden
* Woher kommen die Bilddateien?

# Klasse ImageIcon

* Option 1: Bild direkt von einem bekannten Pfad laden

```
// Create and set up the window.
JFrame frame = new JFrame("ImageExample1");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

ImageIcon icon = new ImageIcon(
  "/Users/bastian/Pictures/sunglasses.png",
  "Sonnenbrillen-Emoji");

// We cannot add an ImageIcon directly to a Frame, so we need a JLabel as proxy.
frame.add(new JLabel(icon, SwingConstants.CENTER));

// Display the window.
frame.pack();
frame.setVisible(true);
```

# Klasse ImageIcon

* Funktioniert!

* Nachteil: Pfad muss bekannt sein -> "works on my machine"
* Kann man z.B. mit Umgebungsvariablen umgehen, die auf ein Base-Directory zeigen, von dem aus der Pfad konstruiert wird

# Klasse ImageIcon

* Option 2: Bild zum Code legen und vom Classloader laden lassen
* Bild liegt in diesem Beispiel direkt im gleichen Verzeichnis

# Laden von Bildern via Classloader

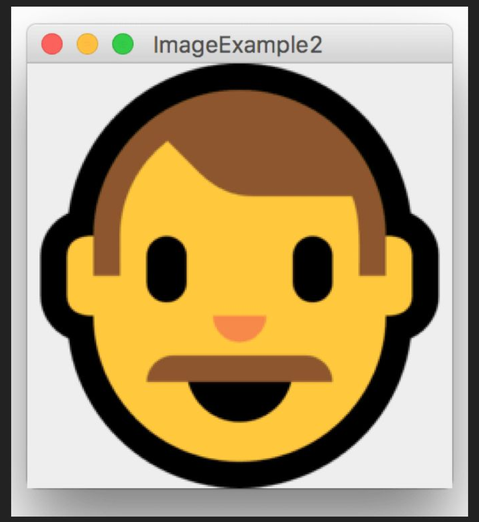
* Option 2: Bild vom Classloader laden lassen

```
// URL zum Bild vom Classloader laden lassen
URL imgURL = ImageExample2.class.getResource("man.png");
ImageIcon icon = new ImageIcon(imgURL, "Mann-Emoji");
```

* Rest wie in Beispiel 1

# Laden von Bildern via Classloader

* Funktioniert!




* Vorteil: Unabhängig von der Plattform / Gerät - Bild liegt direkt beim Code und wird von der JVM geladen
* Nachteil: Gehören Bilder wirklich ins gleiche Verzeichnis wie Code?

# Exkurs: Class Loader

* Was passiert hier genau?

```
URL imgURL = ImageExample2.class.getResource("man.png");
```

* `Klassenname.class` gibt ein Objekt der Klasse **java.lang.Class** zurück. Dessen Methode getResource() wird aufgerufen:

```
Class<ImageExample2> c = ImageExample2.class;
URL imgURL = c.getResource("man.png");
```

https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

# Exkurs: Class Loader

* `Class.getResource(string s)` delegiert an ein Objekt der Klasse **`java.lang.ClassLoader`** und ruft dessen Methode `getSystemResource()` auf.
* ClassLoader-Objekte sind dafür zuständig, Java-Klassen zu laden
* Erst dadurch werden sie in der JVM verfügbar
* ClassLoader können auch sog. Ressourcen (z.B. Bilder) laden
* Der ClassLoader sucht nach Klassendateien im **Class Path**

https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

# Exkurs: Class Loader

* Der Class Path ist das Verzeichnis, welches vom Class Loader nach Java-Klassen durchsucht wird
* Es kann mehrere Class Paths für eine JVM geben
* Beim Starten von Programmen aus der IDE wird der Class Path von der IDE korrekt gesetzt.
* Wenn man Java über das CLI ausführt, muss das manuell passieren

https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

# Exkurs: Class Loader

* Klasse auf der Console compilieren:

```
javac ImageExample2.java
```

* Einfach starten reicht nicht:

```
java gui.components.images.ImageExample2

Error: Could not find or load main class gui.components.images.ImageExample2
Caused by: java.lang.ClassNotFoundException: gui.components.images.ImageExample2
```

https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

# Exkurs: Class Loader

* Class Path richtig setzen:

```
java -cp "/Users/bastian/src/vl/repo/programmierbeispiele/src"
  gui.components.images.ImageExample2
```

* Mehrere Class Paths können per : getrennt werden:

```
java -cp "/Users/bastian/src/vl/repo/programmierbeispiele/src:/Users/bastian/hier/liegt/auch/code"
  gui.components.images.ImageExample2
```

https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

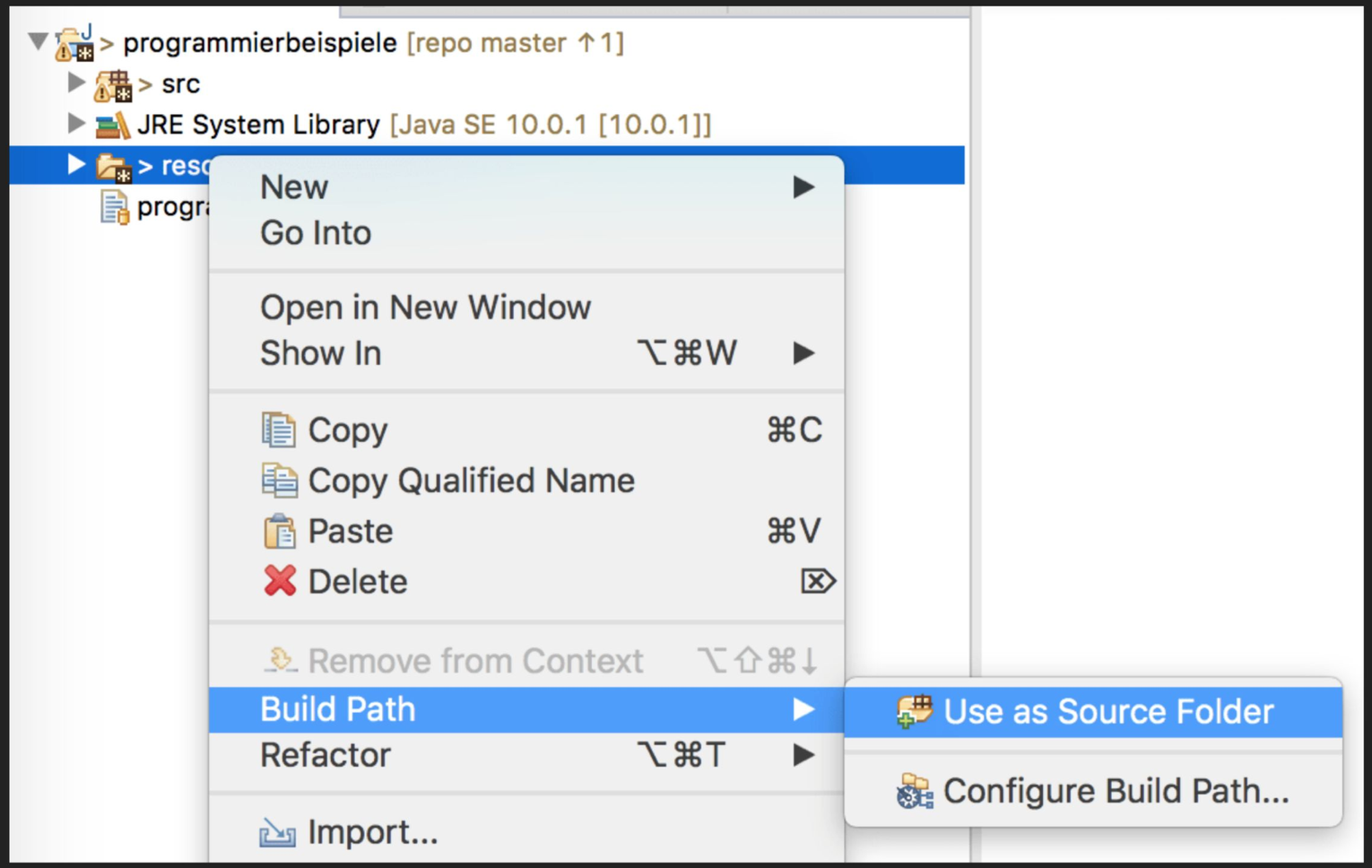
# Laden von Bildern via Classloader

* Option 3: Bild von ClassLoader laden lassen - saubere Variante
* Verzeichnis "resources" auf der obersten Ebene eures Projektes anlegen (sollte im gleichen Vz. wie "src" sein).

https://docs.oracle.com/javase/tutorial/uiswing/components/icon.html#getresource

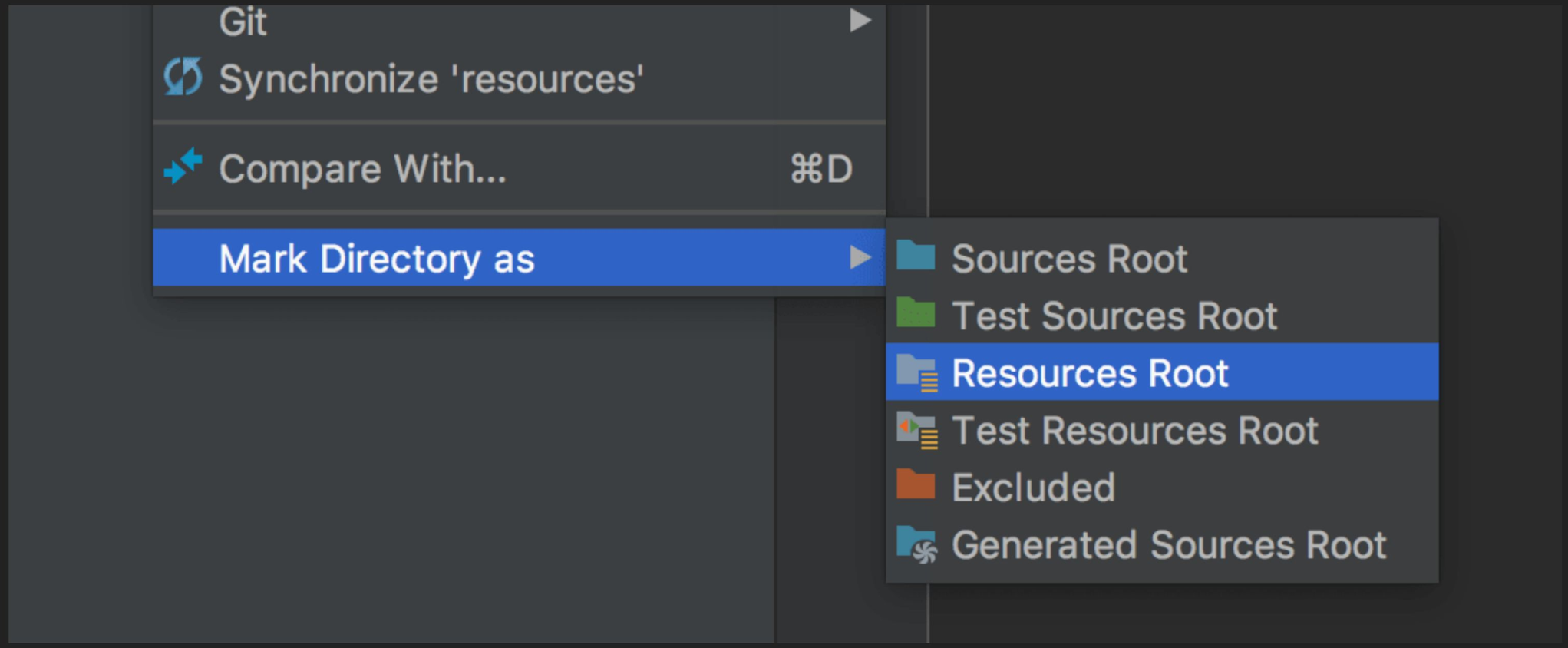
# Laden von Bildern via Classloader

* Der IDE sagen, dass auch dieses Vz. auf dem Class Path sein soll:



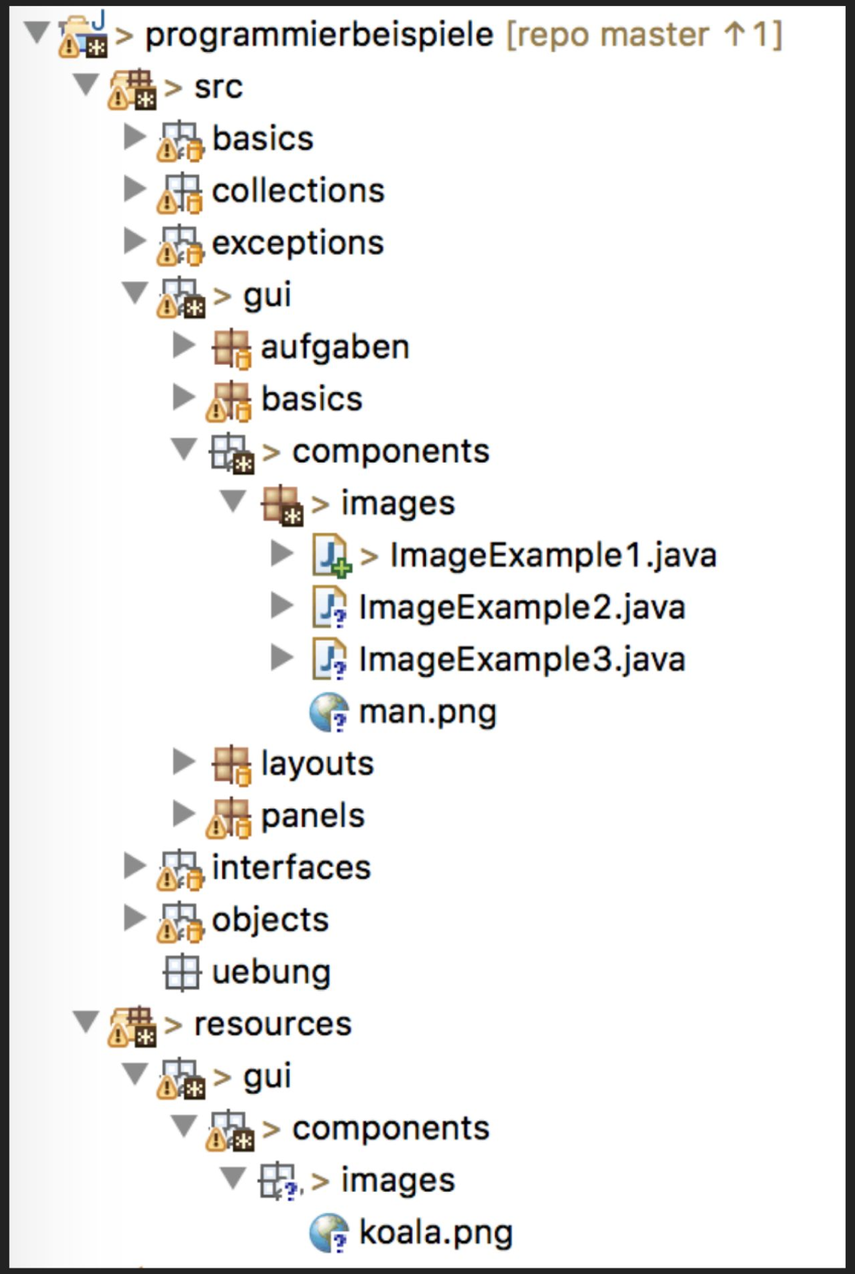
# Laden von Bildern via Classloader

* In anderen IDEs kann das anders heißen, hier z.B. IntelliJ:



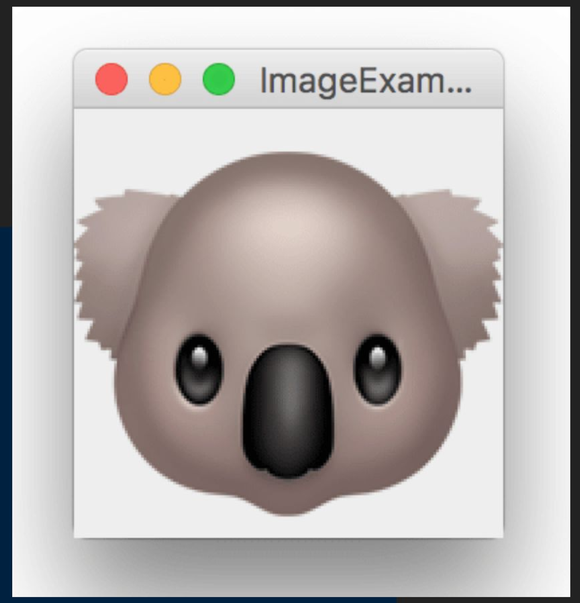
# Laden von Bildern via Classloader

* Dann den Package-Pfad im Verzeichnis "resources" nochmal nachbauen.
* Jetzt sind Bilder und Code relativ sauber getrennt.
* In der "echten Welt" würde man Bilder z.B. in ein separates JAR-File packen und von dort laden



# Laden von Bildern via Classloader

* Jetzt funktioniert dieser Code:




```
// Create and set up the window.
JFrame frame = new JFrame("ImageExample3");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// URL zum Bild vom Classloader laden lassen
URL imgURL = ImageExample3.class.getResource("koala.png");
ImageIcon icon = new ImageIcon(imgURL, "Koala!");

// We cannot add an ImageIcon directly to a Frame, so we need a JLabel as proxy.
frame.add(new JLabel(icon, SwingConstants.CENTER));

// Display the window.
frame.pack();
frame.setVisible(true);
```

# Event Listener

# Text-Komponenten

# Menüs

# Model-View-Controller