

# 2D-Graphik & Frameworks

## Teil 1: AWT / Swing

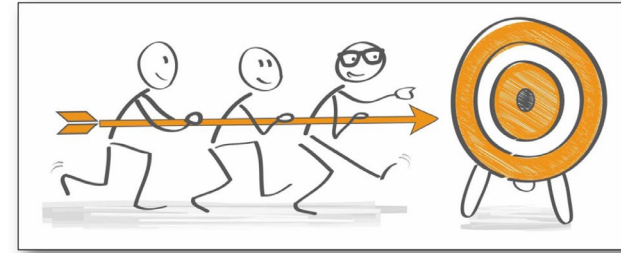
**Björn Näf**  
Dozent

[bjoern.naef@edu.teko.ch](mailto:bjoern.naef@edu.teko.ch)



# Ziele: 2D-Graphik & Frameworks

8 – 10 Lektionen



## ▲ Lernfelder / Lerninhalte

- ▲ Linien, Rechtecke, Kreise, Ellipsen, Kreisbogen, Polygone, Graphische Anwendungen
- ▲ Image-, Shape-Komponenten, Standarddialoge
- ▲ Frameworks (J2EE / JEE 6, OSGi, Maven, Hudson, JSF, EJB, JPA, Eclipse, Swing)

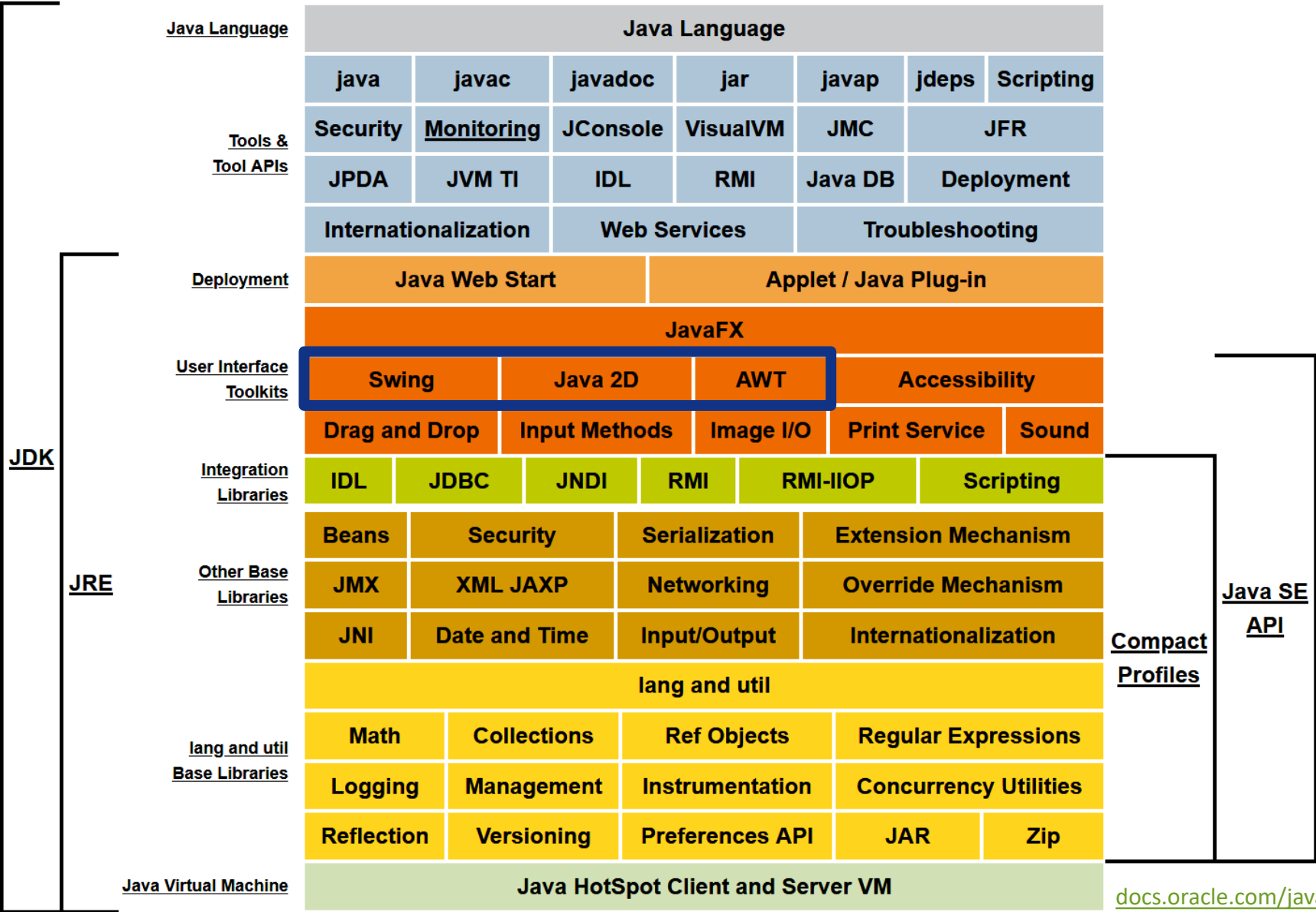
## ▲ Lernziele

- ▲ Sie kennen die Grafikelemente und können einfache Grafikprogramme erstellen.
- ▲ Sie können Image- und Shape-Komponenten, sowie Standarddialoge praxisgerecht einsetzen.
- ▲ Sie kennen bekannte Java-Frameworks und können einige in Praxisbeispielen anwenden

The background features a complex arrangement of red wireframe geometric shapes, including cylinders, rectangular prisms, and cubes, some of which are nested or overlapping. These shapes are rendered in a minimalist, skeletal style against a solid black background. A semi-transparent red horizontal band spans the lower portion of the image, serving as a backdrop for the title text.

# **AWT / Swing – Einfache Grafikanwendungen gestalten**

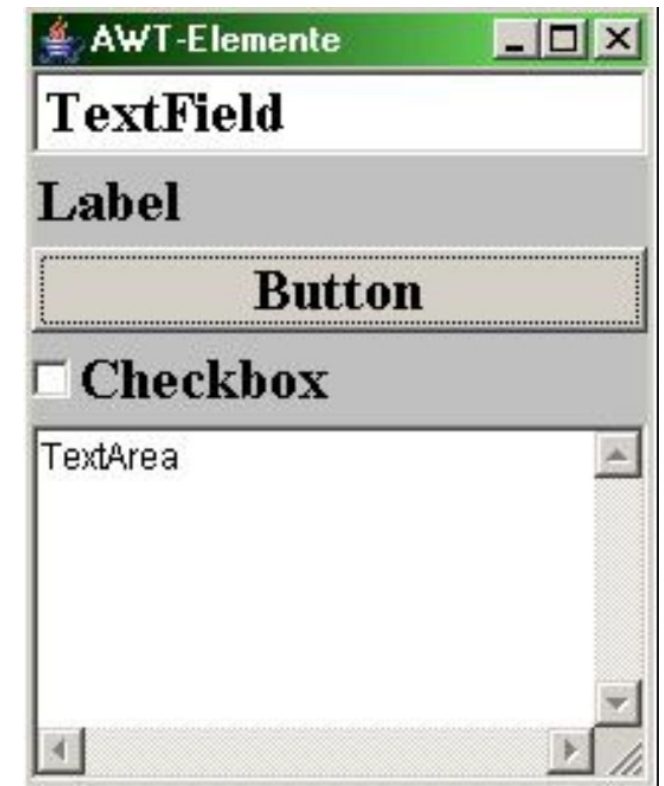
# Swing: Einordnung im Java / JDK Framework



# Abstract Windowing Toolkit (AWT)

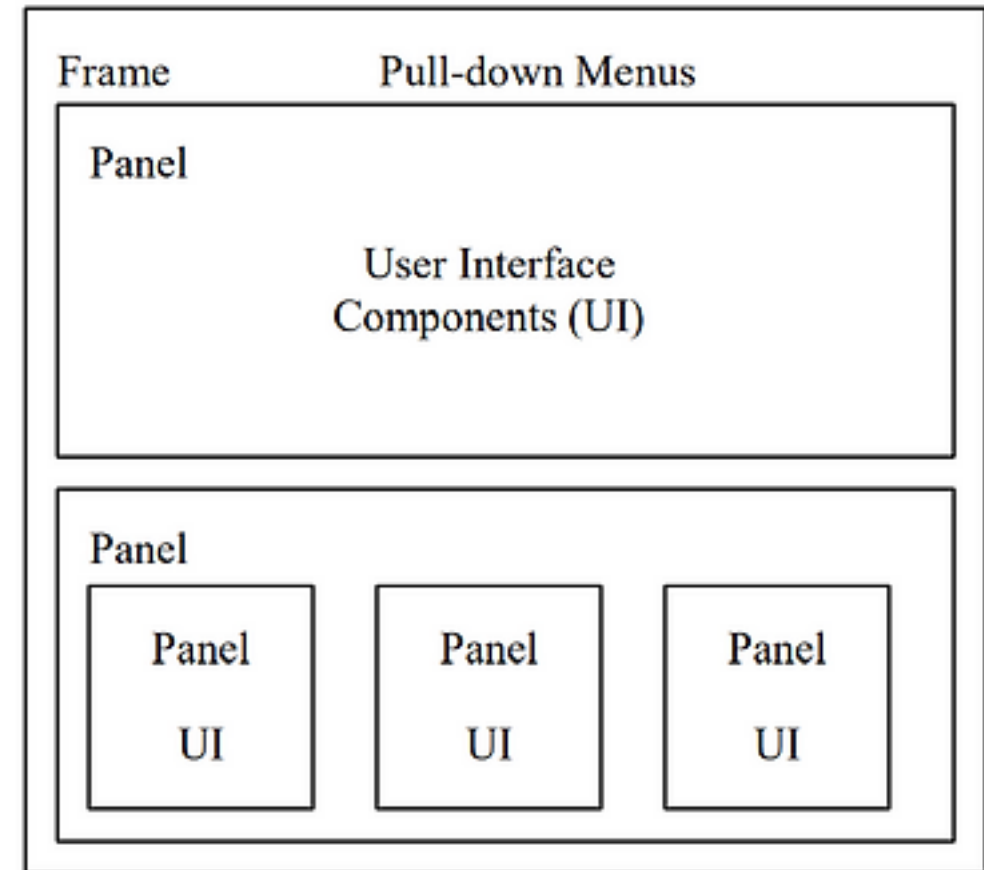
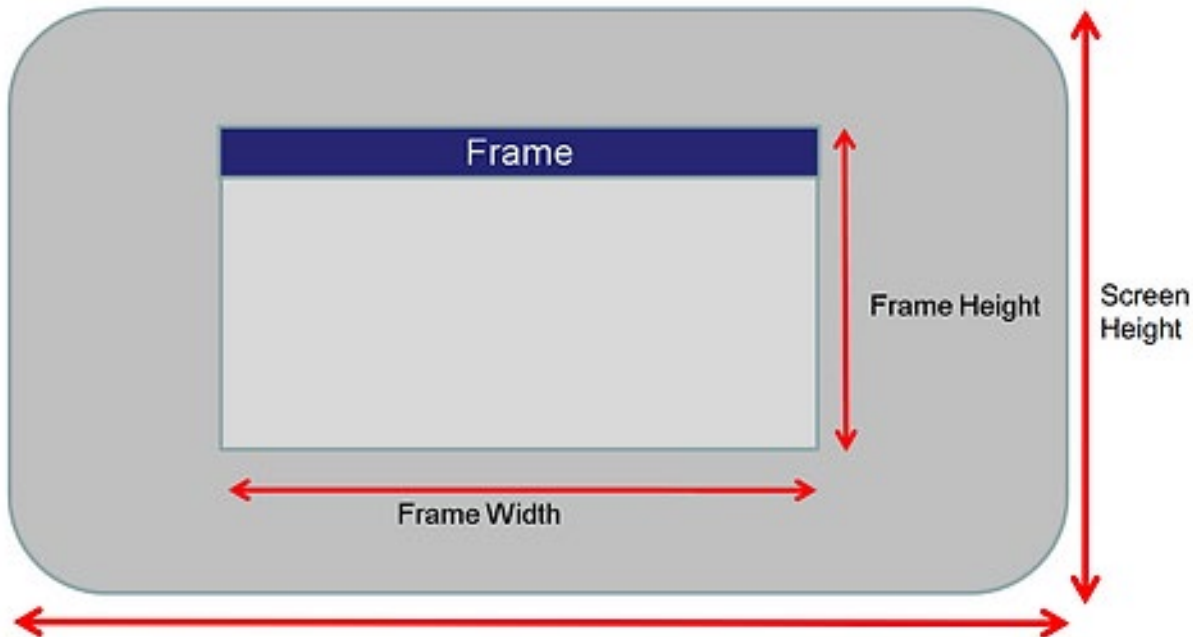
## Basis für die meisten Grafik-Anwendungen im Java-Framework

- ▲ 1995 wurde mit der ersten Java-Version eine integrierte Programmbibliothek für GUI-Anwendungen ausgeliefert
- ▲ AWT setzte auf den GUI-Elementen der damaligen Betriebssystemen Windows, Unix / Linux und Mac OS-X auf
- ▲ Die Grafik- und Steuerelemente wurden „Widgets“ genannt
- ▲ Mit Java 2 (JDK 1.2) erweiterte Swing 1998 AWT massiv
- ▲ Neu wurden die GUI-Elemente von Java selber gezeichnet, diese sind AWT-Erweiterungen, ihre Namen beginnen mit „J“
- ▲ Es entstand das Plattform-übergreifende «Java Look and Feel»
- ▲ Da Swing auf AWT aufsetzt, sind beide heute noch Teil des JDK und können (mit Vorsicht) sogar gemischt eingesetzt werden.



# AWT / Swing: Layout-Konzept (1)

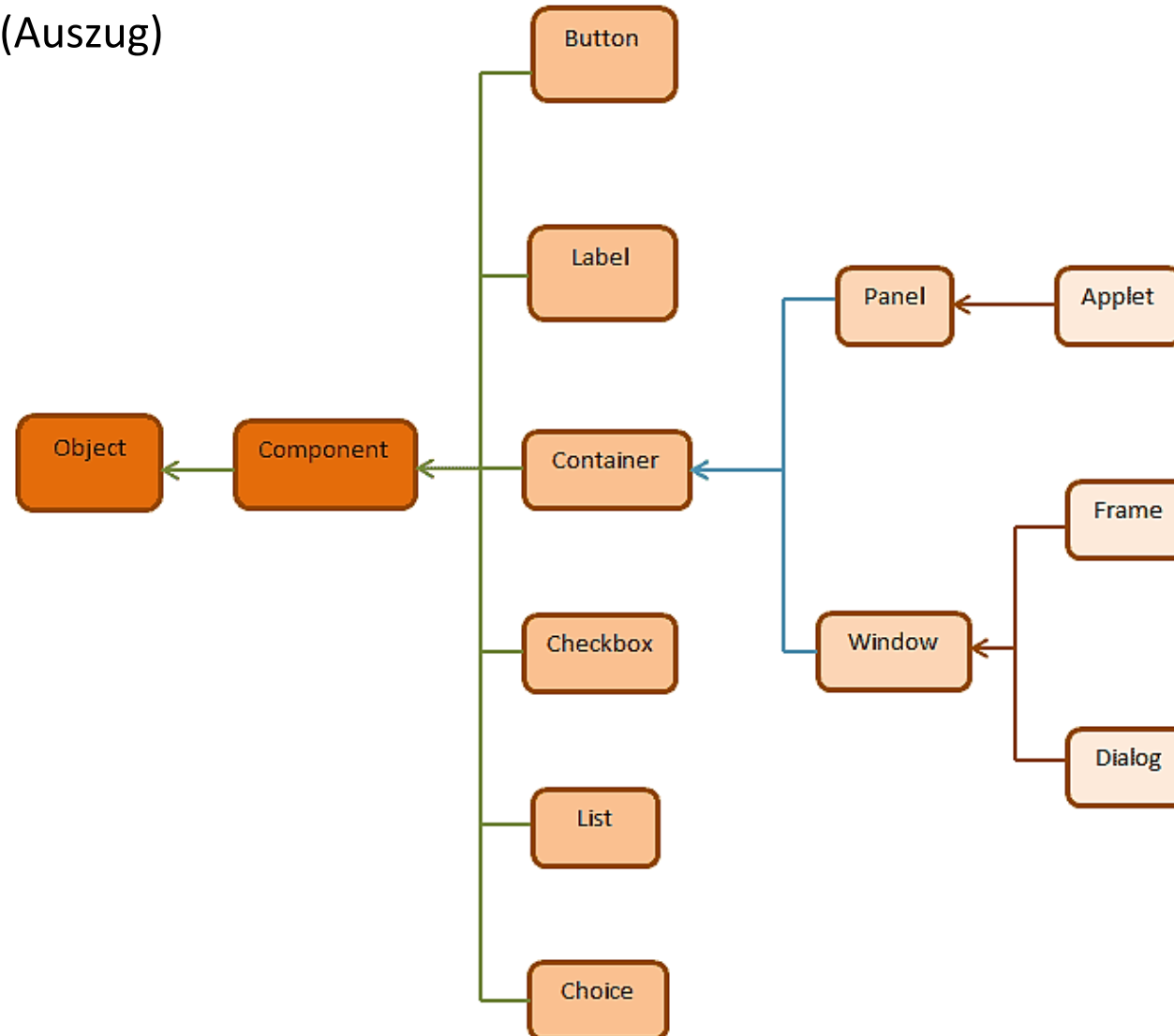
Die UI-Elemente sind hierarchisch gegliedert – Hauptkomponente ist das „Frame“-Objekt



# AWT / Swing: Layout-Konzept (2)

Die UI-Elemente sind hierarchisch gegliedert – es existieren verschiedene Typen

AWT-Klassenhierarchie (Auszug)

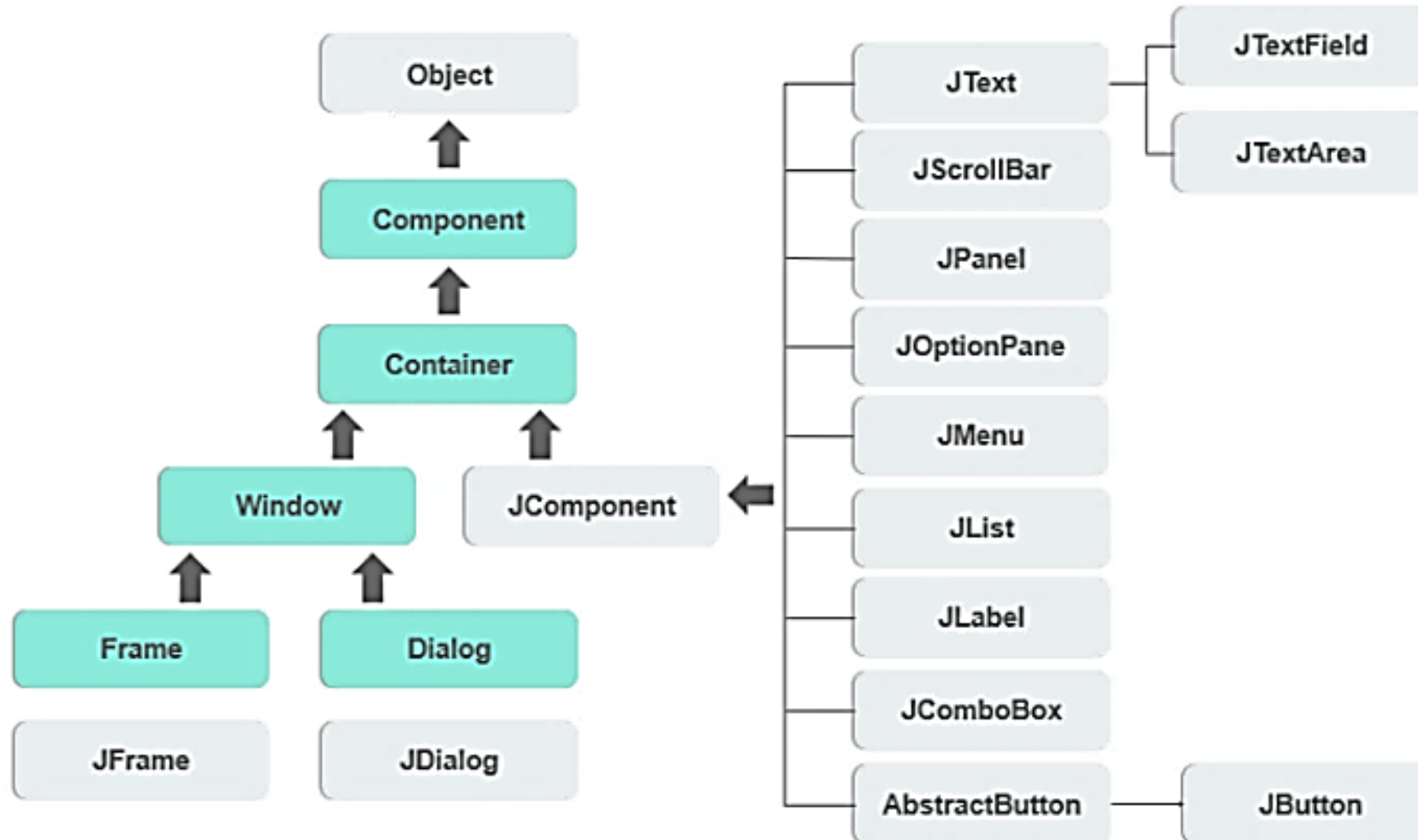




# AWT / Swing: Layout-Konzept (3)

Die UI-Elemente sind hierarchisch gegliedert – es existieren verschiedene Typen

Swing-Klassenhierarchie (Auszug)





# AWT / Swing: Ein erstes Beispiel...

```
import javax.swing.JFrame

fun main() {
    val frame = JFrame("Test Frame")
    frame.setSize(400, 300)
    frame.isVisible = true
    frame.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
}
```

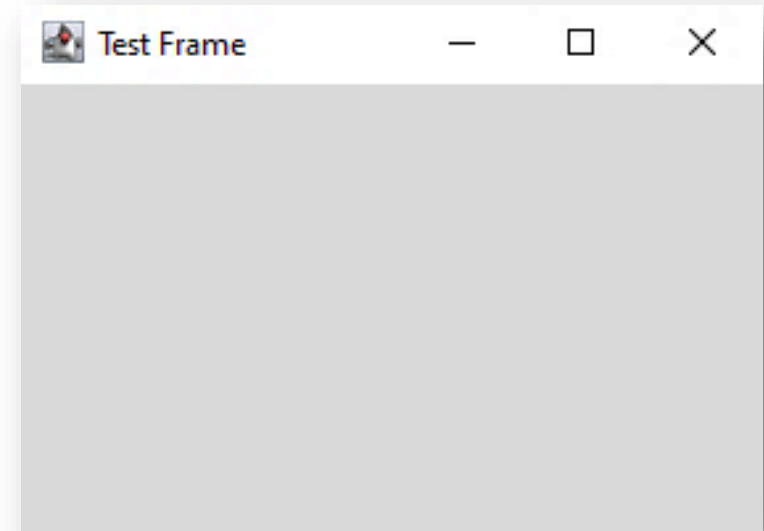
1.

2.

3.

4.

5.



1. Swing-Bibliothek importieren (=verfügbar machen)
2. Frame-Instanz erzeugen (Fenstertitel als Parameter)
3. Frame-Abmessungen festlegen
4. Frame anzeigen
5. Operation für das Schliessen des Fensters bestimmen

# Vorbereitung für Grafik: Eigene Frame-Klasse erstellen

```
import javax.swing.JFrame

fun main() {
    val frame = MyFrame("Test Frame") 6.
    frame.setSize(400, 300)
    frame.isVisible = true
    frame.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
}
```

```
import ...

class MyFrame(title: String) : JFrame() {

    override fun paint(canvas: Graphics?) {
        super.paint(canvas)

        canvas as Graphics2D
    }
}
```

1.  
2.  
3.  
4.  
5.

1. Swing- / AWT-Bibliotheken importieren
2. Klasse definieren, die von „JFrame“ ableitet (vererbt)
3. Methode „paint“ überschreiben (Keyword „override“)
4. Methode „paint“ der Mutterklasse aufrufen („super“)
5. Datentyp des Parameters „canvas“ als „Graphics2D“ festlegen
6. In der main-Methode des Programms Frame-Instanz erzeugen

Wir beginnen im  
Folgenden HIER

# Vorbereitung für Grafik: Benötigte Libraries / Imports

Liste kann variieren und erhebt keinen Vollständigkeitsanspruch

```
import java.awt.BasicStroke
import java.awt.Color
import java.awt.Graphics
import java.awt.Dimension
import java.awt.Graphics2D
import java.awt.geom.Path2D
import java.io.File
import javax.imageio.ImageIO
import javax.swing.*
import kotlin.system.exitProcess
```

# Grafikformen (1)

## Linie, Strichfarbe, Rechteck, Kreis, Ellipse, Kreisbogen

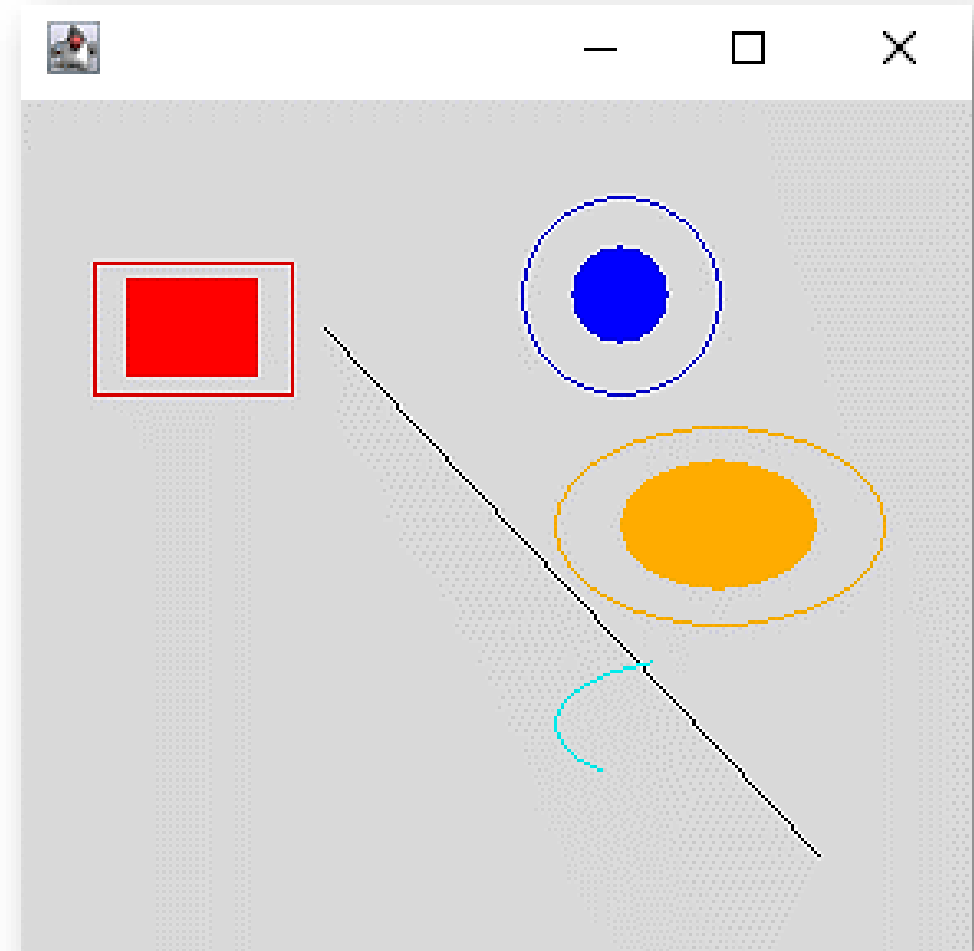
```
// Linie
canvas.drawLine(100, 100, 250, 260)

// Rechteck
canvas.color = Color.RED;
canvas.drawRect(30, 80, 60, 40)
canvas.fillRect(40, 85, 40, 30)

// Kreis
canvas.color = Color.BLUE;
canvas.drawArc(160, 60, 60, 60, 0, 360)
canvas.fillArc(175, 75, 30, 30, 0, 360)

// Ellipse
canvas.color = Color.ORANGE;
canvas.drawArc(170, 130, 100, 60, 0, 360)
canvas.fillArc(190, 140, 60, 40, 0, 360)

// Kreisbogen
canvas.color = Color.CYAN;
canvas.drawArc(170, 200, 90, 40, 110, 120)
```



# Grafikformen (2)

## Polygone

```
// Polygon - beliebig
canvas.color = Color.GREEN;
val xPoly = intArrayOf(150, 250, 325, 375, 450, 275, 100)
val yPoly = intArrayOf(150, 100, 125, 225, 250, 375, 300)
canvas.drawPolygon(xPoly, yPoly, 7)

// Polygon als Shape - gleichseitig (regulär)
canvas.translate(400, 50)
val polygon1 = Utils().generatePolygon(5, 70, true)
canvas.draw(polygon1)

// Polygon als Shape - speziell
canvas.color = Color.PINK;
canvas.translate(65, 72)
var polygon2 = Utils().generatePolygon(5, 50, 30)
canvas.draw(polygon2)
```

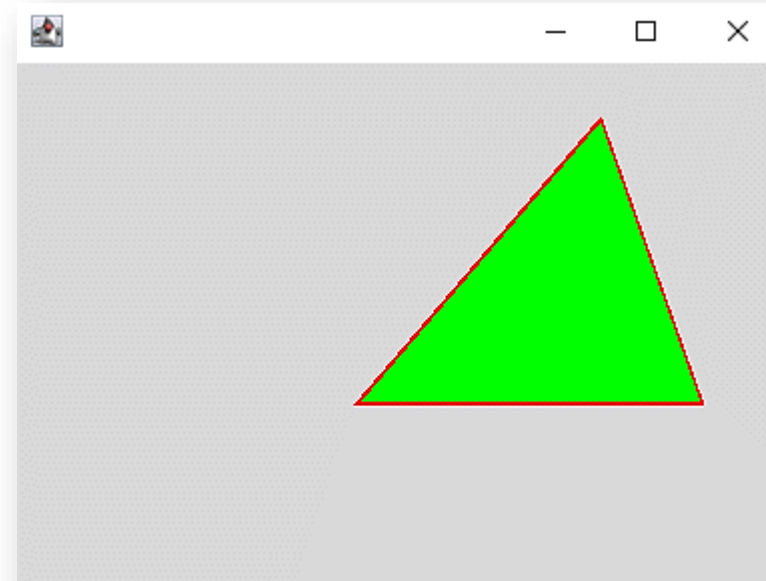


# Grafikformen (3)

## Freier Pfad

```
// Freier Pfad - hier: Dreieck
val path: Path2D = Path2D.Double()
path.moveTo(300.0, 60.0)
path.lineTo(350.0, 200.0)
path.lineTo(180.0, 200.0)
path.closePath()

canvas.setStroke(BasicStroke(3f))
canvas.setColor(Color.RED)
canvas.draw(path)
canvas.setColor(Color.GREEN)
canvas.fill(path)
```

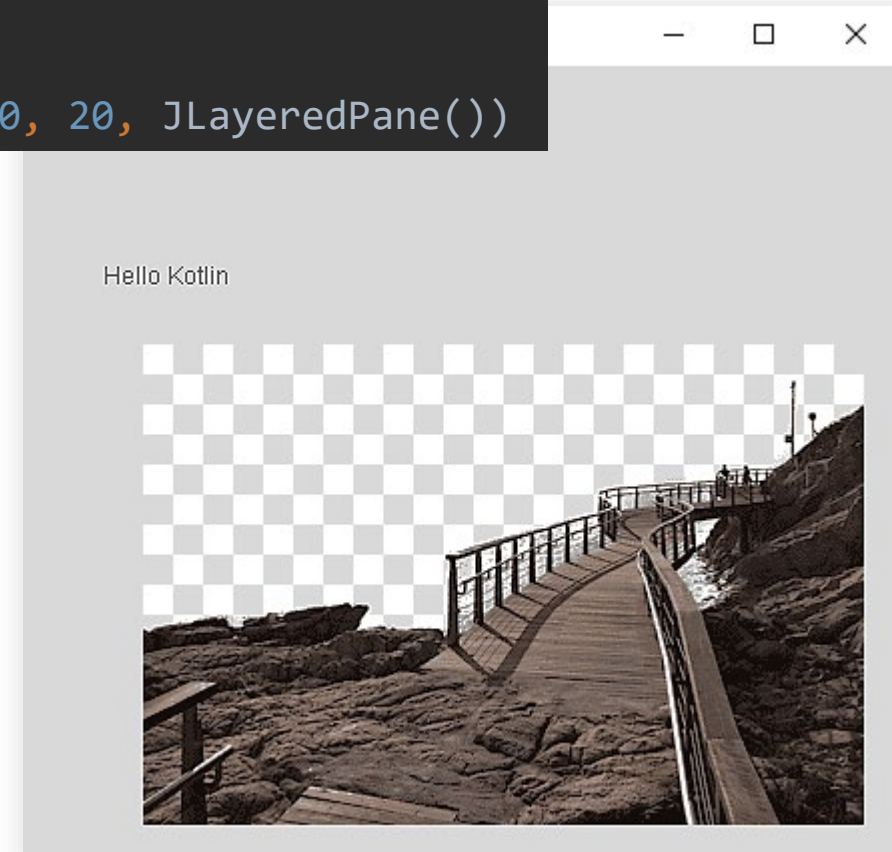


# Grafikformen (4)

## Text, Bild

```
// Text
canvas.color = Color.DARK_GRAY
canvas.drawString("Hello Kotlin", 50, 140)

// Bild
canvas.translate(50, 150)
canvas.drawImage(ImageIO.read(File("beach_road.png")), 20, 20, JLayeredPane())
```

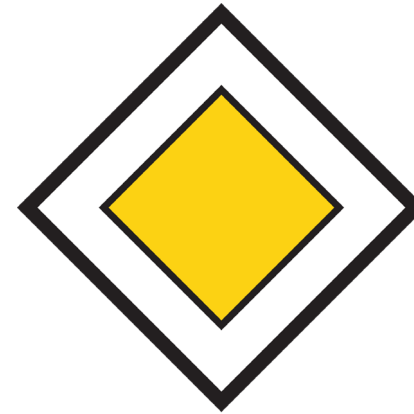




# Aufgabe 4.1

## « Einfache Grafikbeispiele »

Versuchen Sie folgende Verkehrsschilder als Grafik zu programmieren:



Hinweise:










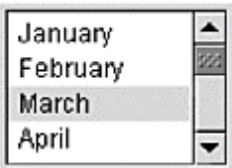
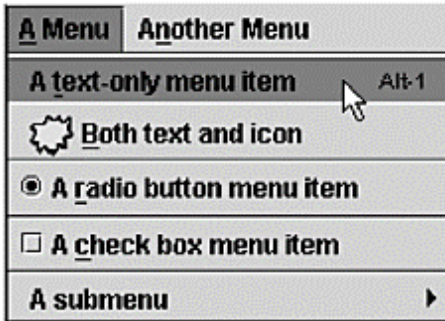

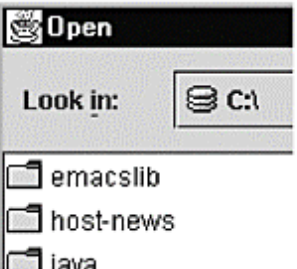


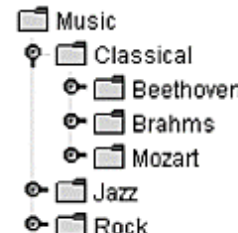

- ▲ Komplexere Formen werden aus mehreren einfachen Formen zusammengesetzt – die Reihenfolge ist dabei wichtig, damit sich die Formen wie gewünscht verdecken
- ▲ Orientieren Sie sich an den gezeigten Code-Bespielen (inkl. Imports)
- ▲ Recherchieren Sie bei Bedarf und/oder diskutieren Sie die Lösung mit Kolleg:innen



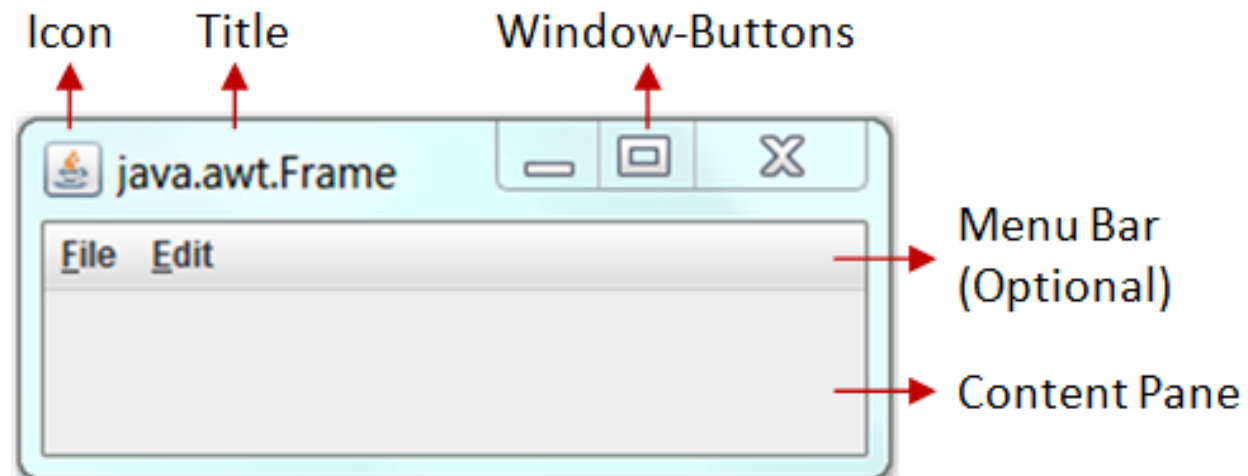
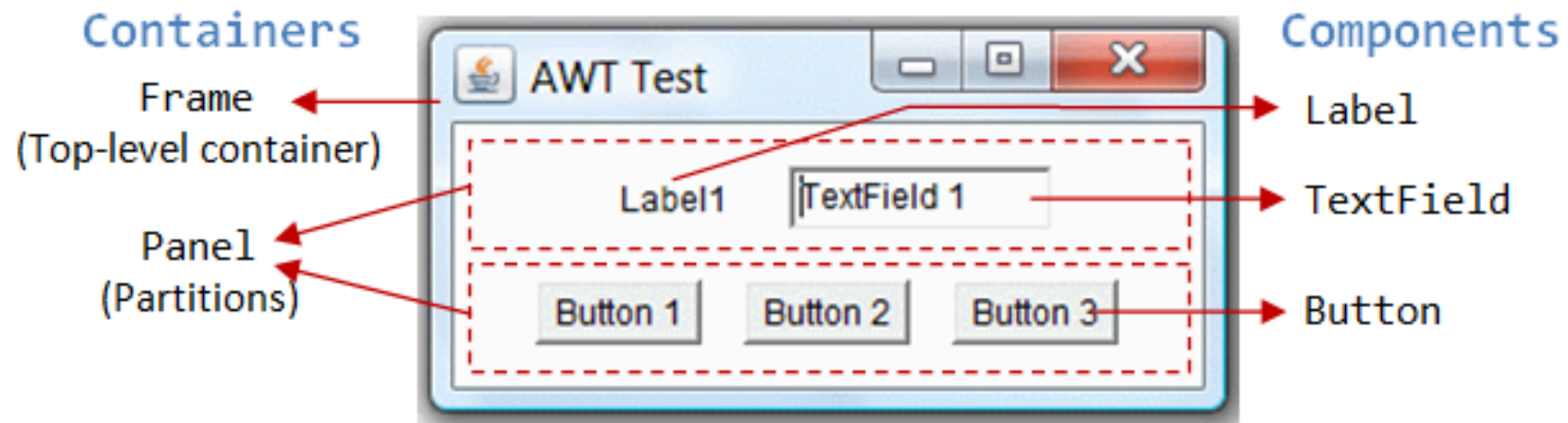
# **AWT / Swing – Benutzeroberfläche – Desktopanwendung**

# Swing Widgets: Steuerelemente für Benutzeroberflächen

AWT und Swing bieten eine Sammlung aus vielen häufig verwendeten UI-Elementen

JButton 	JCheckBox 	JRadioButton 	 														
JTextField 	JSlider 	JToolBar 															
JComboBox 	JList 	JMenuBar, JMenu, JMenuItem 															
JColorChooser 	JFileChooser 	JTable <table border="1" data-bbox="1339 1150 1790 1383"><thead><tr><th>First Name</th><th>Last Name</th><th>Favorite F</th></tr></thead><tbody><tr><td>Jeff</td><td>Dinkins</td><td rowspan="5"></td></tr><tr><td>Ewan</td><td>Dinkins</td></tr><tr><td>Amy</td><td>Fowler</td></tr><tr><td>Hania</td><td>Gajewska</td></tr><tr><td>David</td><td>Gearv</td></tr></tbody></table>	First Name	Last Name	Favorite F	Jeff	Dinkins		Ewan	Dinkins	Amy	Fowler	Hania	Gajewska	David	Gearv	JTree 
First Name	Last Name	Favorite F															
Jeff	Dinkins																
Ewan	Dinkins																
Amy	Fowler																
Hania	Gajewska																
David	Gearv																

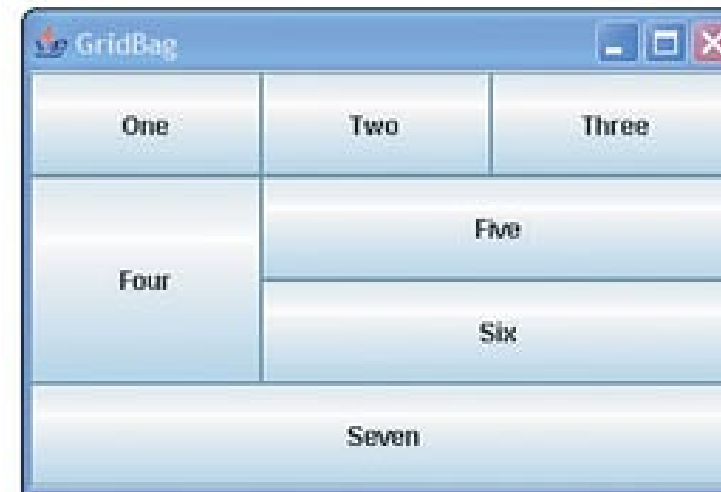
# Aufbau / Grundlayout von Benutzeroberflächen





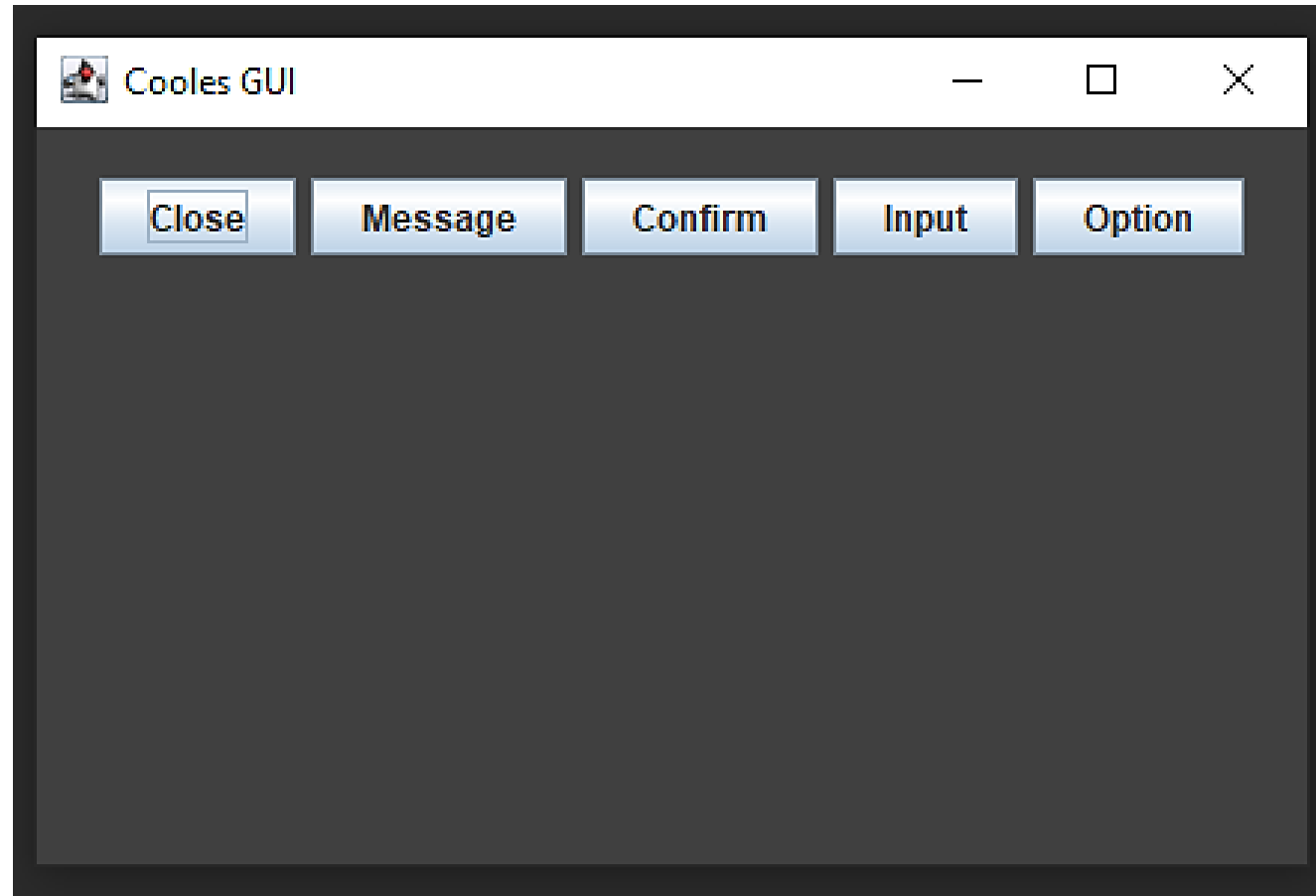
# LayoutManager: Anordnungs-Schablone für UI-Elemente

AWT / Swing bieten verschiedene LayoutManagers für unterschiedliche UI-Gestaltungen



# Schaltflächen und Standard-Dialoge: Ergebnis

Wird im Folgenden erarbeitet...



# Vorbereitung für UI: Eigene Frame-Klasse erstellen

```
import javax.swing.JFrame

fun main() {
    val frame = MyFrame("Test Frame") 4.
    frame.setSize(400, 300)
    frame.isVisible = true
    frame.defaultCloseOperation = JFrame.EXIT_ON_CLOSE
}
```

```
import ...

class MyFrame(title: String) : JFrame() {

    init {

    }

}
```

1. Swing- / AWT-Bibliotheken importieren
2. Klasse definieren, die von „JFrame“ ableitet (vererbt)
3. Methode „init“ implementieren („befüllen“)
4. In der main-Methode des Programms Frame-Instanz erzeugen

Wir beginnen im  
Folgenden HIER



# Vorbereitung für UI: Basiscode & Layoutdefinition

## Basisoptionen: Titel, Hintergrundfarbe, Layout vorbereiten

```
// Hintergrundfarbe, Titel des Fensters setzen  
this.contentPane.background = Color.DARK_GRAY  
setTitle(title)
```

```
// Layout-Panel anlegen und Farbe festlegen  
val card1 = JPanel()  
card1.background = Color.DARK_GRAY
```

```
// UI-Elemente erstellen und konfigurieren  
// ...  
// ...  
// ...  
// ...  
// ...  
// ...  
// ...  
// ...  
// ...  
// ...
```

Wir beginnen im  
Folgenden HIER

```
val gl = GroupLayout(contentPane)  
contentPane.Layout = gl
```

```
gl.setAutoCreateContainerGaps = true
```

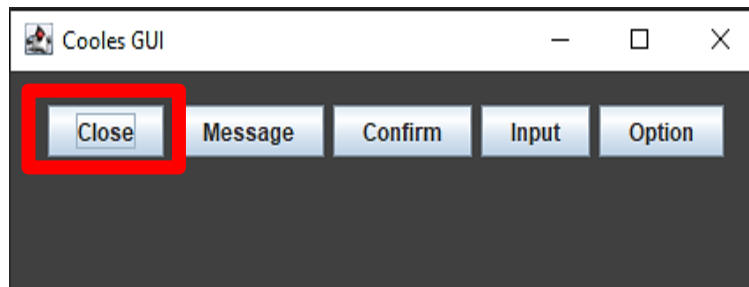
```
gl.setHorizontalGroup(  
    gl.createSequentialGroup()  
        .addComponent(card1)  
    )
```

```
gl.setVerticalGroup(  
    gl.createSequentialGroup()  
        .addComponent(card1)  
    )
```

```
// Restliche Eigenschaften setzen  
defaultCloseOperation = EXIT_ON_CLOSE  
setSize(800, 600)  
setLocationRelativeTo(null)
```

# Schaltflächen zum Schliessen des Fensters / Beenden

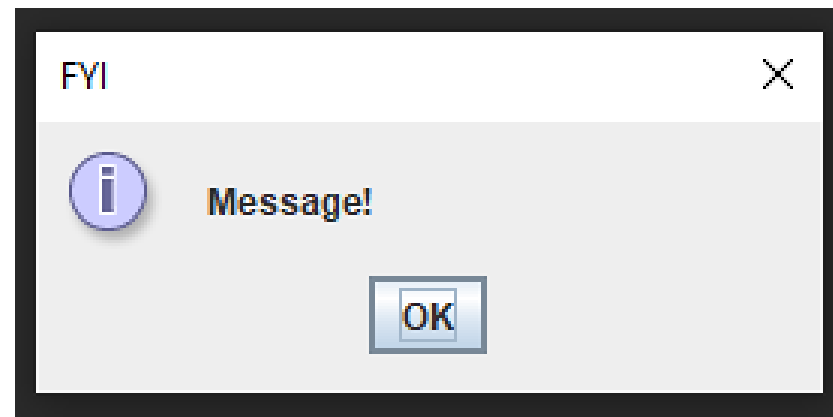
```
// Schaltfläche zum Schliessen des Fensters  
val closeBtn = JButton("Close")  
closeBtn.addActionListener { exitProcess(0) }  
  
card1.add(closeBtn)
```



# Standard-Dialog: Hinweis / Message Box

```
// Hinweise (Message Box)
val messageBtn = JButton("Message")
messageBtn.addActionListener {run {
    JOptionPane.showMessageDialog(null, "Message!", "FYI", JOptionPane.INFORMATION_MESSAGE)
}}

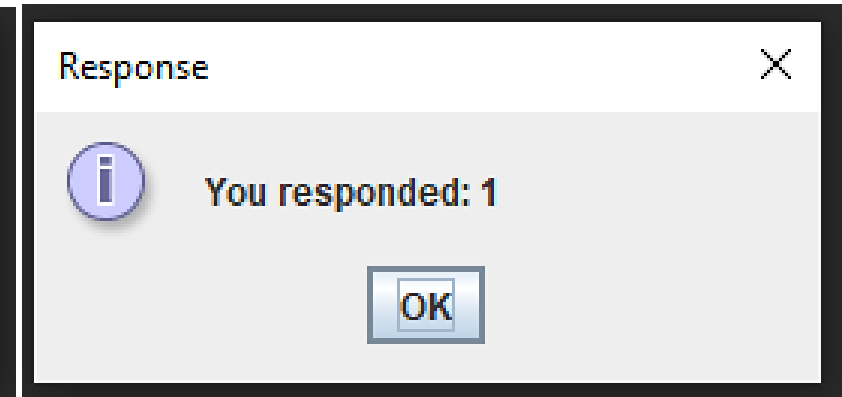
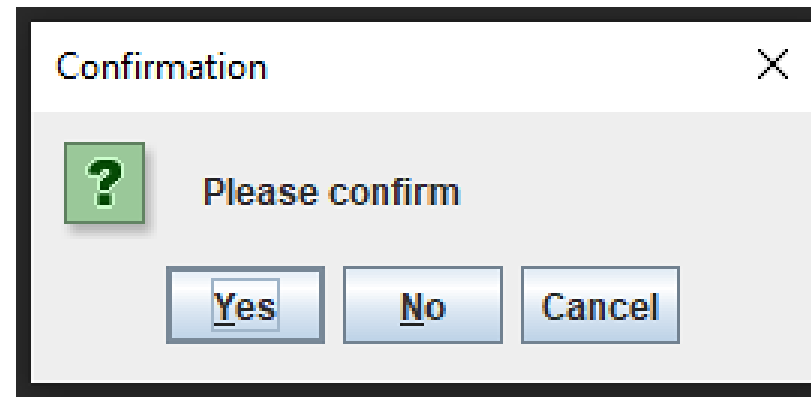
// Button dem Layout hinzufügen
card1.add(messageBtn)
```



# Standard-Dialogs: Bestätigung / Confirm Dialog

```
// Bestätigung (Confirm Dialog)
val confirmBtn = JButton("Confirm")
confirmBtn.addActionListener {run {
    val response = JOptionPane.showConfirmDialog(null, "Please confirm", "Confirmation",
        JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE)
    JOptionPane.showMessageDialog(null, "You responded: " + response, "Response",
        JOptionPane.INFORMATION_MESSAGE)
}}

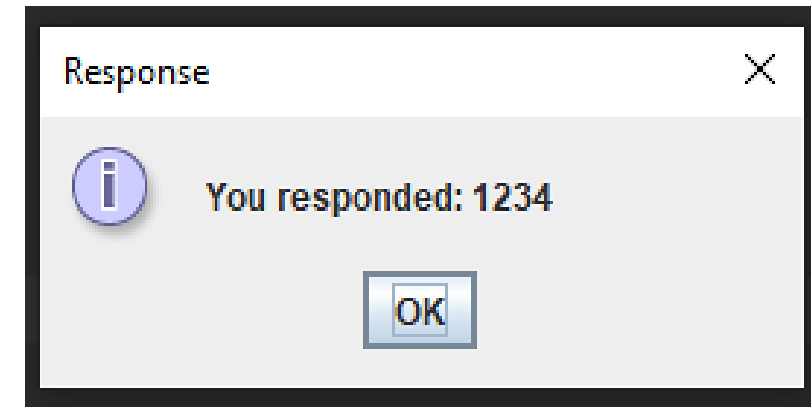
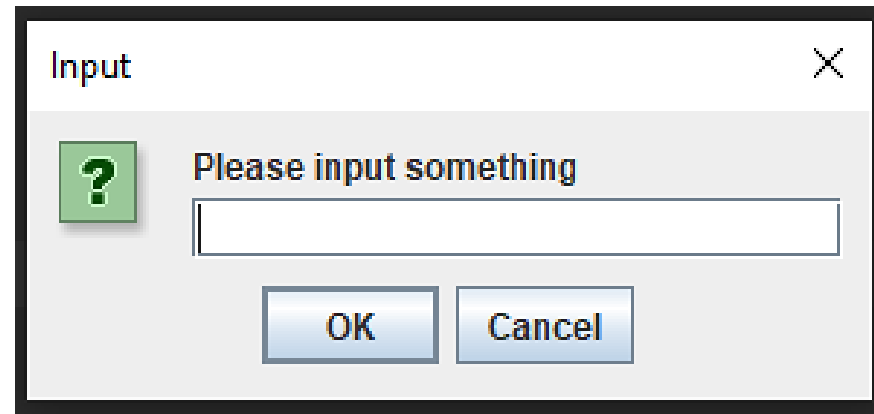
// Button dem Layout hinzufügen
card1.add(confirmBtn)
```



# Standard-Dialoge: Eingabe / Input Dialog

```
// Eingabe (Input Dialog)
val inputBtn = JButton("Input")
inputBtn.addActionListener { run {
    val response = JOptionPane.showInputDialog(null, "Please input something", "Input",
        JOptionPane.QUESTION_MESSAGE)
    JOptionPane.showMessageDialog(null, "You responded: " + response, "Response",
        JOptionPane.INFORMATION_MESSAGE)
}}

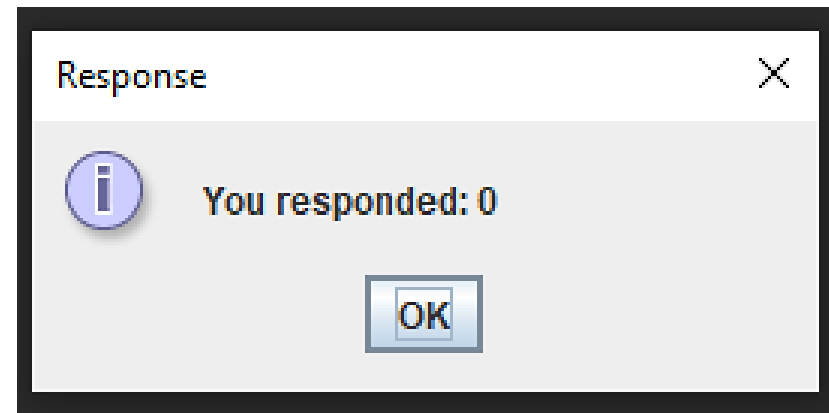
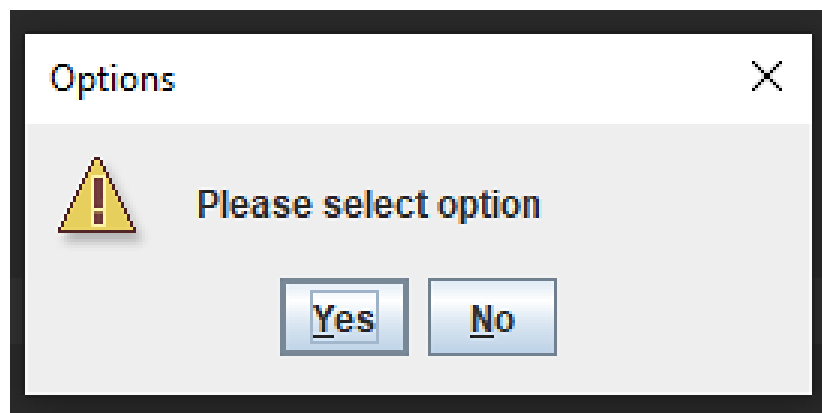
// Button dem Layout hinzufügen
card1.add(inputBtn)
```



# Standard-Dialoge: Auswahl / Option Dialog

```
// Auswahl (Option Dialog)
val optionBtn = JButton("Option")
optionBtn.addActionListener { run {
    val response = JOptionPane.showOptionDialog(null, "Please select option", "Options",
        JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE, null, null, null)
    JOptionPane.showMessageDialog(null, "You responded: " + response, "Response",
        JOptionPane.INFORMATION_MESSAGE)
}}

// Button dem Layout hinzufügen
card1.add(optionBtn)
```



# Aufgabe 4.2

## « Einfache Benutzeroberfläche »

Auf Teams finden Sie das komplette Code-Beispiel « TEKO-GUI-Mustercode.kt »

- ▲ Bringen Sie das Programm in der eigenen Entwicklungsumgebung zum Laufen.
- ▲ Versuchen Sie das komplette Code-Beispiel mit allen UI-Elementen zu verstehen. Machen Sie hierzu einen „Code Walkthrough“ zu zweit.
- ▲ Versuchen Sie das Code-Beispiel wie folgt anzupassen:
  - Wenn beim Confirm-Dialog mit „Ja“ geantwortet wird, dann erscheint der Input-Dialog, andernfalls der Option-Dialog
  - Wenn beim Option-Dialog mit „Ja“ geantwortet wird, dann wird das Programm beendet, andernfalls erscheint der Message-Dialog.
  - Passen Sie die verwendeten Texte und Titel so an, dass Sie dem neuen Verhalten entsprechend Sinn ergeben.