

# *Layout und Panel-Bau in Swing*

Karsten Lentzsch



# *Ziel*

Mit Swing  
einfach, schnell, präzise und einheitlich  
gestalten können.

Lernen, worauf es dabei ankommt.

# *Vorstellung*

- Ich baue Swing-Anwendungen, die viele Leute elegant finden
- biete Bibliotheken, die Swing ergänzen
- arbeite seit 1995 mit emulierten Looks
- helfe Anderen zu gestalten
- und schreibe über Gestaltung mit Java

# *Gliederung*

- Einführung
- Analyse      Was sind die Probleme?
- Ziele        Was wollen wir?
- Konzepte    Welchen Weg gehen wir?
- Lösung      Wie setzen wir das um?
- Ausblick    Was fehlt noch?

# I. Einführung

# Swing-Bausteine



# Swing-Bausteine



# Wir behandeln Formulare

The screenshot displays the 'Skeleton Pro' software interface. The main window is titled 'Example1 - Skeleton Pro' and contains a menu bar with 'File', 'Components', and 'Help'. Below the menu bar is a toolbar with icons for file operations and help. The interface is divided into three main sections:

- Navigator:** A tree view on the left showing the project structure. It includes 'General Project Data', 'Propeller Shaft1' (with 'Segment 1' selected), 'Intermediate Shaft' (with 'Segment 2a', 'Segment 2b', and 'Flange 2a'), and 'Propeller Shaft2' (with 'Segment 3a', 'Segment 3b', 'Segment 3c', and 'Flange 3a').
- Dynamic Help Topics:** A panel at the bottom left with sections for 'Help' (Segment, Diameters, Criteria) and 'Samples' (Sample1).
- Segment 1 Form:** The main configuration area on the right, titled 'Segment 1'. It contains several sections:
  - Segment:** Includes an 'Identifier' field (set to 'Segment 1'), 'PTI [kW]' and 'Power [kW]' fields, and an 'len [mm]' field.
  - Diameters:** Includes 'da [mm]', 'da2 [mm]', and 'R [mm]' fields on the left, and 'di [mm]', 'di2 [mm]', and 'D [mm]' fields on the right.
  - Criteria:** Includes a 'Location' dropdown menu (set to 'Propeller nut thread'), a 'k-factor' field, and a 'Connection' dropdown menu (set to 'C45E, ReH=600').
  - Holes:** A checkbox labeled 'Has radial holes' which is checked.
  - Slots:** A checkbox labeled 'Has longitudinal slots' which is unchecked.

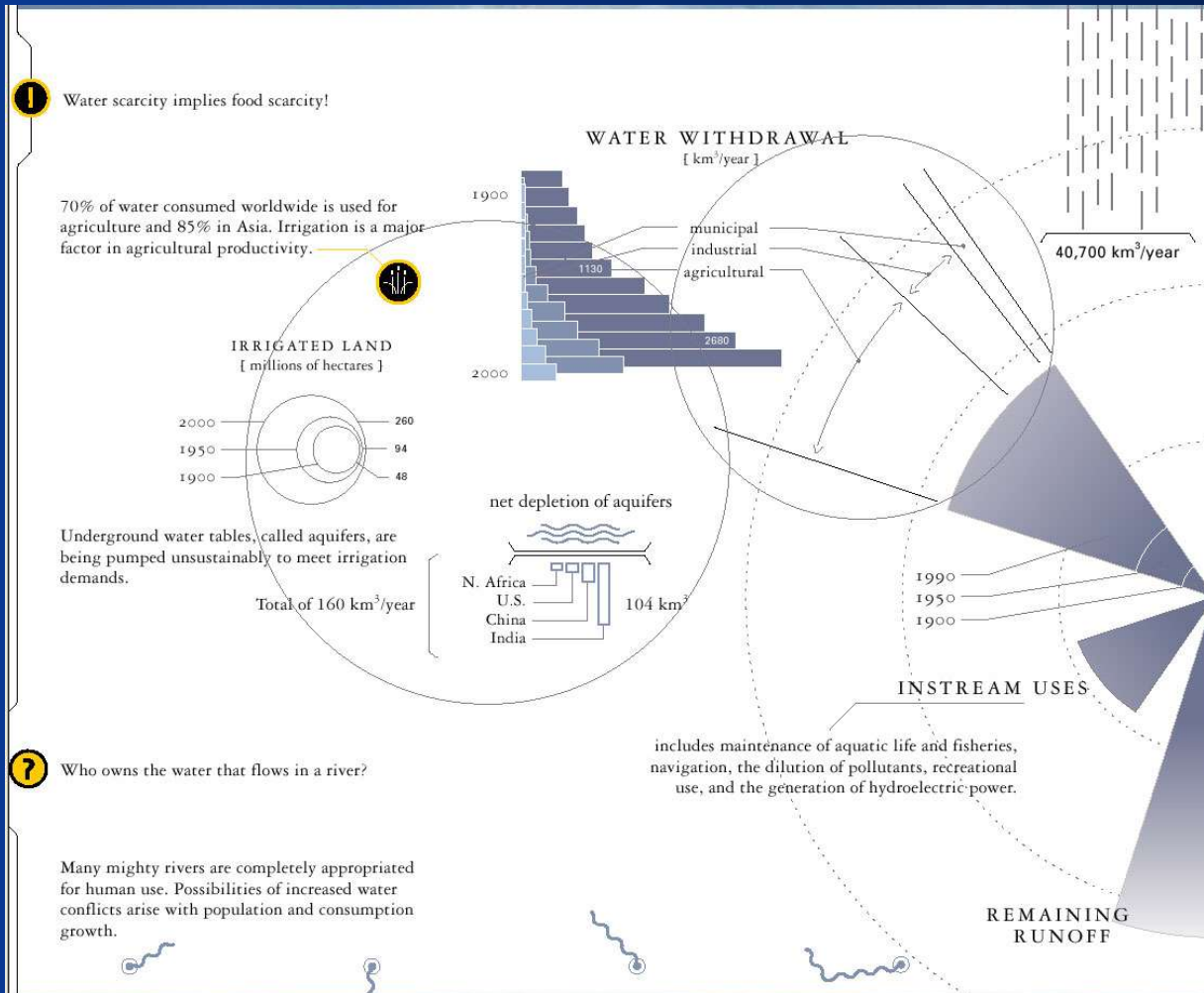
© 2003 JGoodies.com



# *Es geht um Usability*

- Was wir verbessern wollen:
  - Lesbarkeit
  - Fasslichkeit
  - Eleganz (geeignete Wahl)
  - Gebrauchsfähigkeit

# Es geht nicht um Kunst



# *Was ist Gestaltung?*

- Gestalten ist:
  - planen, kontrollieren,
  - gruppieren, ordnen, ausrichten,
  - beziehen, skalieren, balancieren,
  - Bedeutung zufügen, vereinfachen,
  - Klarheit schaffen
  
- Layout ist wichtig für gute Gestaltung

# *Rollen und Tätigkeiten*

- Ein Meta-Designer **definiert** einen Stil
- Der Grafikgestalter **findet** ein Layout
- Ein Entwickler **konstruiert** das Layout
- Code **füllt** einen Container mit Widgets
- Layout-Manager **rechnet und setzt** Positionen

# II. Analyse

*Was sind die Probleme?*

# *Wir analysieren:*

- Probleme, die wir Menschen haben
- Anforderungen an gute Gestaltung
- Wie man leichter zu gutem Layout kommt

# *Was macht es uns schwer?*

- Anleitungen, Bücher und Programme zeigen schlechte Gestaltung
- Layout-Manager sind schwer zu erlernen
- Man kann schlecht mit ihnen arbeiten
- Layout-Code ist schwer zu lesen
- Layout ist schwer am Code zu erkennen

# Länge des Layout-Codes

Ein Beispiel aus Suns Java-Tutorial



SpringLayout

Name:

Phone:

Fax:

Email:

Address:

The SpringLayout window shows a form with five text input fields. The 'Name' field contains the text 'Karsten'. The other fields are empty. The layout is simple and clean.



GridBagLayout

Name:

Phone:

Fax:

Email:

Address:

The GridBagLayout window shows a form with five text input fields. The 'Name' field contains the text 'Karsten'. The other fields are empty. The layout is more complex and less clean than SpringLayout.



FormLayout

Name:

Phone:

Fax:

Email:

Address:

The FormLayout window shows a form with five text input fields. The 'Name' field contains the text 'Karsten'. The other fields are empty. The layout is very clean and professional.



# Layout-Code zu 3 Stilen

```
private Component buildComponentPane() {
    nameLabel = new JLabel("Name");
    phoneLabel = new JLabel("Phone");
    faxLabel = new JLabel("Fax");
    emailLabel = new JLabel("Email");
    addressLabel = new JLabel("Address");
    nameField = new JTextField(20);
    phoneField = new JTextField(20);
    faxField = new JTextField(20);
    emailField = new JTextField(20);
    addressField = new JTextField(20);

    return createPane(new Component[] {
        nameLabel, phoneLabel, faxLabel, emailLabel, addressLabel,
        nameField, phoneField, faxField, emailField, addressField },
        10, 10, 6, 6);
}

private static Component createPane(Component[] leftComponents,
    Component[] rightComponents,
    int initialX, int initialY,
    int wwid, int hwid) {

    springLayout layout = new springLayout();
    int maxWidth = Math.max(leftComponents.length, rightComponents.length);

    // the constant springing we'll use to enforce spacing.
    spring yspring = spring.constant(initialY);
    spring xpadding = spring.constant(wwid);
    spring ypadding = spring.constant(hwid);
    spring hgappadding = spring.constant(wwid);

    // create the container and add the components to it.
    JPanel parent = new JPanel(layout);
    for (int i = 0; i < wwid; i++) {
        parent.add(leftComponents[i]);
        parent.add(rightComponents[i]);
    }

    spring maxWidthspring = layout.getConstraints("maxc", leftComponents[0]);
    for (int row = 0; row < hwid; row++) {
        maxWidthspring = spring.max(maxWidthspring,
            layout.getConstraints("maxc",
                leftComponents[row]));
    }

    springLayout.constraints["maxc"] = null;
    springLayout.constraints["maxcmax"] = null;
    spring parentWidth = layout.getConstraints("maxc", parent);
    spring rwidth = null;
    spring maxHeightSpring = null;
    spring rx = spring.sum(maxWidthspring, xpadding); //right col location
    spring negrx = spring.minus(rx); //negative of rx

    for (int row = 0; row < hwid; row++) {
        springLayout.constraints["maxc"] = layout.getConstraints(
            leftComponents[row]);
        springLayout.constraints["maxc"] = layout.getConstraints(
            rightComponents[row]);

        comel.setSize(yspring);
        comel.setSize(rx);

        rwidth = comel.getWidth();
        comel.setSize(spring.sum(parentWidth, negrx),
            spring.max(ypadding));

        if (row == 0) {
            comel.setSize(yspring);
            comel.setSize(yspring);
            maxHeightSpring = spring.sum(yspring,
                spring.max(comel.getHeight(),
                    comel.getHeight()));
        } else { // row > 0
            spring y = spring.sum(yspring.max(
                layout.constraints["maxc"],
                layout.constraints["maxc"]},
                ypadding);

            comel.setSize(y);
            comel.setSize(y);
            maxHeightSpring = spring.sum(ypadding,
                spring.sum(maxHeightSpring,
                    spring.max(
                        comel.getHeight(),
                        comel.getHeight())));
        }

        layout["maxc"] = comel;
        layout["maxcmax"] = comel;
    }
    // end of the loop
    springLayout.constraints["maxc"] = parent;
    comelParent.getConstraints("maxc",
        spring.sum(rx, spring.sum(rwidth,
            xpadding)));
    comelParent.getConstraints("maxc",
        spring.sum(maxHeightSpring, ypadding));
    return parent;
}
```

```
private Component buildComponentPane() {
    nameLabel = new JLabel("Name");
    phoneLabel = new JLabel("Phone");
    faxLabel = new JLabel("Fax");
    emailLabel = new JLabel("Email");
    addressLabel = new JLabel("Address");
    nameField = new JTextField(20);
    phoneField = new JTextField(20);
    faxField = new JTextField(20);
    emailField = new JTextField(20);
    addressField = new JTextField(20);

    gridBagLayout layout = new gridBagLayout();
    JPanel parent = new JPanel(layout);
    parent.addComponent(new GridBagConstraints(10, 10, 10, 10));

    gridBagConstraints gbc = new GridBagConstraints();
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    gbc.insets = new Insets(0, 0, 0, 0);
    gbc.insets = new Insets(0, 0, 0, 0);

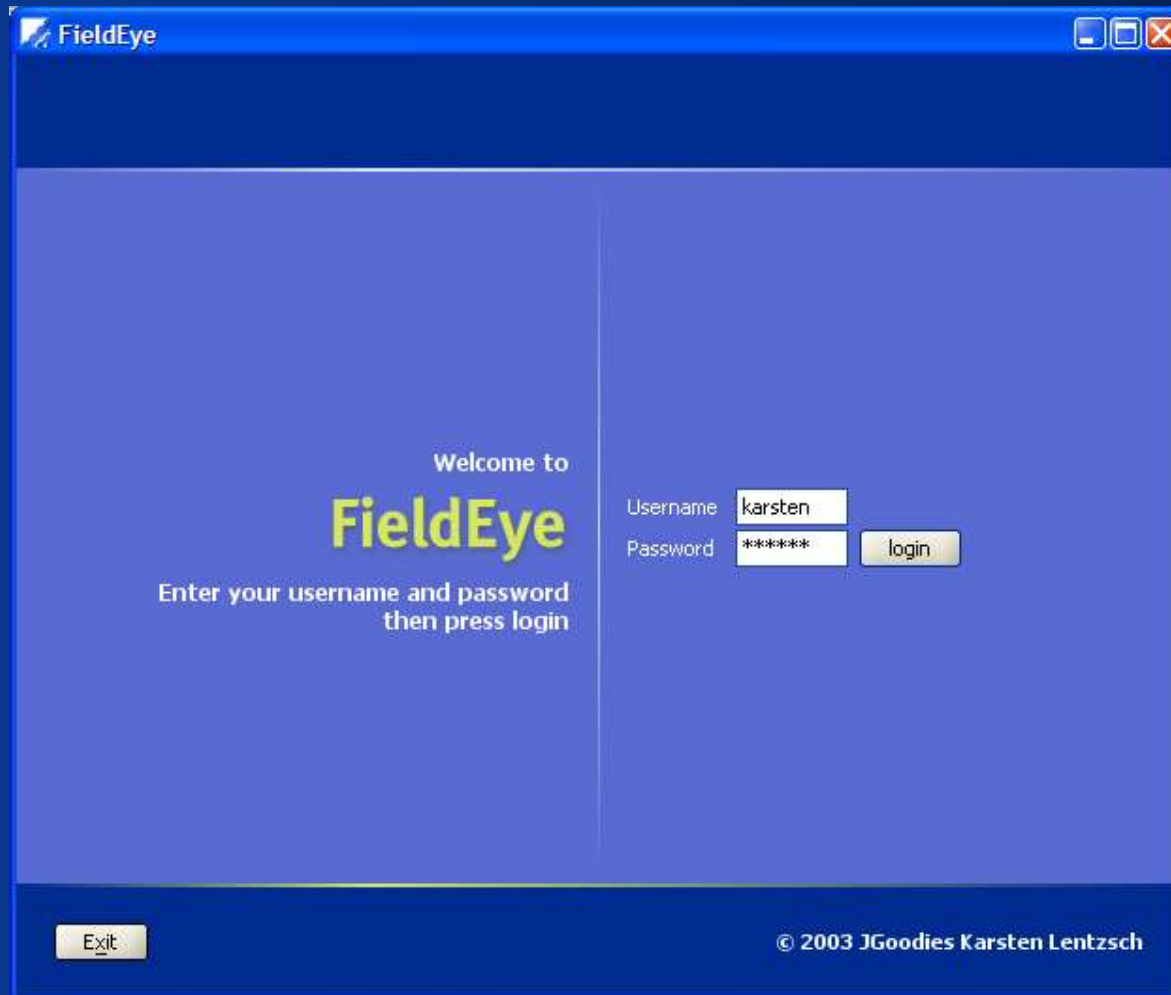
    gbc.gridwidth = 1;
    gbc.gridheight = 0;
    parent.add(nameLabel, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(phoneLabel, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(faxLabel, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(emailLabel, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(addressLabel, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(nameField, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(phoneField, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(faxField, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(emailField, gbc);
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    parent.add(addressField, gbc);

    return parent;
}
```

```
private Component buildComponentPane() {
    nameLabel = new JLabel("Name");
    phoneLabel = new JTextField(20);
    faxLabel = new JTextField(20);
    emailField = new JTextField(20);
    addressField = new JTextField(20);

    FormLayout layout = new FormLayout(
        "left:gap, left, gap",
        "default:row:filler | new:default:row:filler(layout);
        builder.addComponent(nameLabel);
        builder.append("phone:", phoneField);
        builder.append("fax:", faxField);
        builder.append("email:", emailField);
        builder.append("address:", addressField);
    );
    return builder.getPane();
}
```

# *Erwäge Symmetrie*



# *Gleiche Breiten an*

**S Segment 1**

<b>Segment</b>			
Identifier	Segment 1	Power [kW]	
PTI [kW]			
len [mm]			
<b>Diameters</b>			
da [mm]		di [mm]	
da2 [mm]		di2 [mm]	
R [mm]		D [mm]	
<b>Criteria</b>			
Location	Propeller nut thread ▼	k-factor	
Connection	C45E, ReH=600 ▼		
Holes	<input checked="" type="checkbox"/> Has radial holes		
Slots	<input type="checkbox"/> Has longitudinal slots		

# *Gleiche Höhen an*

The image shows a software dialog box titled "Segment 1". It contains several sections for configuring a segment:

- Segment**: Identifier (Segment 1), PTI [kW], Power [kW], and len [mm].
- Diameters**: da [mm], di [mm], da2 [mm], di2 [mm], R [mm], and D [mm].
- Criteria**: Location (Propeller nut thread), k-factor, Connection (C45E, ReH=600), Holes (checked), and Slots (unchecked).

Fields for R [mm], D [mm], and k-factor are highlighted in red, indicating they are the focus of the "Gleiche Höhen an" (Equal Heights) feature.

# Richte Grundlinien aus

**Segment 1**

**Segment**

Identifier

PTI [kW]  Power [kW]

len [mm]

**Diameters**

da [mm]  di [mm]

da2 [mm]  di2 [mm]

R [mm]  D [mm]

**Criteria**

Location  k-factor

Connection

Holes  Has radial holes

Slots  Has longitudinal slots

# Gleiche Panel an

**F** Flange 1a

**Flange**

Identifier: Flange 1a

PTI [kW]  Power [kW]

s [mm]

**Diameters**

da [mm]  di [mm]

da2 [mm]  di2 [mm]

R [mm]  D [mm]

**Criteria**

Location: Propeller nut thread  k-factor

**Bolts**

Material: C45E, ReH=600

Numbers

ds [mm]

**S** Segment 1

**Segment**

Identifier: Segment 1

PTI [kW]  Power [kW]

len [mm]

**Diameters**

da [mm]  di [mm]

da2 [mm]  di2 [mm]

R [mm]  D [mm]

**Criteria**

Location: Propeller nut thread  k-factor

Connection: C45E, ReH=600

Holes  Has radial holes

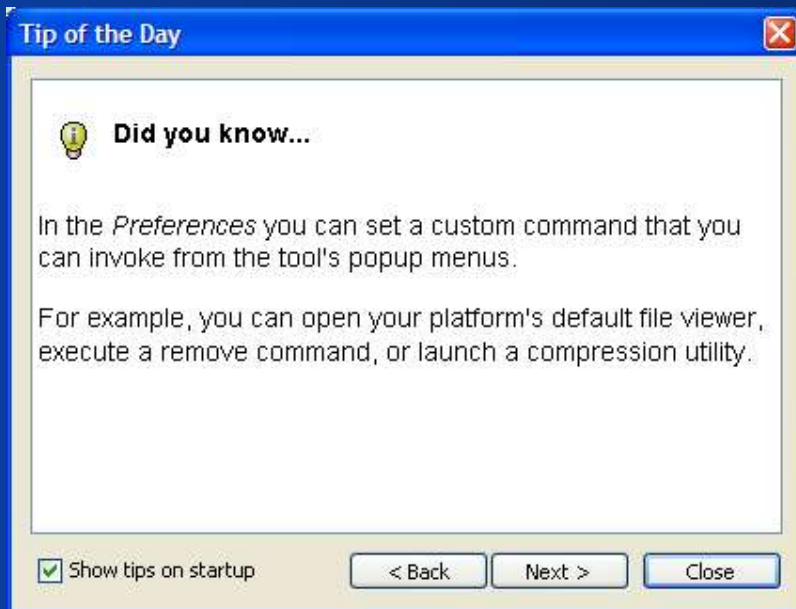
Slots  Has longitudinal slots

# *Beachte minimale Größen*

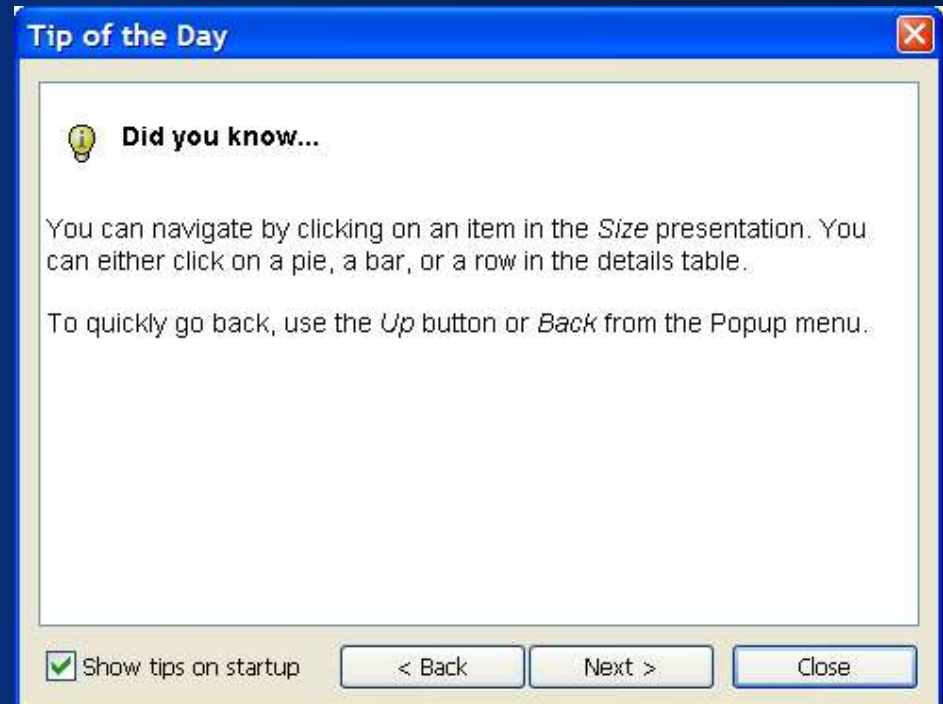
- Knöpfe brauchen eine geeignete minimale Breite
- Der Knopf *weiß* die minimale Breite nicht, denn in verschiedenen Kontexten ist er schmal oder breit.



# Skaliere mit Schrift und Auflösung



96dpi



120dpi



# *Keine Pixel in Screen Design!*

Proportionen gehen sonst verloren



GridBagLayout

Name: Karsten

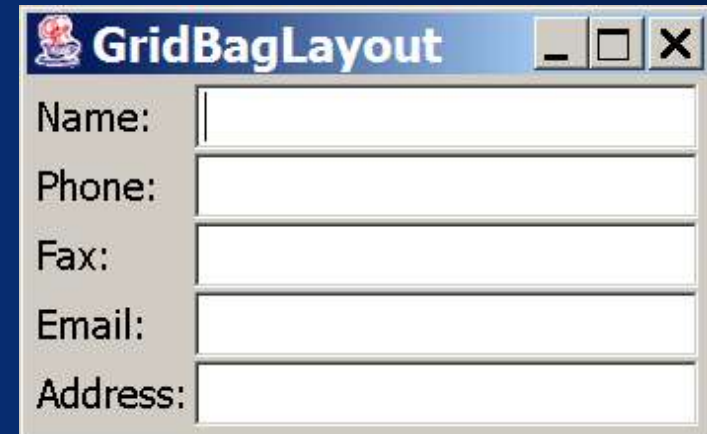
Phone:

Fax:

Email:

Address:

96dpi



GridBagLayout

Name:

Phone:

Fax:

Email:

Address:

180dpi

# *Weitere Schwächen*

- Einfache Aufgaben sind schwierig zu lösen
- Layout ist schwer wieder zu verwenden
- Keine logischen Layouts (Mac vs. PC)
- Dem Layout Manager fehlen Funktionen
- LM-Implementierung ist schwer zu prüfen
- Das LM-API ist überladen
- Der Layout Manager ist langsam

# *Zusammenfassung*

Kniffliges ist unmöglich...

... und Einfaches schwierig.

# III. Ziele

*Was wollen wir?*

# *Hauptziel*

Gute Gestaltung einfach machen ...  
... und schlechte schwierig.

# Ziele I

- Schnell Formulare bauen
- Lösung beherrscht 90% aller Panels
- Anfänger schaffen gute Ergebnisse
- Experten sparen Zeit
- Code ist leicht zu lesen und zu verstehen
- Gestaltung ist konsistent über Panel, Anwendungen, Entwickler und Teams

# *Ziele II*

- Die Lösung arbeitet mit visuellen Editoren,
  - die unsere Produktivität steigern
  - oder Ergebnisse verbessern
- Der Panel-Bau ist einfach zu lernen
- Die Lösung kommt mit guten Beispielen
- Die Lösung liefert alles was man braucht

# IV. Konzepte

*Welchen Weg gehen wir?*



# *Wie gehen wir heran?*

- Nimm' ein **Gitter** für ein Layout
- Nimm' ein **Gittersystem** für viele Layouts
- Verwende eine mächtige Layoutsprache
- Verkürze Code durch String-Encodings
- Separate Concerns
- Biete Hilfe oberhalb des Layout-Managers

# *Gitter*

- Gitter sind mächtig, flexibel und einfach
- Gitter sind einfach zu verstehen
- Grafikerdesigner verwenden Gitter um:
  - Layout zu finden
  - Elemente auszurichten
- Viele Leute benutzen Gitter implizit, wenn sie mit Bleistift und Papier gestalten

# *Gittersysteme*

- Gitter skalieren gut
- Gittersystem erschlagen viele Probleme
- Gittersysteme drängen zu gutem Layout
- Sie leiten uns, ohne uns einzuschränken;  
dann können wir kreativ werden

# *Layout-Reihenfolge und -Sprache*

- Erst beschreiben wir das Layout, dann bauen wir das Panel
- Wir nutzen eine mächtige Layout-Sprache
- Zellen erben von Spalten und Zeilen
  
- So können wir:
  - das Layout aus der Beschreibung ermitteln
  - häufiges Layout schnell hinschreiben
  - komplexes Layout in wenige Zeilen schreiben

# *Layout-Beschreibung in Strings*

- Wir beschreiben ein Layout mit Objekten oder durch lesbare Strings
- So können wir
  - Einfache Gestaltung in zwei Zeilen schreiben
  - Umfangreicheres in wenigen Zeilen definieren

# *Separation of Concerns*

- Der Layout-Manager soll Positionen ermitteln und setzen – sonst nichts
- Andere Klassen helfen
  - das Gitter zu durchlaufen
  - häufige Komponenten zu erzeugen
  - das Gitter zu erweitern
  - einem Stil zu folgen
  - Panels aus XML zu bauen

# *Separation of Concerns II*

- Das Layout-Manager-API bleibt klein
- Wir können Hilfsklassen kombinieren
- Änderungen beeinflussen nur Einzelteile
- Das Layout-System ist mächtig, jedes Teil bleibt einfach

# V. Lösung

*Eine Umsetzung unserer Konzepte*

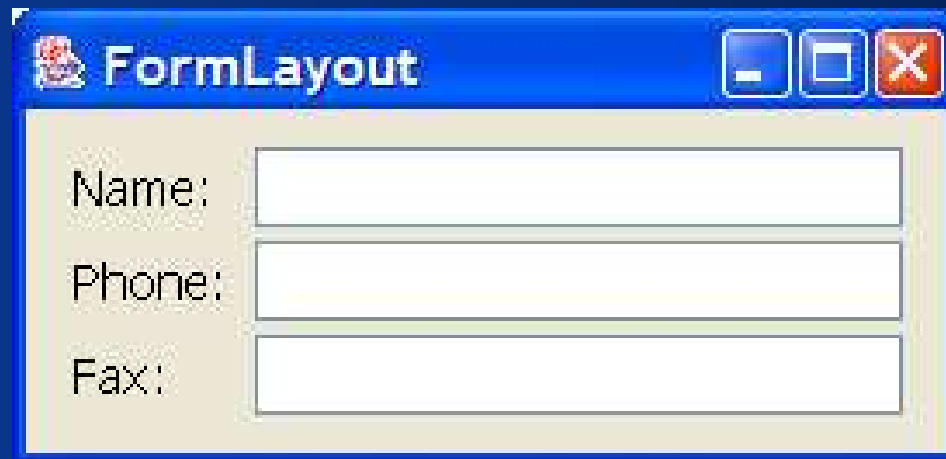


# *Lösung: JGoodies Forms*

- Forms ist **eine** Umsetzung der Konzepte
- Wir wollen lernen, wie man damit arbeitet
- Und verstehen, was es erleichtert

# *Beispiel: Ein Formular*

Wie baut man dieses einfache Formular?



The image shows a screenshot of a Java Swing window titled "FormLayout". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is light beige and contains a simple form with three text input fields. The labels "Name:", "Phone:", and "Fax:" are positioned to the left of their respective input boxes. The input boxes are empty and have a light gray border.

# *1: Anforderungen*

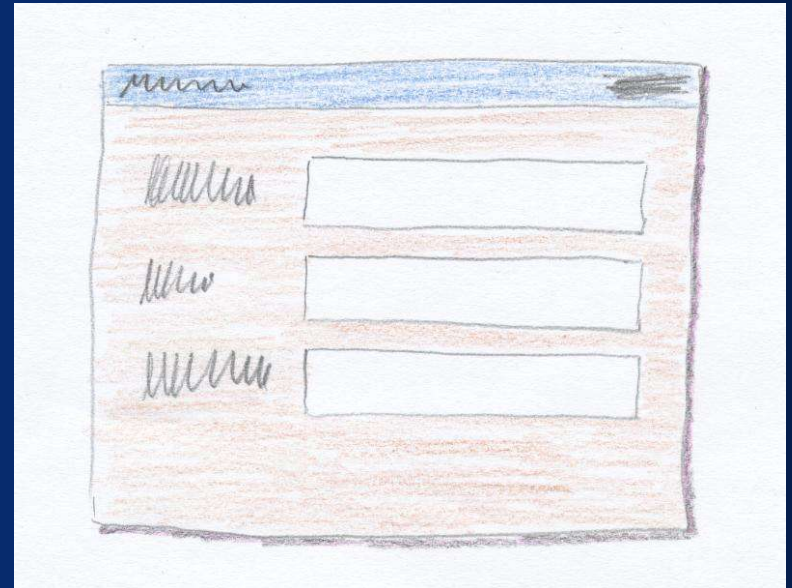
Chef sagt:

„Wir brauchen eine Maske,  
um eine Adresse einzugeben.

Die soll Felder haben für:  
Name, Telefon und Email.“

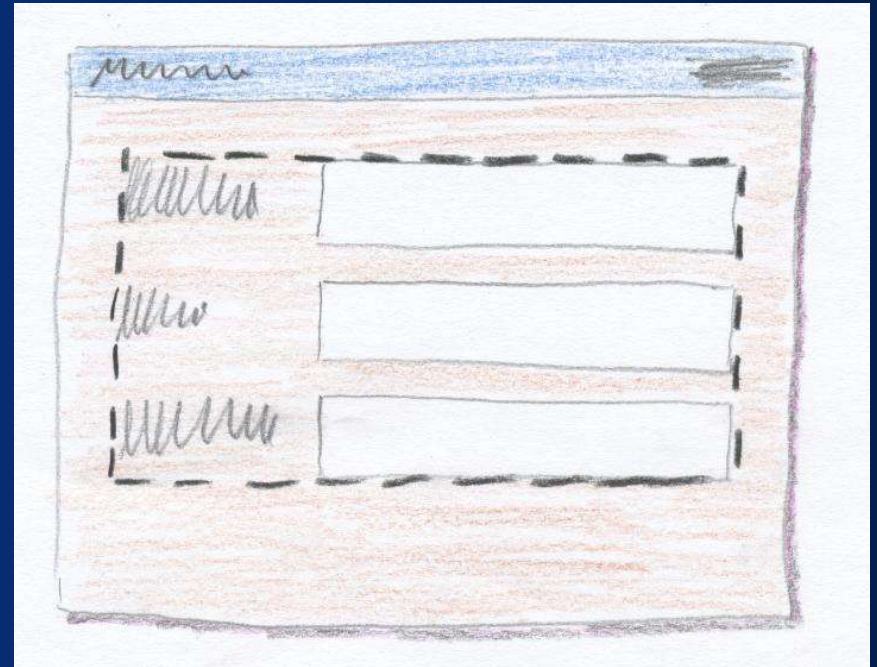
## 2: *Layout finden*

- Die Grafikdesignerin gestaltet eine Vorlage
- Sie reicht die an den Entwickler und sagt: „Beachte bitte den MS-Style Guide!“



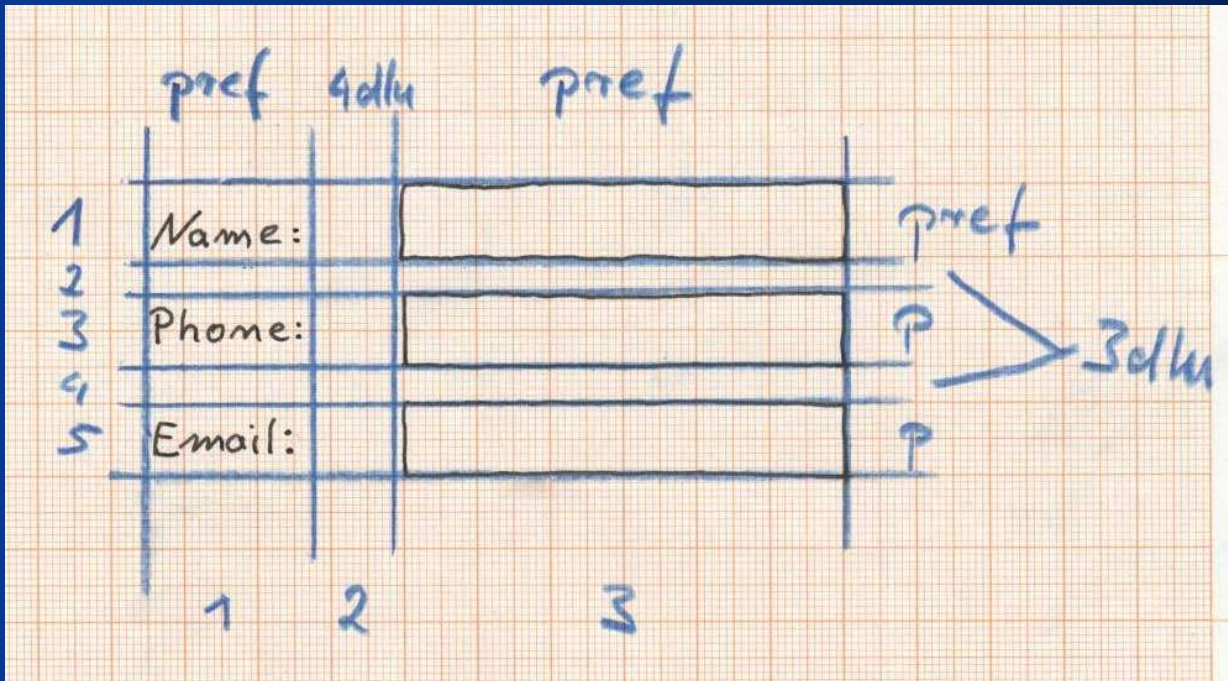
# 3a: *Auf Inhalt konzentrieren*

- Der Entwickler sucht Standard-Rahmen



# 3b: Gitter konstruieren

- Der Entwickler findet das Gitter
- Und ermittelt Spalten- und Zeilengrößen



# 4: *Layout beschreiben*

Der Entwickler beschreibt das Layout:

```
FormLayout layout = new FormLayout(  
    "pref, 4dlu, pref",  
    "pref, 3dlu, pref, 3dlu, pref");
```

# *4b: Layout verfeinern*

Texte links-bündig, Felder wachsen

```
FormLayout layout = new FormLayout(  
    "left:pref, 4dlu, pref:grow",  
    "pref, 3dlu, pref, 3dlu, pref");
```



# 4c: *Layout verfeinern*

Minimale Spaltenbreite; Kurzschreibweise

```
FormLayout layout = new FormLayout(  
    "left:max(75dlu;pref), 4dlu, pref:grow",  
    "p, 3dlu, p, 3dlu, p");
```

# 5: *Komponenten zufügen*

```
JPanel panel = new JPanel(layout);  
CellConstraints cc = new CellConstraints();  
  
panel.add(new JLabel("Name:"), cc.xy(1, 1));  
panel.add(nameField, cc.xy(3, 1));  
  
panel.add(new JLabel("Phone:"), cc.xy(1, 3));  
panel.add(phoneField, cc.xy(3, 3));  
  
...
```

# 5b: Einen Builder nutzen

## PanelBuilder (empfohlen)

```
PanelBuilder builder = new PanelBuilder(layout);  
CellConstraints cc = new CellConstraints();
```

```
builder.addLabel("Name:", cc.xy(1, 1));  
builder.add(nameField, cc.xy(3, 1));
```

```
builder.addLabel("Phone:", cc.xy(1, 3));  
builder.add(phoneField, cc.xy(3, 3));
```

...

# 5c: Zeilenvariable

## Zeilenvariable (nicht empfohlen)

```
PanelBuilder builder = new PanelBuilder(layout);  
CellConstraints cc = new CellConstraints();  
int row = 1;
```

```
builder.addLabel("Name:", cc.xy(1, row));  
builder.add(nameField, cc.xy(3, row));
```

```
row += 2;  
builder.addLabel("Phone:", cc.xy(1, row));  
builder.add(phoneField, cc.xy(3, row));
```

...

# 5d: Mit bequemen Builder

## Mit DefaultFormBuilder

```
FormLayout layout = new FormLayout(  
    "1:p, 4dlu, p:g"); // Columns  
                        // Add rows dynamically
```

```
DefaultFormBuilder builder =  
    new DefaultFormBuilder(layout);
```

```
builder.append("Name:", nameField);  
builder.append("Phone:", phoneField);  
builder.append("Email:", emailField);
```

```
return builder.getPanel();
```

# 6: *Rahmen* zufügen

```
DefaultFormBuilder builder =  
    new DefaultFormBuilder(layout);
```

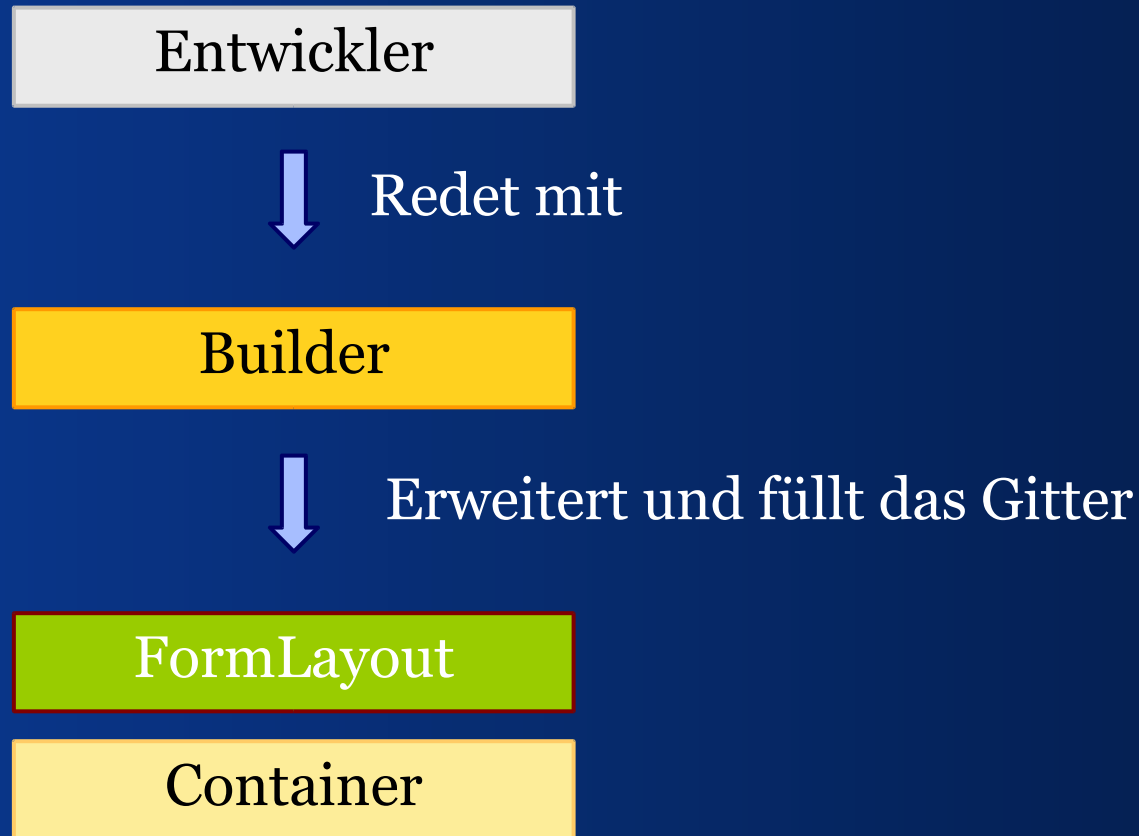
```
builder.setDefaultDialogBorder();
```

...

# *Fabriken und Logische Größen*

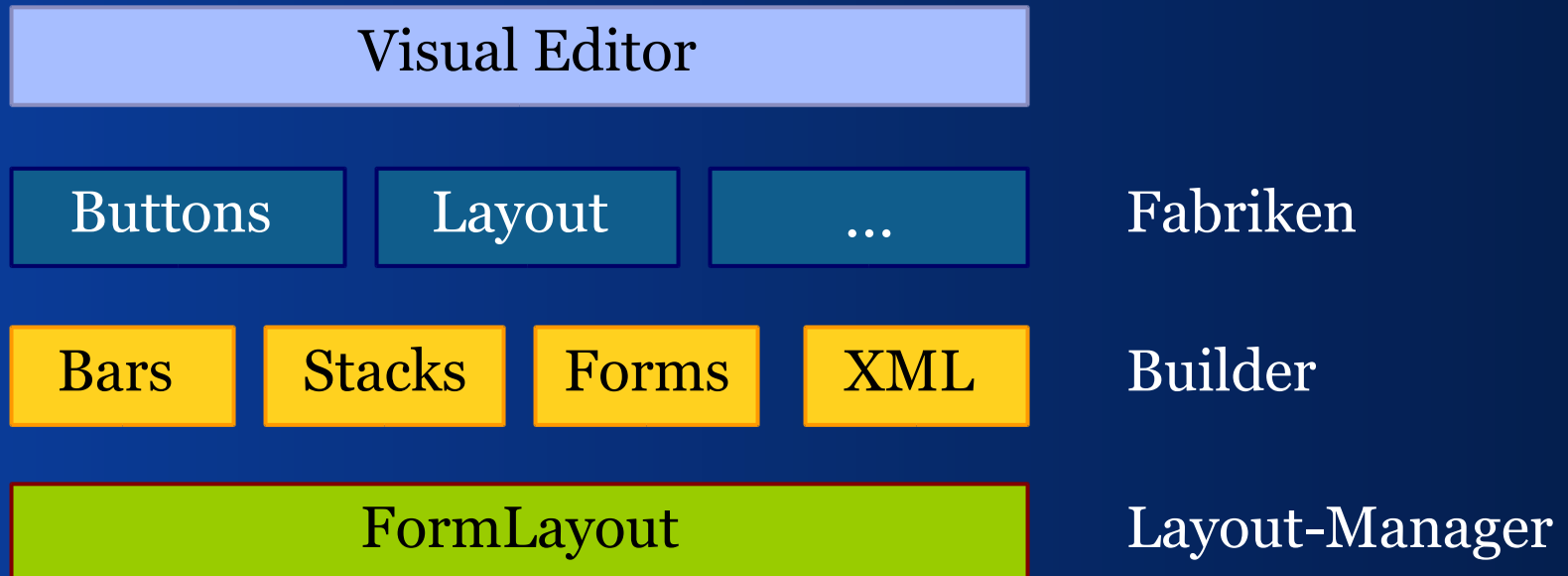
- Der **ButtonBarBuilder**:
  - baut konsistente Knopfleisten
  - beachtet den Plattform-Stil
  - nutzt logische Größen
  - unterstützt logisches Layout (Mac vs. PC)
- Die **ButtonBarFactory**:
  - liefert fertige Knopfleisten
  - nutzt logisches Layout (Mac vs. PC)

# *Nicht-visuelle Builder*





# *Schichten in Forms*



# Demo

# VI. Ausblick

*Was fehlt noch? Wie geht's weiter?*

# Wie kriegt man gute Gestaltung?

Der „typische“  
Entwickler  
geht Folgendes  
nicht an:



# Vorlagen und Wizard

Design Wizard

Standard Dialog Library

NetBeans

Eclipse

IDEA

Visuelle Editoren

Buttons

Layout

...

Fabriken

Forms

Bars

Stacks

XML

Builder

FormLayout

Layout-Manager

# *Was kommt?*

- Mehr Layout-Vorlagen
- Visuelle Form-Editoren
- Mehr nicht-visuelle Builder
- Bessere Unterstützung logischer Größen
- Inter-Panel Layout-Zusammenhänge
- Unterstützung für „perceived bounds“

# *Zusammenfassung*

- Wir haben Gestaltungsprobleme analysiert
- Wir haben gelernt, sie anzugehen
- Wir haben eine Anfangslösung gesehen
- Wir haben Verbesserungen umrissen

# Referenzen

- MS Design Specifications and Guidelines  
[msdn.microsoft.com](http://msdn.microsoft.com), auf 'library' klicken
- Aqua Human Interface Guidelines  
[www.apple.com/developer](http://www.apple.com/developer)
- JGoodies Forms-Rahmenwerk  
[forms.dev.java.net](http://forms.dev.java.net)



# *Ein guter Layout-Manager*

- ExplicitLayout, siehe [www.zookitec.com](http://www.zookitec.com)
  - ist mächtig
  - bietet Stile – ähnlich den Forms Buildern
  - kann auch kurvige Gestaltung
  - kann nur Pixel-Größen
  - kennt keine logischen Größen
  - verfügbar unter der LGPL

# *Literatur*

Kevin Mullet und Darrel Sano

*Designing Visual Interfaces*

Prentice Hall

250 Seiten, von denen man 100 lesen sollte

# *Merke:*

Studiere Mullet & Sano ...

lerne zu sehen ...

und nutze Forms

# Fragen und Antworten

# Frage

Funktioniert Forms mit AWT und SWT?

# Antwort

Forms funktioniert mit AWT nicht SWT.

Zusatzschichten gehen nur mit Swing.

Forms ist nach SWT portiert worden.

# Frage

Wie stabil ist Forms?

Kann man damit produzieren?

# Antwort

FormLayout ist stabil seit Dezember 2002;  
Builder sind gut nutzbar.



*Ende*

Hoffentlich hilft's!

Viel Erfolg!