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

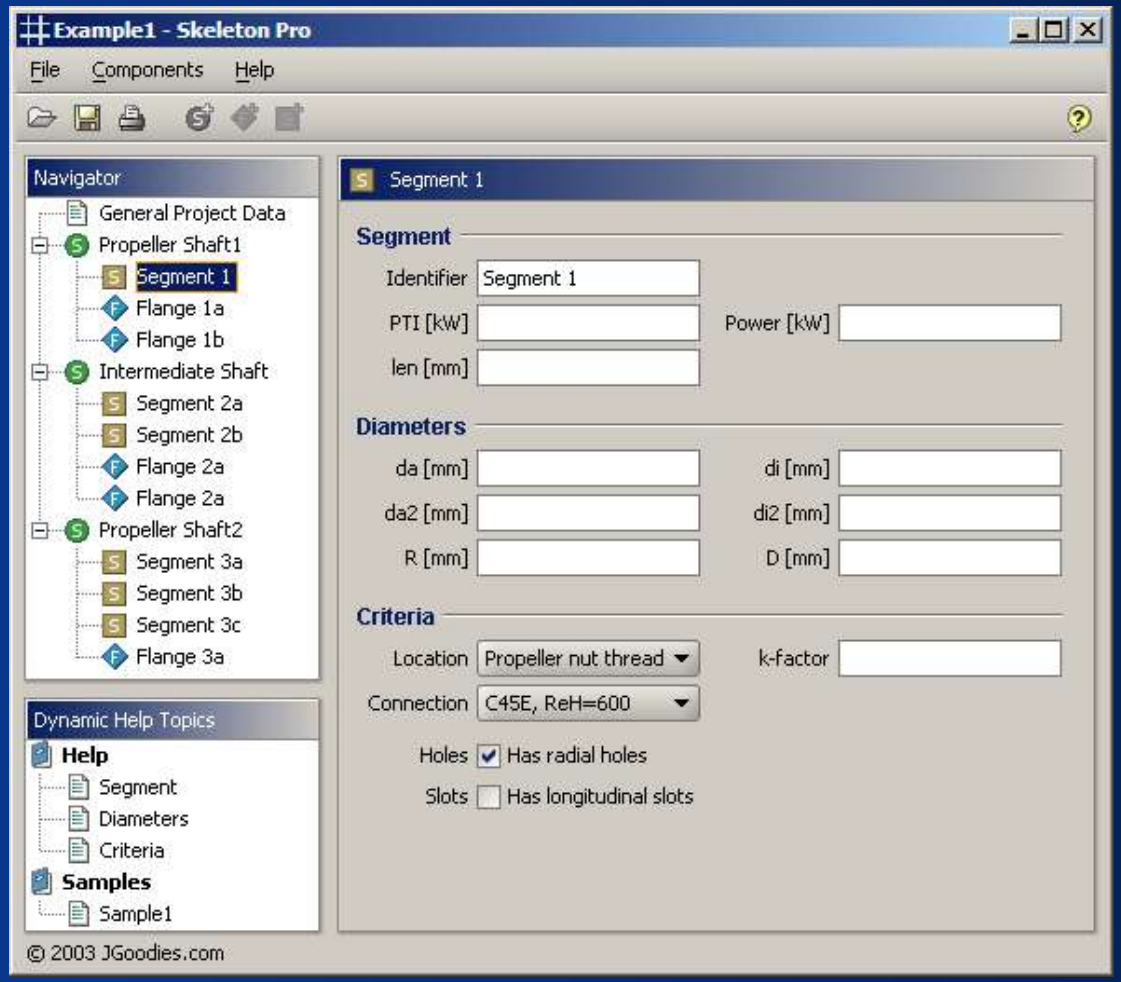
- Einführung
- Analyse — Was sind die Probleme?
- Ziele — Was wollen wir?
- Konzepte — Welchen Weg gehen wir?
- Lösung — Wie setzen wir das um?
- Ausblick — Was fehlt noch?

I. Einführung

Swing-Bausteine

Swing-Bausteine

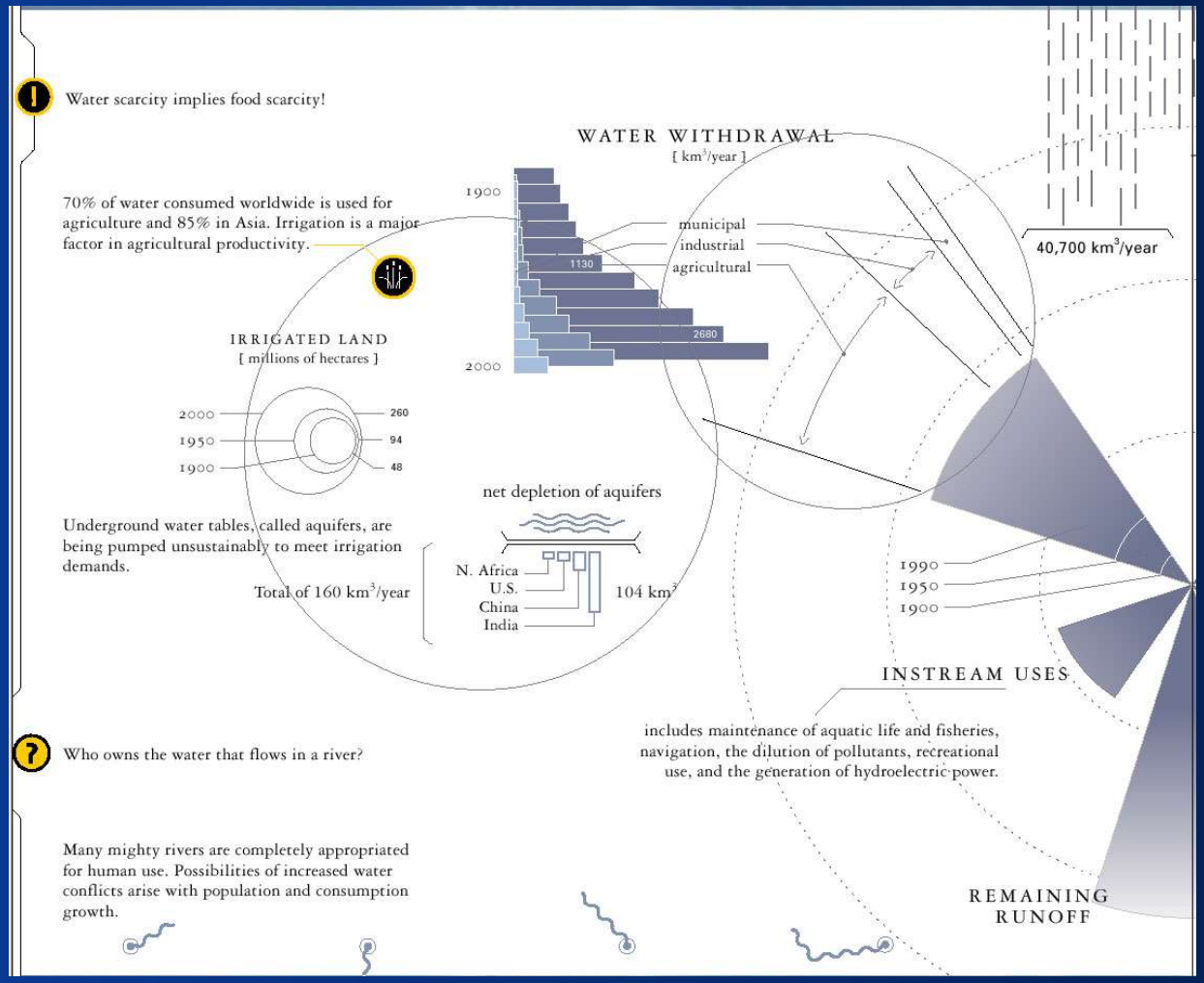
Wir behandeln Formulare



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

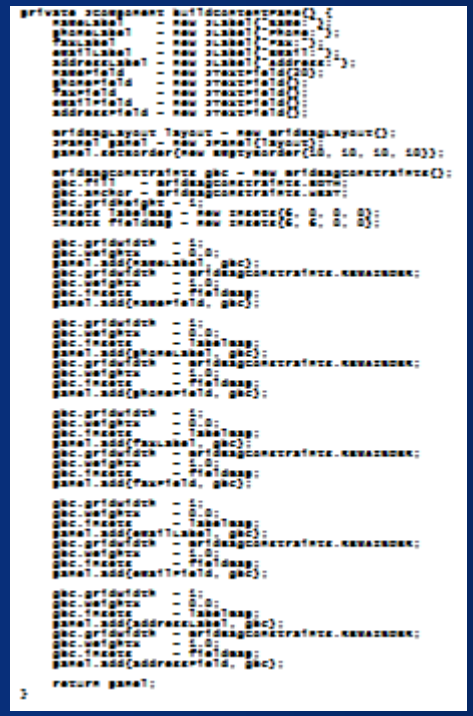
Was macht es uns schwer?

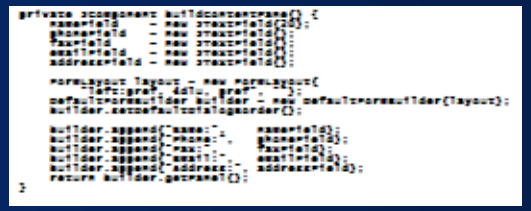
- Anleitungen, Bücher und Programme zeigen schlechte Gestaltung
- Layout-Manager sind schwer zu erlernen
- Man kann schlecht mit ihnen arbeiten
- Layout-Code ist schwer zu lesen
- Layout ist schwer am Code zu erkennen

Länge des Layout-Codes

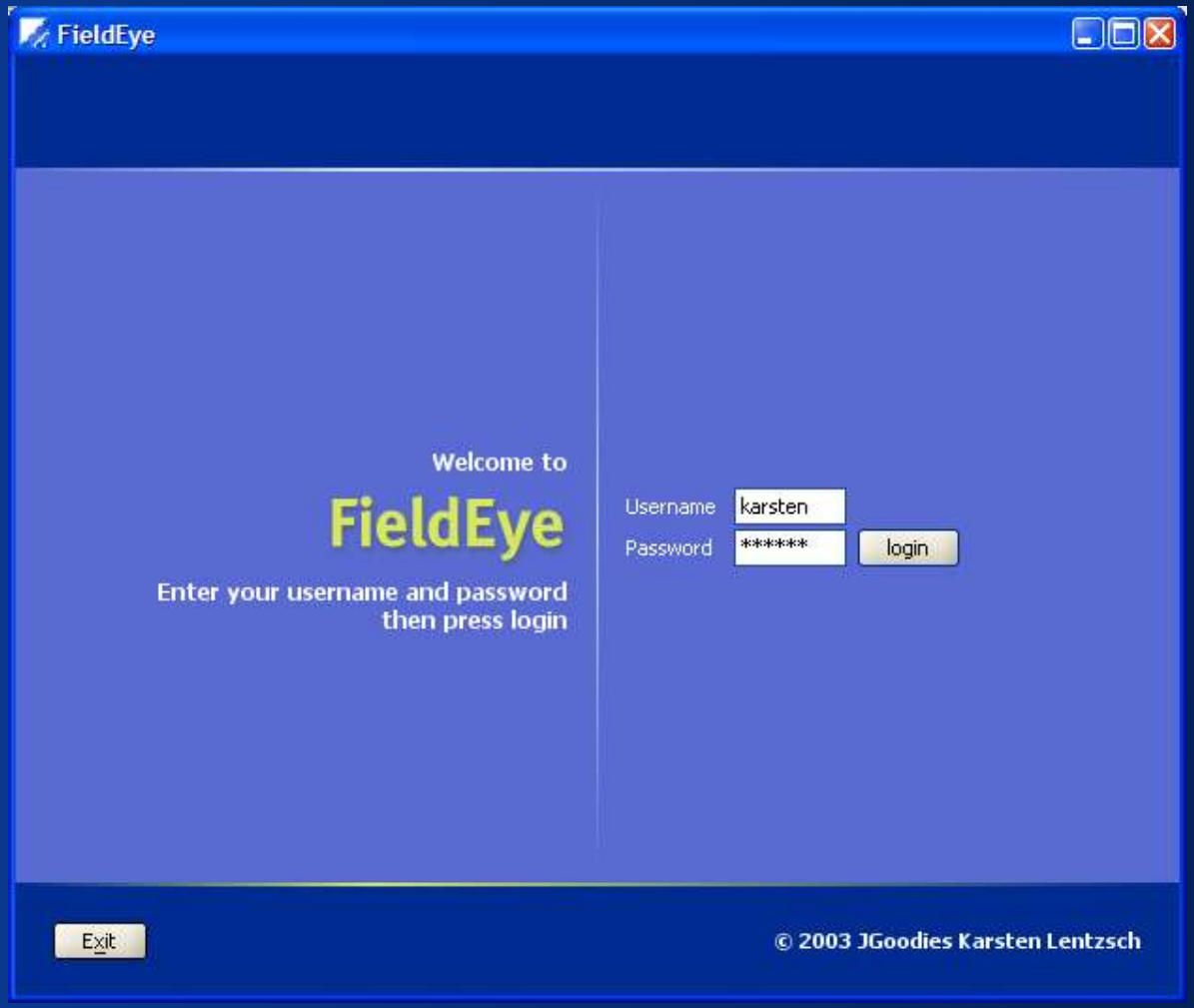
Ein Beispiel aus Suns Java-Tutorial

Layout-Code zu 3 Stilen

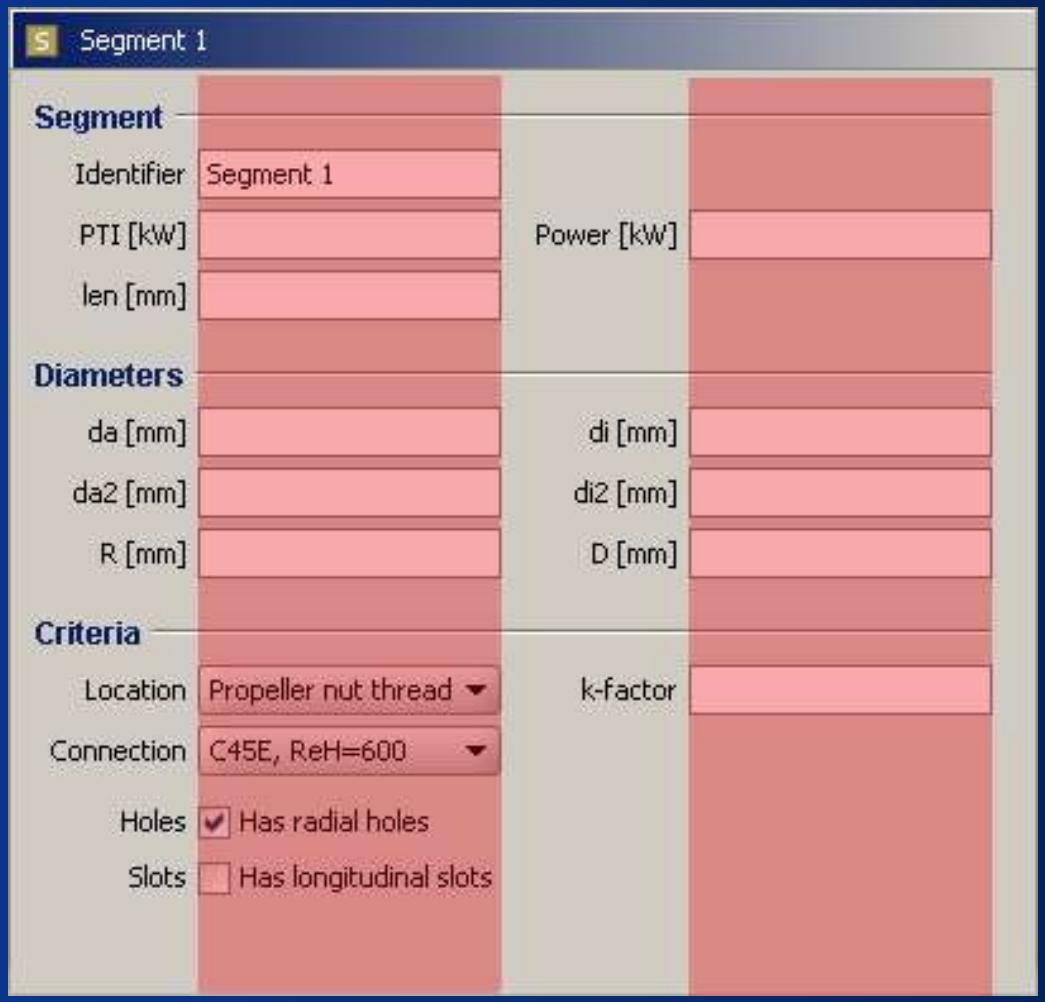




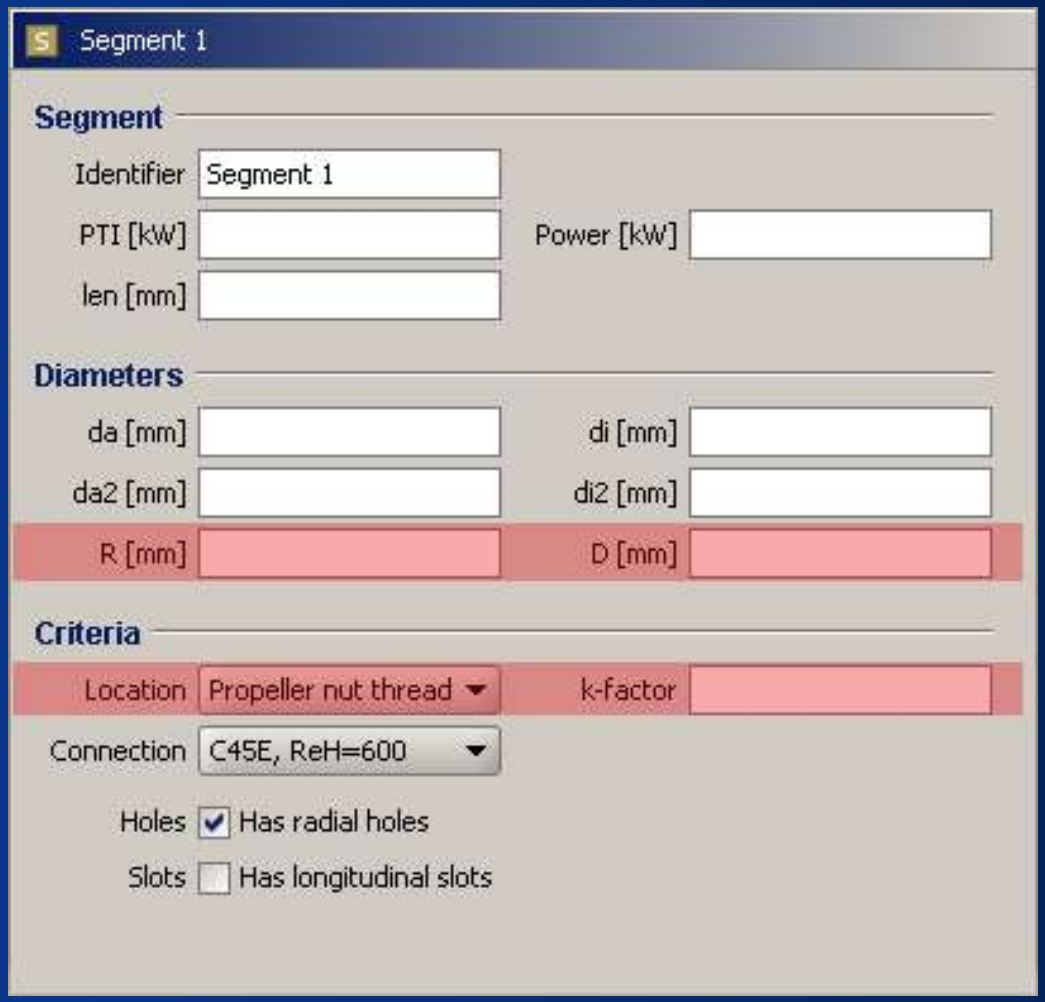
Erwäge Symmetrie



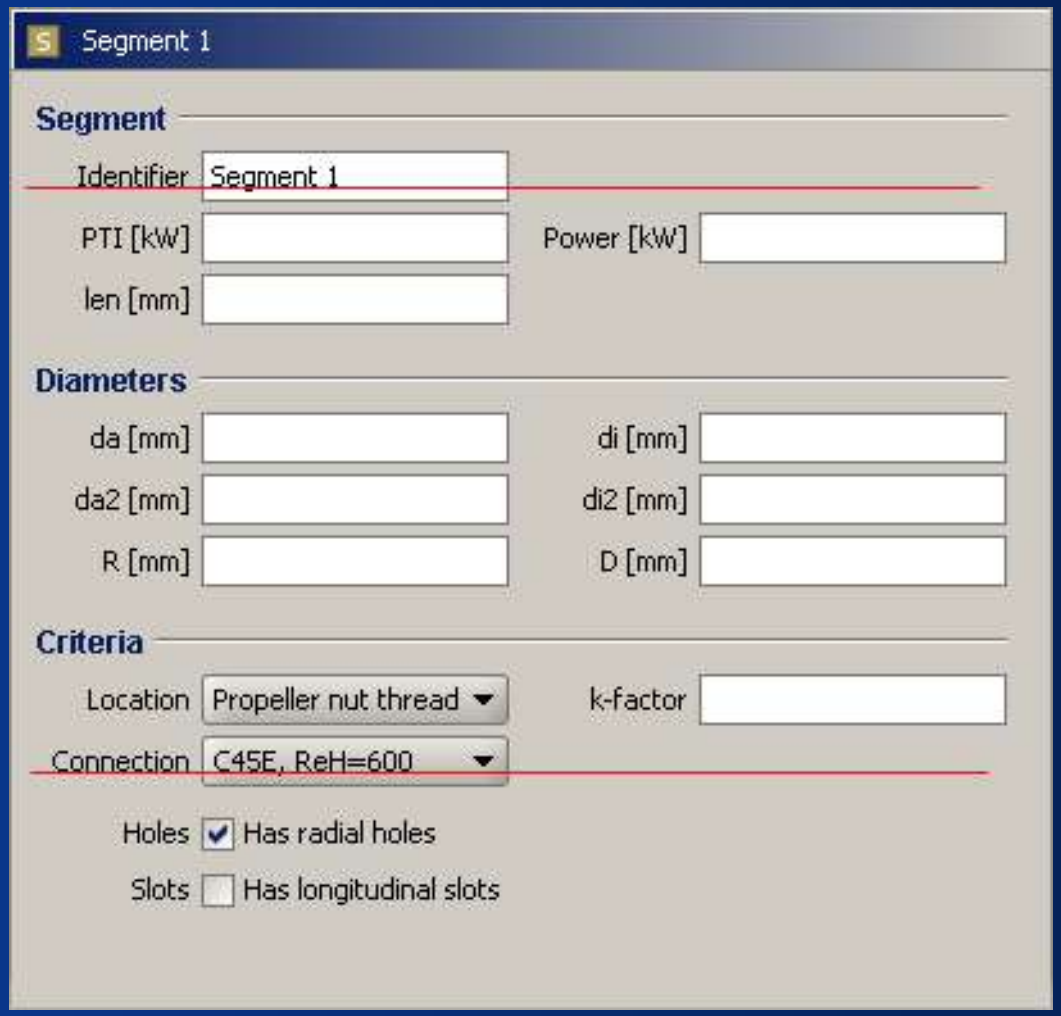
Gleiche Breiten an



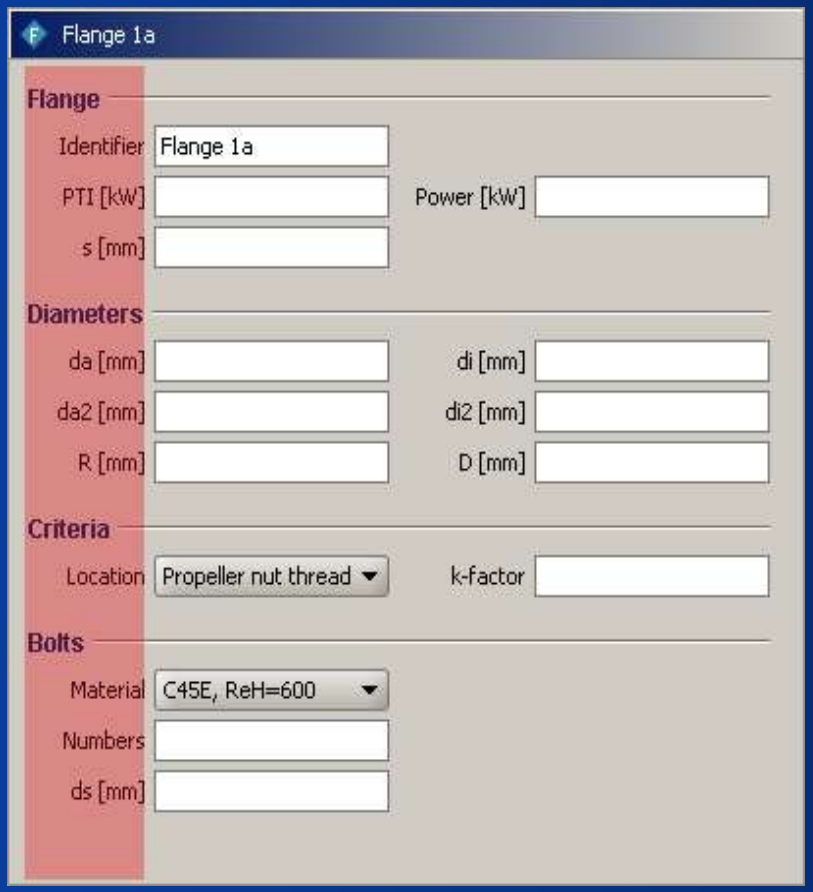
Gleiche Höhen an

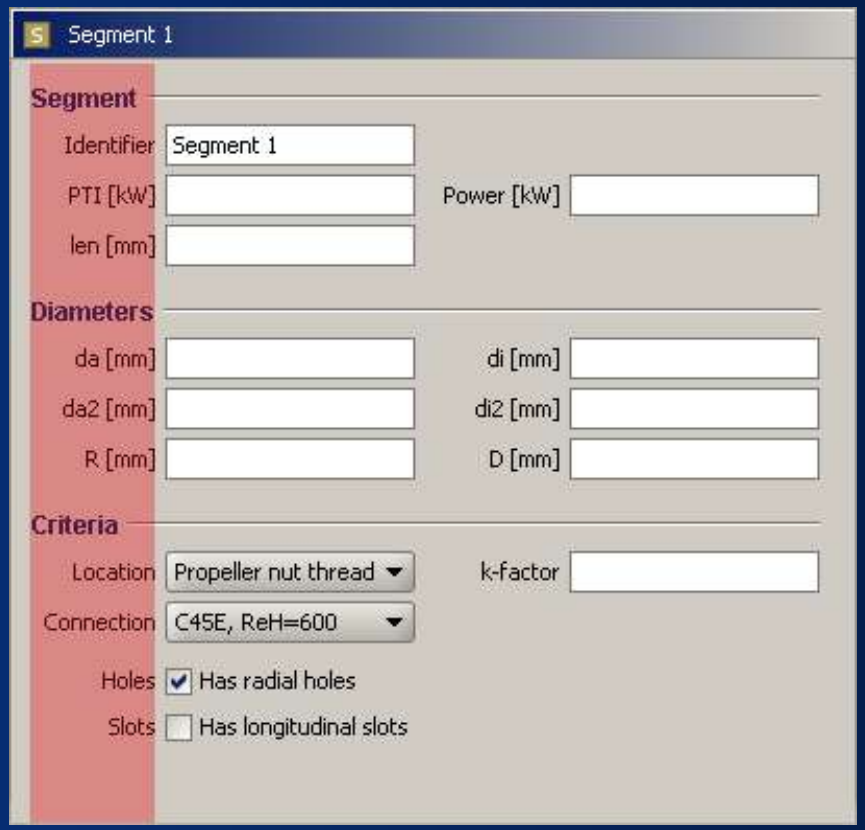


Richte Grundlinien aus



Gleiche Panel an



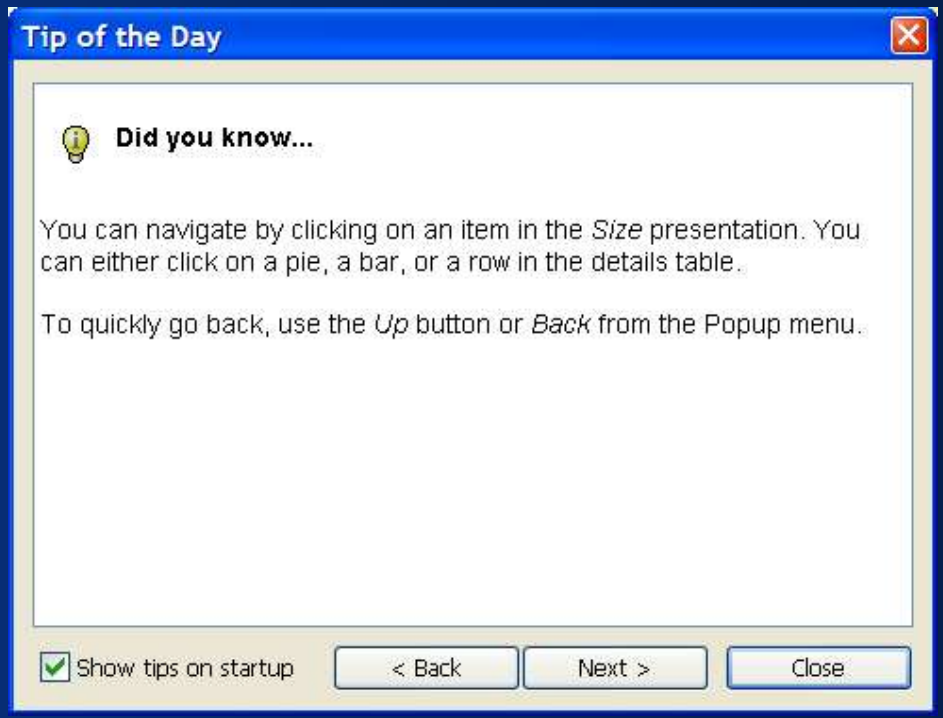


Beachte minimale Größen

- 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

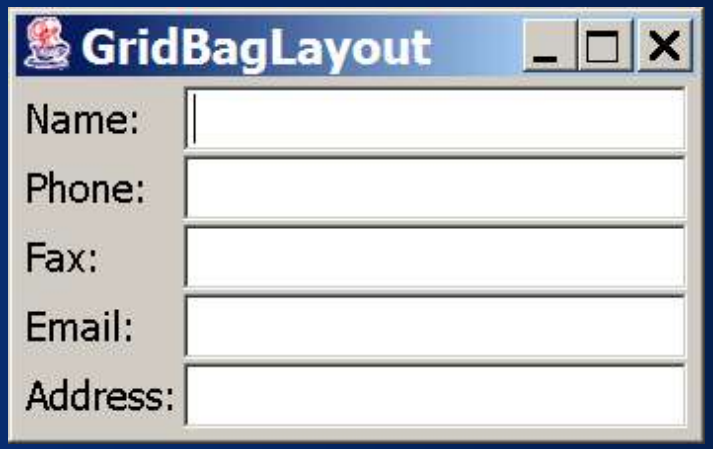


120dpi

Keine Pixel in Screen Design!

Proportionen gehen sonst verloren

96dpi



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
- Grafikdesigner 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

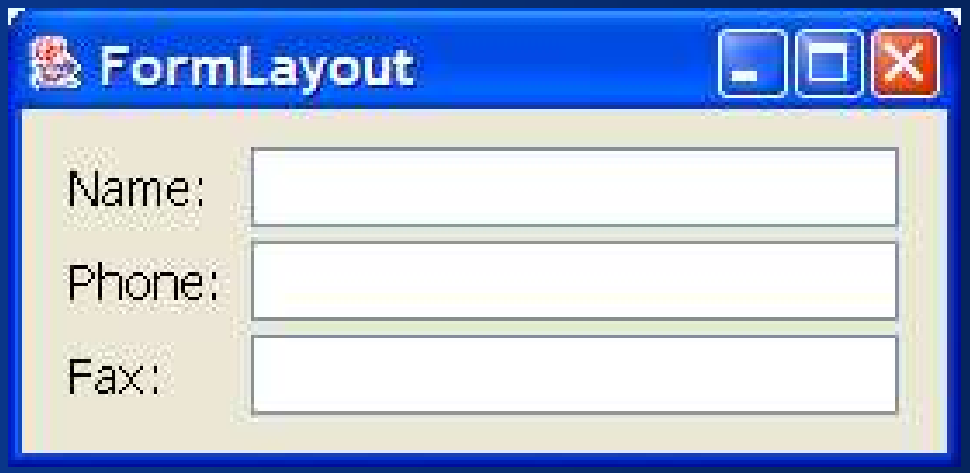
Lösung: JGoodies Forms

- 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?



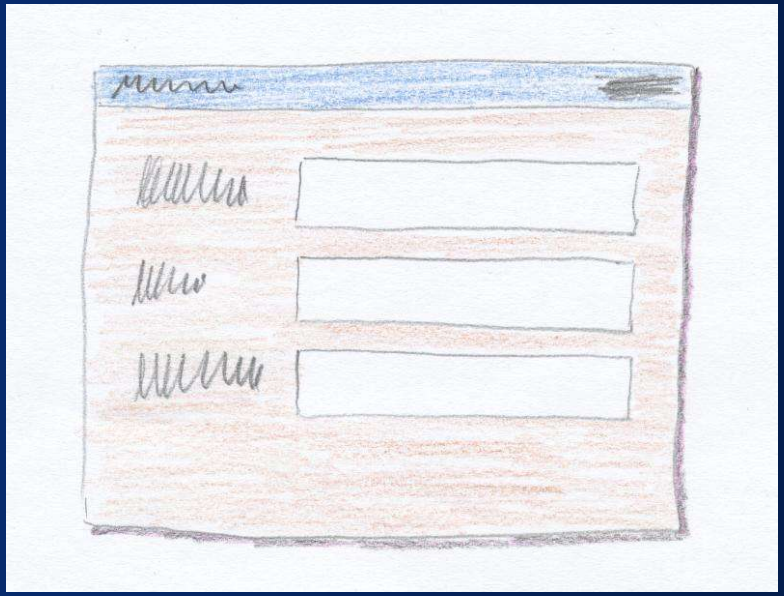
1: Anforderungen

Chef sagt:

„Wir brauchen eine Maske,
um eine Adresse einzugeben.
Die soll Felder haben für:
Name, Telefon und Email.“

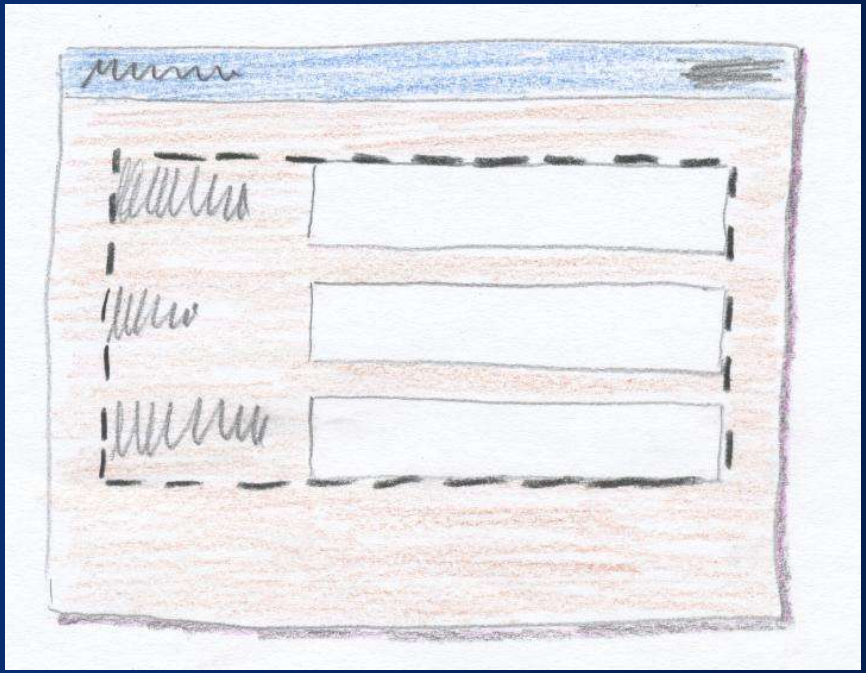
2: *Layout finden*

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



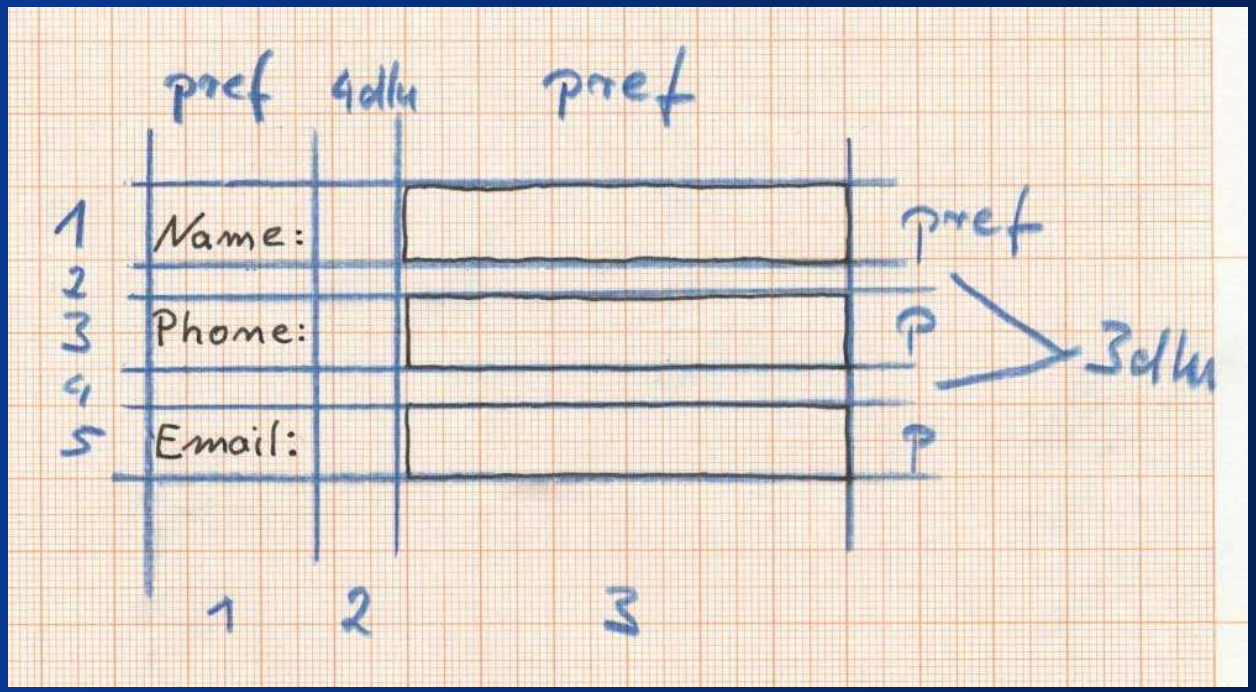
3a: Auf Inhalt konzentrieren

- 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 bequemem Builder

Mit DefaultFormBuilder

```
FormLayout layout = new FormLayout(
    "l: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

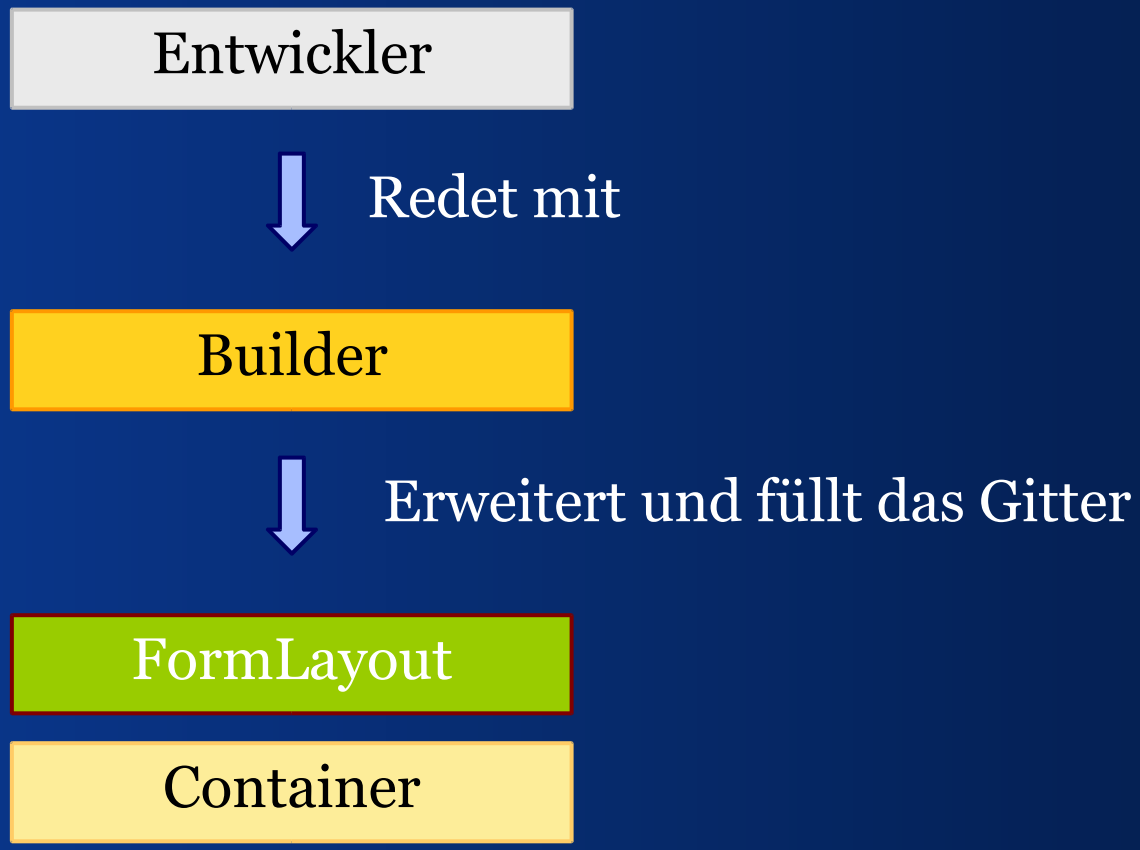
```
DefaultFormBuilder builder =
    new DefaultFormBuilder(layout);

builder.setDefaultDialogBorder();

...
```

Fabriken und Logische Größen

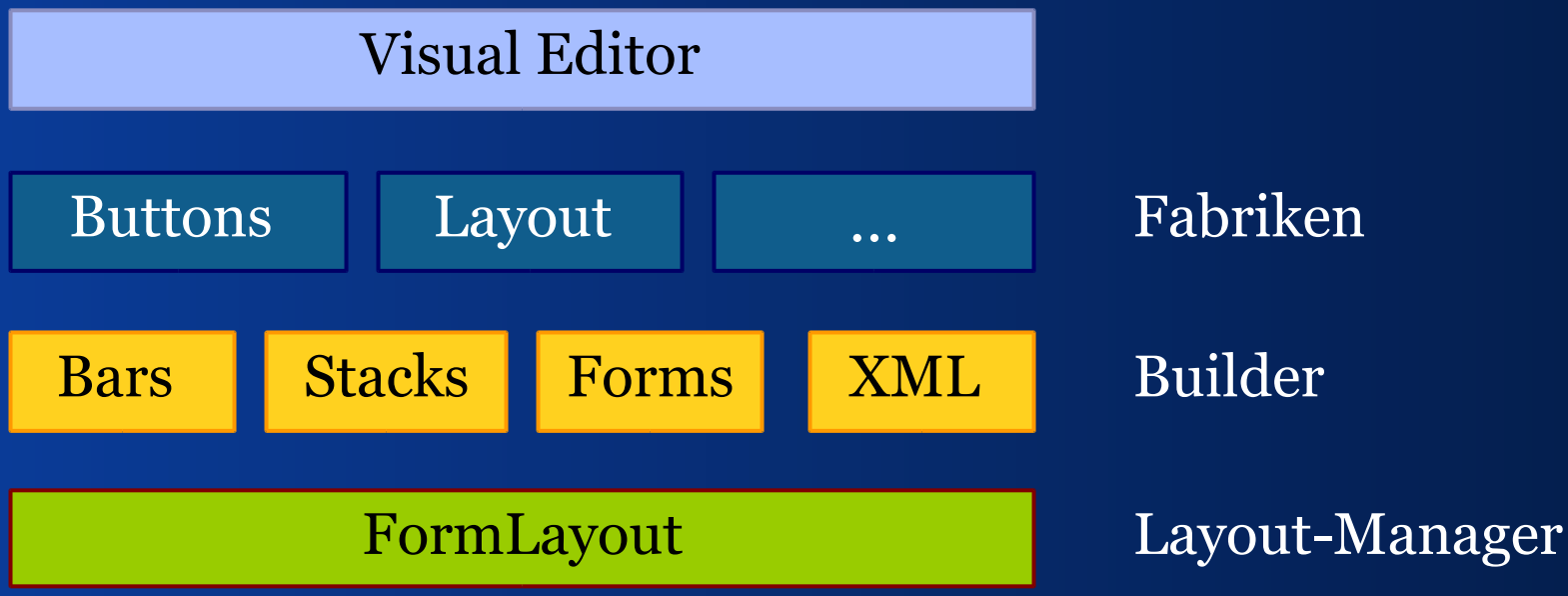
- Der **`ButtonBarBuilder`**:
  - baut konsistente Knopfleisten
  - beachtet den Plattform-Stil
  - nutzt logische Größen
  - unterstützt logisches Layout (Mac vs. PC)

- Die **`ButtonBarFactory`**:
  - liefert fertige Knopfleisten
  - nutzt logisches Layout (Mac vs. PC)

Nicht-visuelle Builder



Schichten in Forms



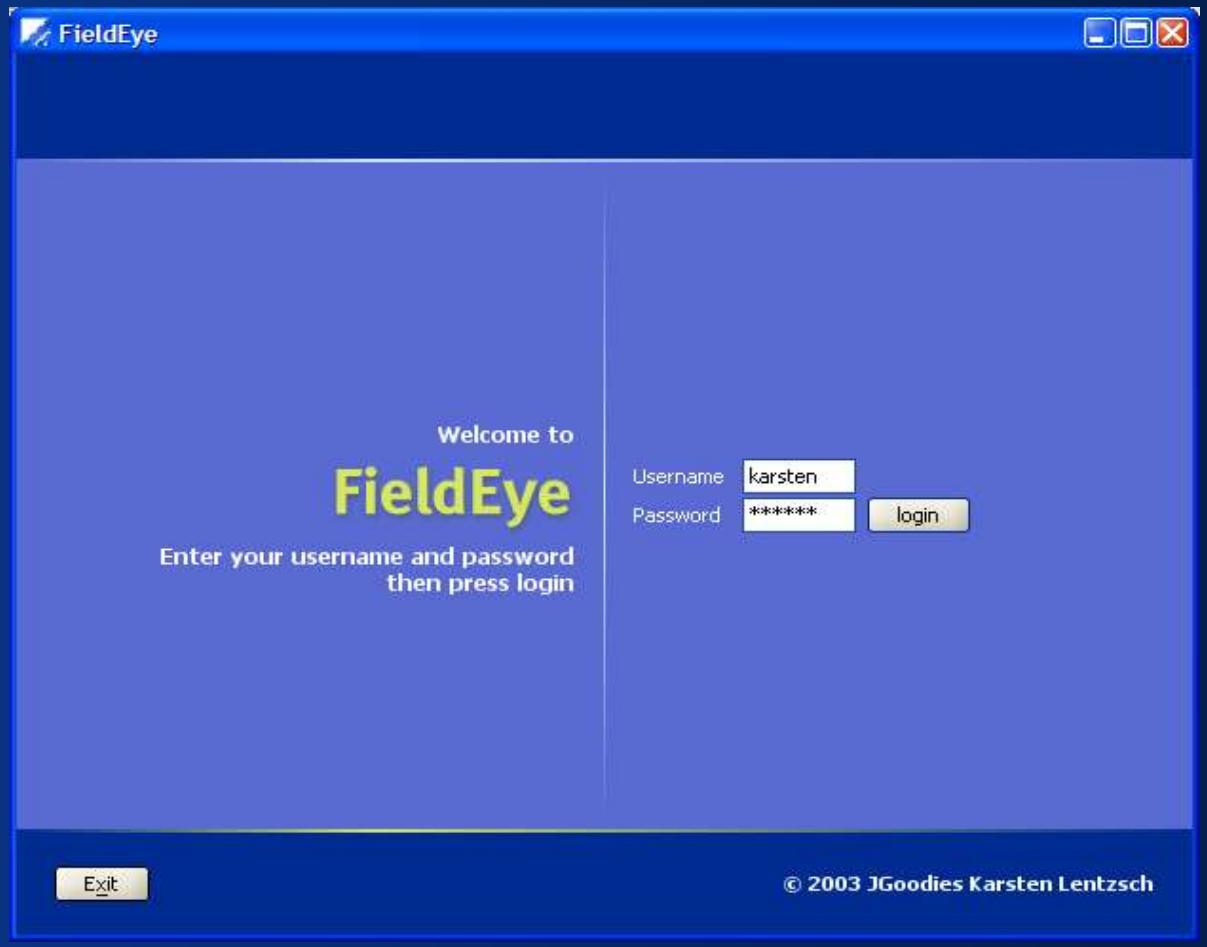
Demo

VI. Ausblick

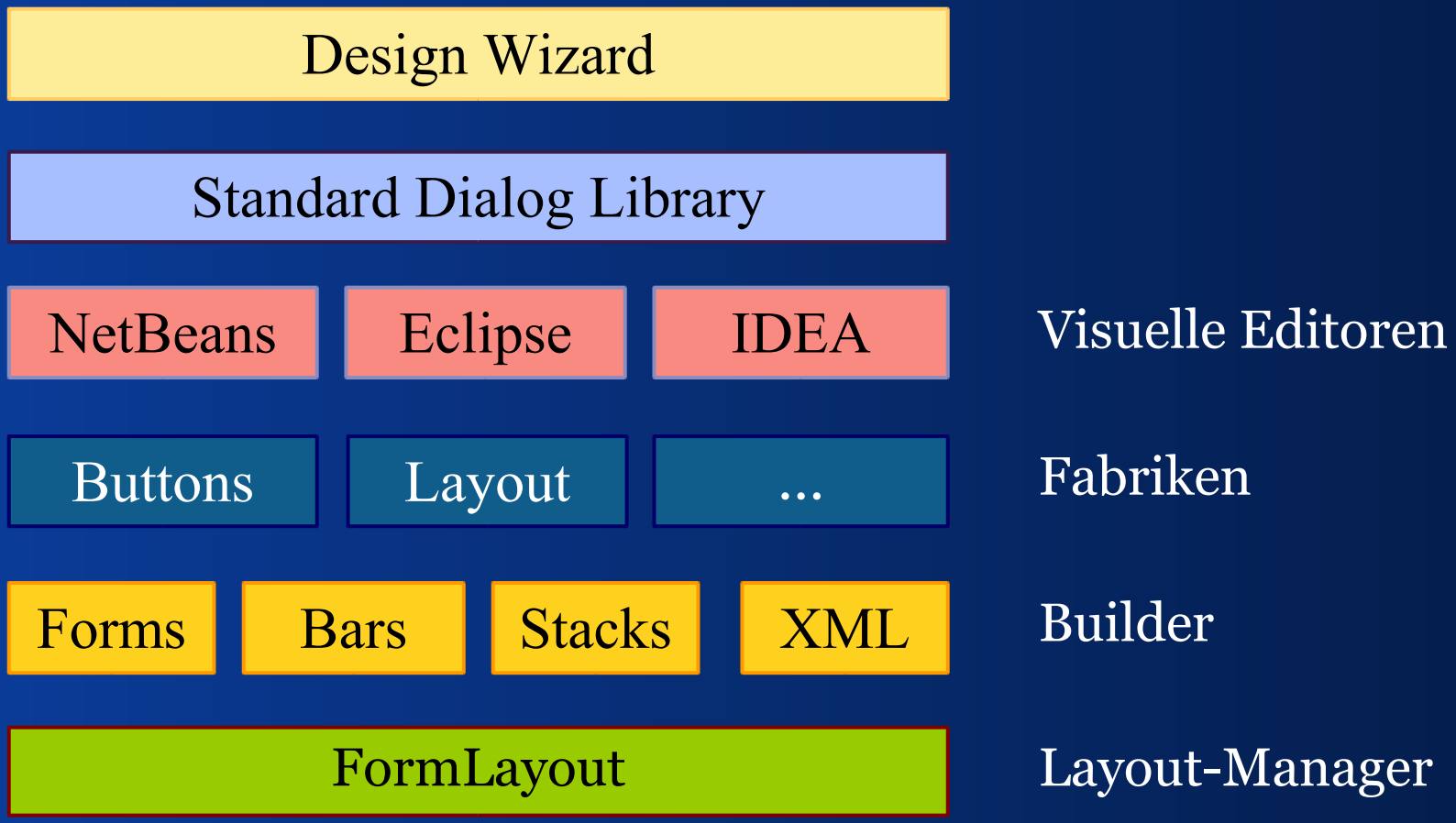
Was fehlt noch? Wie geht‘s weiter?

Wie kriegt man gute Gestaltung?

Der „typische” Entwickler geht Folgendes nicht an:



Vorlagen und Wizard



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, auf 'library' klicken
- Aqua Human Interface Guidelines
  www.apple.com/developer
- JGoodies Forms-Rahmenwerk
  forms.dev.java.net

Ein guter Layout-Manager

- ExplicitLayout, siehe 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!